Refining the Internet Voting Protocol

Anders Smedstuen Lund

Teacher Education with Master of Science
Submission date: June 2011
Supervisor: Kristian Gjøsteen, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

ABSTRACT

We make two improvements to the Internet voting protocol written by Gjøsteen [5]. The first improvement improves the performance of the protocol, by changing the encryption of the votes. The second improvement improves the security of the protocol, by removing a private key used in the original protocol. The second improvement is done to the protocol after the first improvement has been implemented, so we end up with a protocol where both improvements are implemented.

ACKNOWLEDGEMENTS

You are now looking at what represents the work of my last semester at NTNU, my master thesis. There are many persons that deserves my thanks, from former teachers, to fellow students, to professors at NTNU.

First of all I would like to thank my supervisor, Kristian Gjøsteen. Without your supervision, encouragement and helpfulness I would never have been close to writing half of this thesis.

I would also like to thank three of my fellow students in particular. Vidar Sandstad, I owe you a great tanks for reading through my thesis and pointing out my wraiting erors. Torkil Utvik Stai, thank you for all the hours of work together, and for all the help on different subjects. And Marianne Wiik Øberg, thank you for all our discussions on different aspects of my thesis, and for always having a moment to help.

To all my friends I want to say thank you for all good hours I have spent with you. For all the fun and social times, and for all support. You are truly great.

Lastly, to my family. Without your encouragement and support I could never have finished neither my thesis nor my studies. For that I am ever grateful.

CONTENTS

CHAPTER 1

INTRODUCTION

In 2011, Norway is going to run a trial on Internet voting in the local government election. For this purpose, Kristian Gjøsteen has described and analysed a cryptographic protocol [5] to be used for these trial elections, hereafter called the Internet voting protocol. The protocol is designed by a Spanish company named Scytl.

In my thesis I have done two improvements to the Internet voting protocol. One of them is to improve the performance of the protocol by changing the way we encrypt ballots. The other is to improve the security of the protocol by removing a connection between the three private keys used for encryption and decryption. When doing this last improvement, I have used the first improvement as starting point, to end up with a protocol where both changes are implemented.

Chapter 2 contains some general definitions and theory, and some calculations on probabilities. In Chapter 3 we describe the protocol, make a general assumption and make the changes of the first improvement to what Gjøsteen has called the simplified protocol. Then we end the chapter by analysing the simplified protocol with the new changes. The first improvement implemented into what Gjøsteen refers to as the full protocol (this is what we refer to as the protocol) is found in Chapter 4, this chapter also contains a new analysis of this protocol with the new changes made. Chapter 5 describes two small protocols needed for the second improvement, and some proofs that these two protocols fulfil some properties we want. In Chapter 6, we give the new versions of the different parts in the protocol with both improvements implemented. Then we analyse the new protocol with both improvements implemented in Chapter 7. Lastly Chapter 8 summarizes the thesis by looking at what gain we got from the first improvement, and we also look at how the second improvement has effected the performance.

In this paper I will assume the knowledge of basic group theory, general cryptography and basic probability theory.

CHAPTER 2

THEORY

In this chapter we give some general terminology and theory which we will make use of in later chapters to make both improvements. First we relate a problem we want to prove is difficult to solve, to a known problem which it is believed is difficult to solve, the Decision Diffie-Hellman problem (DDH-problem). After that we will define a special protocol and some other general theory that we will need for our second improvement.

2.1 Subgroup Membership Problems

The Decision Diffie-Hellman problem was first described in [1]. Another version, more suitable for our work, is described in [2]. We define the problem as follows:

Definition 2.1. Let G be a group of prime order. Given the tuple $(g, \alpha, \beta, \gamma)$ where g and α are random elements, *the DDH-problem* is now to decide if $(\beta, \gamma) \in K$ where $K = \langle (g, \alpha) \rangle$.

Another perspective on the DDH-problem is to look at it as a subset membership problem which is also described in the same article. We define it only for groups, and call it a subgroup membership problem:

Definition 2.2. Given a finite abelian group G, and a proper non-trivial subgroup K. *The subgroup membership problem* now is to decide if an element x in G is in K or $G \setminus K$

From now on we denote a subgroup membership problem as follows: if G is the group and K the subgroup, then we denote the problem $G \overset{?}{\leftrightarrow} K$. We now make a simulator S and an attacker A that plays a game between them relating to $G \overset{?}{\leftrightarrow} K$. Let both know what the group and the subgroup is. Then S draws an element b randomly from $\{0, 1\}$. If $b = 0$, S generates a random element from K and sends to A. If $b = 1$, S generates a random element from G, and sends to A. The attacker is now supposed to decide what the value of b is and send it to S as $\check{b}$. A wins if $\check{b} = b$. We now define the advantage of an attacker A in this game:

Definition 2.3. Define E to be the event that A guesses correctly, i.e that $\check{b} = b$. We now define *the advantage of an attacker* in this game as follows:

$$\mathrm{Adv}_{\mathrm{A}} = \left| \Pr[E] - \frac{1}{2} \right|$$

The reason we subtract a half is that any attacker can always guess, and thereby get a probability of guessing of a half (we want to find out how much better than a guess an attacker can do). Since we are always working with a finite group we can define drawing randomly from G and randomly from K as drawing from G with different probability distributions.

When we are drawing randomly from G we are drawing from G with uniform distribution. When we are drawing randomly from K, we are drawing from G with equal probability of drawing any element from K and zero probability of drawing any element from $G \setminus K$. We denote drawing from G with uniform distribution by $x \overset{r}{\leftarrow} X_1$, and drawing from G with the other distribution is denoted by $x \overset{r}{\leftarrow} X_0$.

In the case where we have a subgroup of K, call it L, we want to consider the three possible subgroup membership problems $G \overset{?}{\leftrightarrow} K$, $G \overset{?}{\leftrightarrow} L$ and $K \overset{?}{\leftrightarrow} L$. Then drawing from G with all elements of L having equal probability of being drawn, and all other elements having probability zero of being drawn, is denoted by $x \overset{r}{\leftarrow} X_0$. Drawing from G with equal probability of drawing any element from K, and zero probability of drawing any other element, will be denoted $x \overset{r}{\leftarrow} X_1$. Lastly drawing from G with uniform probability distribution we in this case denote $x \overset{r}{\leftarrow} X_2$. If we have even longer sequences of subgroups we denote it in the corresponding way.

By $A(x)$ we mean the answer of the attacker A on input x. We now define the following probabilities (for the case $G \overset{?}{\leftrightarrow} K$):

$$\mu_{00} = \Pr[A(x) = 0 \mid x \overset{r}{\leftarrow} X_0]$$
$$\mu_{01} = \Pr[A(x) = 0 \mid x \overset{r}{\leftarrow} X_1]$$
$$\mu_{10} = \Pr[A(x) = 1 \mid x \overset{r}{\leftarrow} X_0]$$

$$\mu_{11} = \Pr[A(x) = 1 \mid x \overset{r}{\leftarrow} X_1]$$

Correspondingly, when we have an additional subgroup of K, L, we get $\mu_{02} = \Pr[A(x) = 0 \mid x \overset{r}{\leftarrow} X_2]$, etc.

2.2 Subgroup Membership Problems in Three Variables

The goal of this section is to show that given an attacker with an advantage for the problem $G^3 \overset{?}{\leftrightarrow} H_0$, the attacker also has an advantage on the DDH-problem, where $H_0 = \langle(g_1, g_2, g_3)\rangle$ and g_1, g_2 and g_3 are generators of G. We will also use the subgroup H_1. We define H_1 as $H_1 = \langle(g_1, g_2, g_3), (g_1, g_2, 1)\rangle$. We assume known to us, not any attacker, that $g_2 = g_1^{k_1}$ (for some number k_1) and $g_3 = g_2^{k_2}$ (for some number k_2).

We now show that given a subgroup membership problem $G^2 \overset{?}{\leftrightarrow} K$, we get the following probabilities when playing the game discussed in the previous section:

$$\begin{aligned}\Pr[A(x) = b] &= \frac{1}{2}Pr[A(x) = 0 \mid x \overset{r}{\leftarrow} X_0] + \frac{1}{2}Pr[A(x) = 1 \mid x \overset{r}{\leftarrow} X_1] \\ &= \frac{1}{2}\mu_{00} + \frac{1}{2}\mu_{11} \\ &= \frac{1}{2}\mu_{00} + \frac{1}{2} - \frac{1}{2}\mu_{01}\end{aligned}$$

So it follows that we get:

$$Adv_A = |\frac{1}{2}\mu_{00} + \frac{1}{2} - \frac{1}{2}\mu_{01} - \frac{1}{2}| = \frac{1}{2}|\mu_{00} - \mu_{01}|$$

We now look at the $G^3 \overset{?}{\leftrightarrow} H_0$ problem. To avoid confusion, we now use Y instead of X when speaking about drawing from G^3 and its subgroups; i.e. when drawing an element randomly from G^3 with uniform distribution, we denote it by $x \overset{r}{\leftarrow} Y_2$.

We have the problems $G^3 \overset{?}{\leftrightarrow} H_0$, $H_1 \overset{?}{\leftrightarrow} H_0$ and $G^3 \overset{?}{\leftrightarrow} H_1$. These give us three games of the type discussed in the previous section, with respectively Y_2 and Y_0, Y_1 and Y_0 and Y_2 and Y_1 as sets with given probability distributions. We further denote the advantage of an attacker in the three different games by $\mathrm{Adv}_A^{Y_2/Y_0}$, $\mathrm{Adv}_A^{Y_2/Y_1}$ and $\mathrm{Adv}_A^{Y_1/Y_0}$. The variables μ_{00} etc. are denoted accordingly. We now wish to say something about the relationship between $\mathrm{Adv}_A^{Y_2/Y_0}$ and the two other advantages, $\mathrm{Adv}_A^{Y_2/Y_1}$ and $\mathrm{Adv}_A^{Y_1/Y_0}$.

Lemma 2.1. *Having an advantage of* ϵ *on* $\mathrm{Adv}_A^{Y_2/Y_0}$ *gives an advantage on either* $\mathrm{Adv}_A^{Y_1/Y_0}$ *or* $\mathrm{Adv}_A^{Y_2/Y_1}$ *of at least* $\epsilon/2$.

Proof. Earlier in this section we have showed that $2\mathrm{Adv}_A^{Y_2/Y_0} = |\mu_{00} - \mu_{02}|$. We now get:

$$\begin{aligned} 2\mathrm{Adv}_A^{Y_2/Y_0} &= |\mu_{00} - \mu_{02}| \\ &= |\mu_{00} - \mu_{01} + \mu_{01} - \mu_{02}| \\ &\leq |\mu_{00} - \mu_{01}| + |\mu_{01} - \mu_{02}| \\ &= 2\mathrm{Adv}_A^{Y_1/Y_0} + 2\mathrm{Adv}_A^{Y_2/Y_1}. \end{aligned}$$

So,

$$\mathrm{Adv}_A^{Y_2/Y_0} \leq \mathrm{Adv}_A^{Y_2/Y_1} + \mathrm{Adv}_A^{Y_1/Y_0}$$
$$\Downarrow$$
$$\text{If } \mathrm{Adv}_A^{Y_2/Y_0} = \epsilon \text{ then } \mathrm{Adv}_A^{Y_1/Y_0} \geq \epsilon/2 \text{ or } \mathrm{Adv}_A^{Y_2/Y_1} \geq \epsilon/2.$$

□

Definition 2.4. Define $F_1 : G^2 \longrightarrow G^3$ by:

$$\begin{cases} F_1 : & G^2 & \longrightarrow & G^3 \\ & (x_1, x_2) & \longmapsto & (x_1^t, x_1^{k_1 t}, x_2^{k_2 t}) \end{cases}$$

and $F_2 : G^2 \longrightarrow G^3$ by:

$$\begin{cases} F_2 : & G^2 & \longrightarrow & G^3 \\ & (x_1, x_2) & \longmapsto & (x_1^t, x_2^t, r) \end{cases}$$

Where $t \xleftarrow{r} \mathbb{Z}_{|G|}$ and $r \xleftarrow{r} G$

Remark: F_1 and F_2 are not functions. Since we use a random exponent t every time, we can get different outputs on the same input.

Lemma 2.2. F_1 *sends the tuple* (x_1, x_2) *into a random element from* H_0 *if the tuple is of the form* $(x_1, x_2) = (g_1^s, g_2^s)$ *and into a random element from* $H_1 \setminus H_0$ *otherwise. Similarly* F_2 *sends the tuple* (x_1, x_2) *into a random element from* H_1 *if the tuple is of the stated form, and into a random element from* $G^3 \setminus H_1$ *otherwise.*

Proof. We start with the case of F_1. Assume $(x_1, x_2) = (g_1^s, g_2^s)$, then $F_1((x_1, x_2))$ $= (x_1^t, x_1^{k_1 t}, x_2^t) = (g_1^{st}, g_1^{k_1 st}, g_2^{k_2 st}) = (g_1^{st}, g_2^{st}, g_3^{st}) \in H_0$. The exponent t is a random number from $\mathbb{Z}_{|G|}$, hence x_1^t is a random element of G. The element in the first coordinate of a tuple from H_0 uniquely determines the second and third coordinate, hence if F_1 sends into H_0 it is sent to a random element of H_0.

Now assume there does not exist a number s such that $(x_1, x_2) = (g_1^s, g_2^s)$. Then $F_1((x_1, x_2)) = (x_1^t, x_1^{k_1 t}, x_2^{k_2 t}) \neq (g_1^{st}, g_1^{k_1 st}, g_2^{k_2 st})$ so $F_1((x_1, x_2)) \notin H_0$, but clearly we always have $F_1((x_1, x_2)) \in H_1$. So F_1 sends into $H_1 \setminus H_0$. Since t is a random number, x_1^t is a random element from G. The first coordinate of a tuple from H_1 uniquely determines the second coordinate, and now x_2 is independent of x_1, and x_2^t is a random element of G, hence the third coordinate is a random element from G independent of the two others. So when F_1 sends into $H_1 \setminus H_0$, it sends to a random element of $H_1 \setminus H_0$.

Now we look at the case of F_2. Assume $(x_1, x_2) = (g_1^s, g_2^s)$, then $F_2((x_1, x_2)) = (x_1^t, x_2^t, r) = (g_1^{st}, g_2^{st}, r) \in H_1$. The exponent t is a random number from the set $\mathbb{Z}_{|G|}$. So x_1^t is a random element of G, and r is by definition a random element of G independent of the two others. Further if F_2 sends into H_1 then the element in the first coordinate uniquely determines the element in the second coordinate, hence we get a random element from H_1.

Lastly assume that $(x_1, x_2) \neq (g_1^s, g_2^s)$ for all numbers $s \in \mathbb{Z}_{|G|}$. Then $F_2((x_1, x_2)) = (x_1^t, x_2^t, r) \neq (g_1^{st}, g_2^{st}, r)$, so we have $F_2((x_1, x_2)) \notin H_1$, but clearly $F_2((x_1, x_2)) \in G^3$. Still t is a random number, hence x_1^t and x_2^t are random elements from G. Since $(x_1, x_2) \neq (g_1^s, g_2^s)$, x_1^t and x_2^t are also independent from each other. The third coordinate is also a random element chosen independently from the two other coordinates, hence we have three random elements independent of each other, and therefore a random element from G^3. □

By Lemma 2.2, we get the following equalities:

$$\begin{aligned}&\Pr[(y_1, y_2, y_3) = (r_1, r_2, r_3) \mid (y_1, y_2, y_3) \xleftarrow{r} Y_b] \\ &\quad = \Pr[F_1((x_1, x_2)) = (r_1, r_2, r_3) \mid (x_1, x_2) \xleftarrow{r} X_b]\end{aligned}$$

and

$$\begin{aligned}&\Pr[(y_1, y_2, y_3) = (r_1, r_2, r_3) \mid (y_1, y_2, y_3) \xleftarrow{r} Y_{b'}] \\ &\quad = \Pr[F_2((x_1, x_2)) = (r_1, r_2, r_3) \mid (x_1, x_2) \xleftarrow{r} X_b]\end{aligned}$$

Where $b \in \{0, 1\}$, $b' = 1$ if $b = 0$ and $b' = 2$ if $b = 1$.

Lemma 2.3. *If we denote the attacker on* $G^2 \overset{?}{\leftrightarrow} K$ *by* B *and the attacker on* $G^3 \overset{?}{\leftrightarrow} H_1$ *or* $H_1 \overset{?}{\leftrightarrow} H_0$ *by* A *then,* $\mathrm{Adv}_A^{Y_1/Y_0} = \mathrm{Adv}_B^{X_1/X_0}$ *and* $\mathrm{Adv}_A^{Y_2/Y_1} = \mathrm{Adv}_B^{X_1/X_0}$.

Proof.
$\mathbf{Adv}_{\mathbf{A}}^{\mathbf{Y_1/Y_0}} = \mathbf{Adv}_{\mathbf{B}}^{\mathbf{X_1/X_0}}$:
We define what the attacker B does and show we get the first claimed equality.

```
$B((x_1, x_2))$ :
  $(y_1, y_2, y_3) = F_1((x_1, x_2))$
  $\check{b} = A((y_1, y_2, y_3))$
  return $\check{b}$
```

Now we prove that we get the desired equality.

$$\begin{aligned}
2\mathrm{Adv}_B^{X_1/X_0} &= |Pr[B((x_1, x_2)) = 0 \mid (x_1, x_2) \overset{r}{\leftarrow} X_0] \\
&\quad -Pr[B((x_1, x_2)) = 0 \mid (x_1, x_2) \overset{r}{\leftarrow} X_1]| \\
&= |Pr[A((y_1, y_2, y_3)) = 0 \mid \\
&\qquad (y_1, y_2, y_3) = F_1((x_1, x_2)), (x_1, x_2) \overset{r}{\leftarrow} X_0] \\
&\quad -Pr[A((y_1, y_2, y_3)) = 0 \mid \\
&\qquad (y_1, y_2, y_3) = F_1((x_1, x_2)), (x_1, x_2) \overset{r}{\leftarrow} X_1]| \\
&= |Pr[A((y_1, y_2, y_3)) = 0 \mid (y_1, y_2, y_3) \overset{r}{\leftarrow} Y_0] \\
&\quad -Pr[A((y_1, y_2, y_3)) = 0 \mid (y_1, y_2, y_3) \overset{r}{\leftarrow} Y_1]| \\
&= 2\mathrm{Adv}_A^{Y_1/Y_0}
\end{aligned}$$

$\mathbf{Adv}_{\mathbf{A}}^{\mathbf{Y_2/Y_1}} = \mathbf{Adv}_{\mathbf{B}}^{\mathbf{X_1/X_0}}$:
The proof of this case is similar to the other. The attacker B does exactly as before, but uses F_2 instead of F_1. We then get the following probability calculations:

$$\begin{aligned}
2\mathrm{Adv}_B^{X_1/X_0} &= |Pr[B((x_1, x_2)) = 0 \mid (x_1, x_2) \overset{r}{\leftarrow} X_0] \\
&\quad -Pr[B((x_1, x_2)) = 0 \mid (x_1, x_2) \overset{r}{\leftarrow} X_1]| \\
&= |Pr[A((y_1, y_2, y_3)) = 0 \mid \\
&\qquad (y_1, y_2, y_3) = F_2((x_1, x_2)), (x_1, x_2) \overset{r}{\leftarrow} X_0] \\
&\quad -Pr[A((y_1, y_2, y_3)) = 0 \mid \\
&\qquad (y_1, y_2, y_3) = F_2((x_1, x_2)), (x_1, x_2) \overset{r}{\leftarrow} X_1]| \\
&= |Pr[A((y_1, y_2, y_3)) = 0 \mid (y_1, y_2, y_3) \overset{r}{\leftarrow} Y_1] \\
&\quad -Pr[A((y_1, y_2, y_3)) = 0 \mid (y_1, y_2, y_3) \overset{r}{\leftarrow} Y_2]| \\
&= 2\mathrm{Adv}_A^{Y_2/Y_1}
\end{aligned}$$

□

By combining the lemmas in this section we get the following theorem,

Theorem 2.4. *An advantage ϵ on $\mathrm{Adv}_A^{Y_2/Y_0}$ gives an advantage on $\mathrm{Adv}_B^{X_1/X_0}$ (DDH) of at least $\epsilon/2$.*

Proof. Assume $\mathrm{Adv}_A^{Y_2/Y_0} = \epsilon$, by Lemma 2.1 either $\mathrm{Adv}_A^{Y_1/Y_0} \geq \epsilon/2$ or $\mathrm{Adv}_A^{Y_2/Y_1} \geq \epsilon/2$. By Lemma 2.3 we get $\mathrm{Adv}_A^{X_1/X_0} = \mathrm{Adv}_A^{Y_1/Y_0} \geq \epsilon/2$ or $\mathrm{Adv}_A^{X_1/X_0} = \mathrm{Adv}_A^{Y_2/Y_1} = \epsilon/2$, so $\mathrm{Adv}_A^{X_1/X_0} \geq \epsilon/2$. □

2.3 Subgroup Membership Problems in More Than Three Variables

In this section we generalize what we did in the previous section into several variables. We want to prove that an advantage on a subgroup membership problem with G^l, $l > 3$, and a subgroup of it gives an advantage on DDH.

Firstly we now get several more subgroups. Let $H_0 = \langle(g_1, g_2, \ldots, g_l)\rangle$, $H_1 = \langle(g_1, g_2, \ldots, g_l), (g_1, g_2, \ldots, g_{l-1}, 1)\rangle, \ldots, H_{l-2} = \langle(g_1, \ldots, g_l), (g_1, \ldots, g_{l-1}, 1), \ldots, (g_1, g_2, 1, \ldots, 1)\rangle$ and $H_{l-1} = G^l$. We assume that it is known to us, not any attacker, that $g_2 = g_1^{u_1}$, $g_3 = g_2^{u_2}, \ldots, g_l = g_{l-1}^{u_{l-1}}$. We continue using Y instead of X when talking about drawing from G^l and its subgroups; i.e. when drawing with uniform distribution from H_i we denote it $x \overset{r}{\leftarrow} Y_i$.

The lemmas in this section are given without proof since they are analogous to the proofs of the three variable cases done in the previous section.

Lemma 2.5. *An advantage on* $\mathrm{Adv}_A^{Y_{l-1}/Y_0}$ *gives an advantage on* $\mathrm{Adv}_A^{Y_1/Y_0}$, $\mathrm{Adv}_A^{Y_2/Y_1}, \ldots, \mathrm{Adv}_A^{Y_{l-2}/Y_{l-3}}$ *or* $\mathrm{Adv}_A^{Y_{l-1}/Y_{l-2}}$ *of at least* $\epsilon/(l-1)$

Proof. Similar to the proof of Lemma 2.1. □

Definition 2.5. Define $F_i : G^2 \longrightarrow G^l$ by:

$$\begin{cases} F_i : & G^2 & \longrightarrow & G^l \\ & (x_1, x_2) & \longmapsto & (x_1^{u_0 t}, x_1^{u_0 u_1 t}, \ldots, x_1^{t \prod_{j=0}^{l-(i+1)} u_j}, x_2^{t \prod_{j=1}^{l-i} v_j}, r_{l-i+2}, \ldots, r_l) \end{cases}$$

$i \in \{1, \ldots, l-1\}, t \overset{r}{\leftarrow} \mathbb{Z}_{|G|}, r_i \overset{r}{\leftarrow} G, v_j = u_j$ for $j \in \{2, \ldots, l-1\}, v_1 = u_0 = 1$

Lemma 2.6. *$F_i((x_1, x_2))$ sends the tuple into a random element from H_{i-1} if it is of the form $(x_1, x_2) = (g_1^t, g_2^t)$ and into a random element from $H_i \setminus H_{i-1}$ otherwise, $i \in \{1, \ldots, l-1\}$, where $H_{l-1} = G^l$.*

Proof. Similar to the proof of Lemma 2.2. □

By Lemma 2.6 we get the following equalities:

$$\begin{aligned} &\Pr[(y_1, \ldots, y_l) = (r_1, \ldots, r_l) \mid (y_1, \ldots, y_l) \overset{r}{\leftarrow} Y_{b'}] \\ &\quad = \Pr[F_i(x_1, x_2) = (r_1, \ldots, r_l) \mid (x_1, x_2) \overset{r}{\leftarrow} X_b] \end{aligned}$$

Where $b \in \{0,1\}$, $b' = i-1$ if $b=0$ and $b' = i$ if $b = 1$, $i \in \{1,\dots,l-1\}$.

Lemma 2.7. *If we denote the attacker on $G^2 \overset{?}{\leftrightarrow} K$ by B and the attacker on one of $H_1 \leftrightarrow H_0, \dots, G^l \overset{?}{\leftrightarrow} H_{l-2}$ by A, then* $\mathrm{Adv}_A^{Y_1/Y_0} = \mathrm{Adv}_B^{X_1/X_0}$, $\mathrm{Adv}_A^{Y_2/Y_1} = \mathrm{Adv}_B^{X_1/X_0}, \dots, \mathrm{Adv}_A^{Y_{l-1}/Y_{l-2}} = \mathrm{Adv}_B^{X_1/X_0}$.

Proof. Similar to the proof of Lemma 2.3. □

Theorem 2.8. *An advantage ϵ on* $\mathrm{Adv}_A^{Y_{l-1}/Y_0}$ *gives an advantage on* $\mathrm{Adv}_B^{X_1/X_0}$ *(DDH) of at least $\epsilon/(l-1)$*

Proof. Similar to the proof of Theorem 2.4 □

2.4 Σ-protocols and Commitments

For the second change we are going to propose, we are going to need some theory on Σ-protocols and commitments. We define what they are in this section.

Σ-protocols

We are now going to define what a Σ-protocol is. A Σ-protocol is defined for a relation R. The relations R we are going to use is a subset of $G^{2i} \times \mathbb{Z}_q$, where $q, i \in \mathbb{Z}^+$. The relation is such that for $(v,w) \in R$, $x = (\vec{r}, \vec{s})$ and w is a witness such that $(\vec{r})^w = \vec{s}$. It is important that $\vec{r}$ and $\vec{s}$ has the same number of coordinates, namely i coordinates. To make a Σ-protocol, the protocol must be of a three move from. A three move from just means that first the prover sends a message to the verifier, the verifier replies with a challenge, before the prover comes up with a answer. We now define a Σ-protocol. The definition is found in [3]:

Definition 2.6. A protocol $\mathcal{P}$ is said to be a *Σ-protocol* for relation R if:

- $\mathcal{P}$ is of the above 3-move form, and we have completeness: if P, V follow the protocol on input x and private input w to P where $(x,w) \in R$, the verifier accepts.
- From any x and any pair of accepting conversations on input x, (a,e,z), (a,e',z') where $e \neq e'$, one can efficiently compute w such that $(x,w) \in R$. This is sometimes called the special soundness property.
- There exists a polynomial-time simulator M, which on input x and a random e outputs an accepting conversation of the form (a,e,z), with the same probability distribution as conversations between the honest P, V on input x. This is sometimes called special honest-verifier zero-knowledge.

In the same article it is also said that given a *random oracle* one can make the Σ-protocol non-interactive, i.e. the prover can make a satisfactory conversation before sending anything to the verifier and still not be able to cheat. A random oracle is a functionality that all players have access to, and can be viewed as a random function hidden in a black box such that non of the players can look inside of the box. It will always return the same answer on a given input, but given the answer of one input it is not possible to deduce something about the answer on another input. From a heuristic point of view you can view one-way hash functions as random oracles, but this cannot be formally proved.

Commitments

Commitments are described in [4]. A commitment scheme is a way of convincing another person that you have made a choice and committed to some secret (a number, a message, etc.) without having to reveal the actual secret when convincing the other person. A commitment scheme has two basic properties:

- Binding property: When you have committed to a secret, you may no longer change the secret. You have have made your choice, and cannot change that choice.
- Hiding property: When convincing the other person, you have to give him some sort of evidence that you have committed to some secret. The other person should not be able to find the secret based on the evidence he gets, even though it should be enough to convince him that you have made your choice.

Both the binding property and the hiding property can either be unconditional or computational. If either the hiding or the binding property is unconditional, it means that it holds no matter how much time or how much computing power you get to brake the property; i.e. if you have a unconditional binding commitment scheme, it is not even theoretically possible to change your choice after committing to the choice with this scheme.

If you have a computational hiding property or binding property it means that you can, given large enough resources, break the property, but the chances for doing so would be very small. The point about computational binding and hiding is that it is theoretically possible to change your mind or break the hiding property, but that it is very unlikely to happen. Damgård and Nielsen point out that unfortunately it is not possible with a commitment scheme that satisfies both unconditional hiding and unconditional binding, so one can at most make one of the properties unconditional.

We now define the commitment scheme we are going to use. First we must choose a random element y_c from the group G. Then a commitment to a message

m is defined as $com_{y_c}(m, r) = (g^r, y_c^r m) \in G \times G$, where $r \overset{r}{\leftarrow} \mathbb{Z}_{|G|}$. Sometimes we will also write $[m]_r$ for this commitment. The commitment scheme satisfies unconditional binding since when you have committed to m and r is chosen, there is only one r' such that $g^{r'} = g^r$, namely $r' = r$. And given r, m is uniquely determined. The commitment scheme also satisfies computational hiding. There are two ways of finding the message hidden in the commitment, either you find p such that $y_c = g^p$ or you find the exponent r. So if you are able to break the hiding property you should be able to get an advantage on the DDH-problem in G^2, which for the group we are going to use in the protocol is believed to be hard.

CHAPTER 3

SIMPLIFIED PROTOCOL

In this chapter we want to use what we have proved in Chapter 2 to make changes to what Gjøsteen has called the simplified protocol in [5]. The simplified protocol is found in Chapter 3 of his paper. Firstly, we will describe the different players in the original protocol and some assumptions we will make. Secondly, we will come up with three different choices for how to change the protocol, and give arguments for the choice we have made. Then we make the changes that are needed in the simplified protocol, before we analyse the security of the new simplified protocol in the last section.

3.1 The Different Players in the Protocol

The protocol consists of many players. We will call the collection of players that the voter has no control over "the system". The players we consider are the voter V, the voter's computer P, a key generation system KG, a ballot box B, a receipt generator R, a decryption service D and an auditor A. The voter and it's computer are the only players relating to the voter, all other players are part of the system.

Before the election the key generation functionality KG generates private and public keys and functions used under the election. The key generation functionality KG closes before the election starts. The private keys are a_1, a_2 and a_3, and we have corresponding public keys $y_1 = g^{a_1}$, $y_2 = g^{a_2}$ and $y_3 = g^{a_3}$ where g is a fixed generator in the group G that is to be used. The private key a_1 is sent to D, the private key a_2 is sent to B and the private key a_3 is sent to R. The ballot box is the part receiving encrypted votes from the voter's computer and storing them until counting, when they are sent to the decryption service. The ballot box also generates some messages that are sent to R.

The receipt generator is the part that generates codes to send directly to the voter (without going trough the voter's computer or the ballot box). At the moment it looks like SMS will be the way the codes will be sent. The point of generating the codes is to stop P from altering the ballots before sending them to the ballot box. The decryption service is responsible for decrypting the ciphertexts after receiving them from the ballot box when the election has closed. At last the auditor controls that the other players in the system do what they are supposed to do, simplified you can say that the auditor gets all the contents of the other players inside the system and check their computations.

From now on we assume that G, the group used in the protocol, is a multiplicative cyclic group of prime order.

3.2 What to Change?

In this section we propose three possible ways of changing the protocol, and then give arguments for why two of them would not be secure.

The change we want to make is how the voter's computer encrypts the ballots. Instead of having one secret key a_1 and one public key $y_1 = g^{a_1}$ used by the computer, we now want to have k_{max} different secret keys a_{1i} and k_{max} different public keys $y_{1i} = g^{a_{1i}}$, one for each possible choice the voter can make. The idea is that we now only will need one x, and that encryption will be as follows:

$$(x, w_1, \dots, w_{k_{max}}) = (g^t, y_{11}^t f(v_1), \dots, y_{1k_{max}}^t f(v_{k_{max}})),\ t \in \mathbb{Z}_{|G|}$$

Then we get decryption as follows:

$$f(v_i) = w_i x^{-a_{1i}},\ i \in \{1, \dots, k_{max}\}$$

So we still have one decryption key for every choice. But in the original protocol we have the following relationship between the private keys: $a_1 + a_2 \equiv a_3 \pmod{|G|}$. Clearly this is no longer a possibility. Instead we get three possible ways in which to get a similar relationship between the private keys. The three ways are as follows:

$$\begin{aligned} a_{1i} + a_{2i} &\equiv a_3 \pmod{|G|} \\ a_{1i} + a_2 &\equiv a_{3i} \pmod{|G|} \\ a_{1i} + a_{2i} &\equiv a_{3i} \pmod{|G|} \\ i &\in \{1, \dots, k_{max}\} \end{aligned}$$

Now we argue why two of these choices are not secure enough. Let us look at $a_{1i} + a_{2i} \equiv a_3 \pmod{|G|}$, $i \in \{1, \dots, k_{max}\}$ first. Assume that an attacker A has corrupted the ballot box, and that A can guess one choice the voter has done. So A knows $f(v_j)$ for one j. The ballot box already knows $a_{2i}\, \forall i \in \{1, \dots, k_{max}\}$, so the attacker can find x^{a_3} by the following calculations:

$$\begin{aligned} x^{a_{1j}} &= g^{a_{1j}t} = y_{1j}^t = w_j(f(v_j))^{-1} \\ x^{a_3} &= x^{a_{1j}} x^{a_{2j}} \end{aligned}$$

Now we can find $x^{a_{1i}}\, \forall i$ by doing the following calculation for all i:

$$x^{a_{1i}} = x^{a_3} x^{-a_{2i}}$$

Hence by taking the inverse of each of these $x^{a_{1i}}$'s we obtain all the information needed to decrypt the ciphertexts.

Let us now look at $a_{1i} + a_2 \equiv a_{3i} \pmod{|G|}$, $i \in \{1, \dots, k_{max}\}$. Assume that an attacker A has corrupted receipt generator, and that A can guess $f(v_j)$ for one j. The receipt generator has $\{a_{3i}\}$, so the attacker can find x^{a_2} by the following calculations:

$$\begin{aligned} x^{a_{1j}} &= g^{a_{1j}t} = y_{1j}^t = w_j(f(v_j))^{-1} \\ x^{a_2} &= x^{a_{3j}} x^{-a_{1j}} \end{aligned}$$

Now we find $\{x^{a_{1i}}\}$ by calculating

$$x^{a_{1i}} = x^{a_{3i}} x^{-a_2}\, \forall i \in \{1, \dots, k_{max}\}$$

By taking the inverse of each $x^{a_{1i}}$ we obtain all the information needed to decrypt the ciphertexts.
Remark: It is not a unlikely event that an attacker can guess one choice the voter has made; for example many voters will not use all their choices, and hence $v_{k_{max}} = 0$ for these voters.
Remark 2: In both cases above it is likely that if a guess is wrong, the ballot box or the receipt generator respectively will notice, since the ballot obtained by decrypting with the found information would likely yield a spoilt ballot.

So we have showed that using $a_{1i} + a_2 \equiv a_{3i}$ or $a_{1i} + a_{2i} \equiv a_3$ will not suffice. Hence we propose to use $a_{1i} + a_{2i} \equiv a_{3i}$, $i \in \{1, \dots, k_{max}\}$ as our change in the protocol. Clearly neither of the above attacks can be used in this case, since the relationship $a_{1i} + a_{2i} \equiv a_{3i}$ is independent of the relationship $a_{1j} + a_{2j} \equiv a_{3j}$ for $j \neq i$, $j \in \{1, \dots, k_{max}\}$. Now we must prove that using the relationship $a_{1i} + a_{2i} \equiv a_{3i}$ will give us a secure protocol.

3.3 The New Simplified Protocol

In this section we implement the changes made to the simplified protocol. I want to mention that we only comment on the parts where there has been made changes. The parts of the protocol we haven't changed will not be mentioned here.

Key generation:

Instead of generating three secret parameters, we must now generate $3k_{max}$ secret parameters. We must generate k_{max} sets of triples a_{1i}, a_{2i} and a_{3i}, $i \in \{1, \ldots, k_{max}\}$ such that

$$a_{1i} + a_{2i} \equiv a_{3i} \pmod{|G|}$$

This also gives $3k_{max}$ public parameters,

$$y_{1i} = g^{a_{1i}}, y_{2i} = g^{a_{2i}}, \text{ and } y_{3i} = g^{a_{3i}}, i \in \{1, \ldots k_{max}\}$$

The ballot box gets the set of a_{2i}'s, the receipt generator gets the set of a_{3i}'s and the decryption service gets the set of a_{1i}'s, $i \in \{1, \ldots, k_{max}\}$. Note that $r(v)$ is not changed because of this, it is just as before.

Vote submission:

Given the vote $\{v_1, \ldots, v_k\}$ we encrypt as follows:

1. The voter sends $(v_1, \ldots, v_k)$ to his computer P. The computer sets $v_i = 0$ for $i = k+1, \ldots, k_{max}$.

2. Sample one t from $\mathbb{Z}_{|G|}$, compute $(x, w_1, \ldots, w_{k_{max}})$ $= (g^t, y_{11}^t f(v_1), \ldots, y_{1k_{max}}^t f(v_{k_{max}})$.

3. The ballot box computes $\check{x} = x^s$ and $\check{w}_i = w_i^s \check{x}^{a_{2i}}$, $i \in \{1, \ldots, k_{max}\}$. The ciphertexts $(\check{x}, \check{w}_1, \ldots, \check{w}_{k_{max}})$ and the voter's name is sent to R.

4. The receipt generator computes $\check{r}_i = d(\check{w}_i \check{x}^{-a_{3i}})$ and sends $(\check{r}_i, \ldots, \check{r}_k)$ to the voter. (Note that the fact that k can be deduced from the number of non-identity decryptions still holds.)

5. The voter verifies that every pair $(v_i, \check{r}_i)$ is in the set of receipt codes received before the election, and if so considers the ballot cast.

Counting:

When we count, we close the ballot box, choose the last submitted ballot from each voter and send the set of these in a random order to the decryption service. Then the decryption service decrypt by $\mu_i = w_i x^{-a_{1i}}$ and outputs the decrypted ballots obtained by this decryption in a random order.

Completeness:

The protocol is complete as long as when every player is honest, the decryption service decrypts the ballots correctly and the receipt generator generates the correct receipt codes. The only non-obvious thing about completeness is the receipt codes received by the voter being correctly generated. The argument is as follows for r_i:

$$\check{w}_i \check{x}^{-a_{3i}} = w_i^s \check{x}^{a_{2i}} \check{x}^{-a_{3i}} = w_i^s \check{x}^{-a_{1i}} = w_i^s (x^s)^{-a_{1i}} = (w_i x^{-a_{1i}})^s = (f(v_i))^s$$

3.4 Analysis of the New Simplified Protocol

The only concerned part of the security analysis from the original protocol is the ballot box in part c) of section 3.2 in [5]. We have to make a new argument for the ballot box not being able to extract any information from the ballots.

Say an attacker A has an advantage on the Decision Diffie-Hellman problem, say $\text{Adv}_A^{\text{DDH}} = \epsilon_{\text{DDH}}$. We have come up with two different analyses giving two different bounds, both analyses are similar and we give a sketch for the first analysis giving the rougher bound, while giving the second and best analysis in detail. In the two analyses we have made, we make a set of games taking our information (the ballots) from being encrypted as in real life to just be encryptions of random elements from G. We name real life for the 0'th game (G_0), and number them increasingly.

In the first analysis we made, we started out with all ballots encrypted as in real life. Then in turn we randomized one voters ballot for each game, keeping the ballots that was randomized in the earlier games still randomized. So in Game i, i ballots were randomized. Then we made algorithms B_i that took as input a tuple $(x_0, \ldots, x_{k_{max}})$ that was either a tuple from $G^{k_{max}+1}$ drawn at random or a tuple from $H_0^{k_{max}+1}$ drawn at random. The algorithms were made so that if the tuple was drawn from $H_0^{k_{max}+1}$, then $B_i \simeq G_i$ (G_i is game number i), and if the tuple was drawn from $G^{k_{max}+1}$, $B_i \simeq G_{i+1}$. Hence if we could distinguish between the Game G_i and the Game G_{i+1}, then we would have a distinguisher for the subgroup membership problem $G^{k_{max}+1} \overset{?}{\leftrightarrow} H_0^{k_{max}+1}$.

Then we basically summed up the advantage an attacker could get on distinguishing each game from the previous one , to find the advantage one could get on distinguishing the first and the last game. Remember that the first game was encryptions of real ballots and the last game was that all ciphertexts were encryptions of random group elements. Taking the sum we got that the advantage an attacker A could get on distinguishing between all ciphertexts being real encryptions and all ciphertexts being random encryptions was $\mathrm{Adv}_{\mathrm{A}}^{dist} = (N+1) \cdot k_{max} \cdot \epsilon_{\mathrm{DDH}}$, where N, is the number of votes (i.e $N \approx 3.8 \times 10^6$ [6], in an analysis we must assume every voter votes).

What does this bound really give us? We know $N + 1 < 2^{22}$ and if we say $k_{max} = 60 \approx 2^6$, then the advantage is $\mathrm{Adv}_{\mathrm{A}}^{dist} = (N + 1) \cdot k_{max} \cdot \epsilon_{\mathrm{DDH}} < 2^{22} \cdot 2^6 \cdot \epsilon_{\mathrm{DDH}} = 2^{28} \cdot \epsilon_{\mathrm{DDH}}$. It is reasonable to say that 2^{-60} is a small advantage and 2^{-10} is a large advantage, so we therefore assume an attacker can get an advantage of 2^{-60} on the Decision Diffie-Hellman problem. Then the advantage becomes $\mathrm{Adv}_{\mathrm{A}}^{dist} = 2^{28} \cdot \epsilon_{\mathrm{DDH}} = 2^{-32}$. This is neither close to 2^{-60} nor 2^{-10}. Therefore we wanted to try to make a better bound, and came up with a analysis giving a better bound.

Now we show how to do the analysis giving the best bound. We make a set of games reducing the problem of finding any information from the ballots when encrypted as in real life, to the problem of finding any information from the ballots when given encryptions of elements drawn at random from the group (in this situation a guess will do as good as anything else). We further show that being able to distinguish a game from the previous one is equal to obtaining an advantage on the Decision Diffie-Hellman problem. We name the games with numbers, i.e. Game 0 (real life) is named G_0 and so on. The games goes as follows:

G_i: In this game we do as follows for each ballot:

1. Set $(u_1, \ldots, u_i)$ to be elements drawn at random from G
2. Set $(u_{i+1}, \ldots u_{k_{max}}) = (f(v_{i+1}), \ldots f(v_{k_{max}}))$
3. Encrypt by: $(x, w_1, \ldots, w_{k_{max}}) = (g^t, y_{11}^t u_1, \ldots, y_{1k_{max}}^t u_{k_{max}})$

Remark: In each game the attacker, A, is supposed to guess 0 or 1, 0 if he thinks the encrypted ballots he gets are encrypted as in real life, and 1 if he thinks the encrypted ballots he gets is encryptions of random elements from G.

Define $H_i^{k_{max}+1} = \langle (g, y_{11}, \ldots, y_{1k_{max}}), (g, r_1, y_{12}, \ldots, y_{1k_{max}}), \ldots,$ $(g, r_1, \ldots, r_i, y_{1(i+1)}, \ldots, y_{1k_{max}}) \rangle$ for $i \in \{0, \ldots, k_{max}\}$. Now we make an algorithm B_i that takes in a tuple $\mathbf{x} = (x_0, \ldots, x_{k_{max}})$ such that $\mathbf{x} \in H_i^{k_{max}+1}$ or $\mathbf{x} \in H_{i+1}^{k_{max}+1}$. The algorithm uses this tuple such that if $\mathbf{x} \in H_i^{k_{max}+1}$ we get $B_i \simeq G_i$ and if $\mathbf{x} \in H_{i+1}^{k_{max}+1}$ we get $B_i \simeq G_{i+1}$. We now describe B_i:

Algorithm B_i:

1. Input: $(x_0, \dots, x_{k_{max}})$
2. For each voter generate a t' and a t and do the following
3. $$\begin{aligned}(x, w_1, \dots, w_{k_{max}}) =& (g^t x_0^{t'}, x_1^{t'}, \dots, x_i^{t'}, y_{1(i+1)}^t x_{i+1}^{t'} f(v_{i+1}),\\ & \dots, y_{1k_{max}}^t x_{k_{max}}^{t'} f(v_{k_{max}}))\end{aligned}$$
4. Send all the $(x, w_1, \cdots, w_{k_{max}})$ tuples, one for each voter, to A and wait for an answer
5. B_i outputs the number A answers with (0 or 1)

Lemma 3.1. $|\Pr[B_i(\mathbf{x}) = 1 \mid \mathbf{x} \in H_{i+1}^{k_{max}+1}] - \Pr[B_i(\mathbf{x}) = 1 \mid \mathbf{x} \in H_i^{k_{max}+1}]| = \epsilon_{\text{DDH}}$.

Proof. If we keep on using the notation of Y_i we used in Chapter 2, i.e we write $x \overset{r}{\leftarrow} Y_i$ when drawing from $G^{k_{max}+1}$ with equal probability of drawing any element from $H_i^{k_{max}+1}$ and zero probability of drawing any other element, we get the following

$$\begin{aligned}&|\Pr[B_i(\mathbf{x}) = 1 \mid \mathbf{x} \in H_{i+1}^{k_{max}+1}] - \Pr[B_i(\mathbf{x}) = 1 \mid \mathbf{x} \in H_i^{k_{max}+1}]|\\ &\quad = \text{Adv}_A^{Y_{i+1}/Y_i} = \epsilon_{\text{DDH}}\end{aligned}$$

Where the last equality follows from Lemma 2.7. □

Lemma 3.2. *If* $\mathbf{x} \in H_i^{k_{max}+1}$ *then* $B_i \simeq G_i$ *and if* $\mathbf{x} \in H_{i+1}^{k_{max}+1}$ *then* $B_i \simeq G_{i+1}$.

Proof. Assume $\mathbf{x} = (g^{t''}, x_1, \dots, x_i, y_{1(i+1)}^{t''}, \dots, y_{1k_{max}}^{t''}) \in H_i^{k_{max}+1}$. The tuples A gets as input from B_i are then on the following form:

$$\begin{aligned}(x, w_1, \dots, w_{k_{max}}) =& (g^t x_0^{t'}, x_1^{t'}, \dots, x_i^{t'}, y_{1(i+1)}^t x_{i+1}^{t'} f(v_{i+1}),\\ & \dots, y_{1k_{max}}^t x_{k_{max}}^{t'} f(v_{k_{max}}))\\ =& (g^{t+t't''}, x_1^{t'}, \dots, x_i^{t'}, y_{1(i+1)}^{t+t't''} f(v_{i+1}),\\ & \dots, y_{k_{max}}^{t+t't''} f(v_{k_{max}}))\end{aligned}$$

Now the only difference between the tuples sent to A by B_i and the tuples sent to A from game G_i is that in game G_i we multiply the random u_i's with y_{1i}^t while when using B_i we take x_i to the power t' for every i. We have that u_i and x_i are random elements for $i \in \{1, \dots, i\}$, and multiplying a random element with another element is still a random element. An exponent of a random element is still a random element, so in both cases we end up with random elements. Hence we get $B_i \simeq G_i$

Assume now that $\mathbf{x} = (g^{t''}, x_1, \ldots, x_{i+1}, y_{1(i+2)}^{t''}, \ldots, y_{1k_{max}}^{t''}) \in H_{i+1}^{k_{max}+1}$. Now the tuples A gets as input from B_i are on the following form:

$$\begin{aligned}(x, w_1, \ldots, w_{k_{max}}) =&(g^t x_0^{t''}, x_1^{t'}, \ldots, x_{i+1}^{t'}, y_{1(i+2)}^t x_{i+2}^{t'} f(v_{i+2}),\\ &\ldots, y_{1k_{max}}^t x_{k_{max}}^{t'} f(v_{k_{max}}))\\ =&(g^{t+t't''}, x_1^{t'}, \ldots, x_{i+1}^{t'}, y_{1(i+2)}^{t+t't''} f(v_{i+2}),\\ &\ldots, y_{k_{max}}^{t+t't''} f(v_{k_{max}}))\end{aligned}$$

The tuples sent to A by B_i and the tuples sent to A from Game G_{i+1} now only differ in that we multiply the different u_i's with y_{1i}^t in G_{i+1} and exponentiate the x_i's by t' in B_i. With the same argument as for the first case we conclude $B_i \simeq G_{i+1}$. □

Definition 3.1. Define E_i to be the event that A guesses 1 in Game G_i.

We have the following:

$$\begin{aligned}\Pr[\text{“}A \text{ wins“} \mid \text{real encryption}] &= \Pr[\text{“}A \text{ guesses } 0\text{“} \mid A \text{ plays in } G_0]\\ &= 1 - \Pr[E_0]\\ \Pr[\text{“}A \text{ wins“} \mid \text{random encryption}] &= \Pr[E_{k_{max}}]\end{aligned}$$

By the definition of advantage in Chapter 2 we get the following result:

$$\begin{aligned}\left|\Pr[\text{“}A \text{ wins“}] - \frac{1}{2}\right| &= \left|\frac{1}{2}\Pr[\text{“}A \text{ wins“} \mid \text{real encryption}]\right.\\ &\qquad \left. + \frac{1}{2}\Pr[\text{“}A \text{ wins“} \mid \text{random encryption}] - \frac{1}{2}\right|\\ &= \left|\frac{1}{2} - \frac{1}{2}\Pr[E_0] + \frac{1}{2}\Pr[E_{k_{max}}] - \frac{1}{2}\right|\\ &= \frac{1}{2}|\Pr[E_{k_{max}}] - \Pr[E_0]|\\ &= \frac{1}{2}|\Pr[E_{k_{max}}] - \Pr[E_{k_{max}-1}] + \Pr[E_{k_{max}-1}]\\ &\qquad - \ldots + \Pr[E_1] - \Pr[E_0]|\\ &\le \sum_{i=1}^{k_{max}} |\Pr[E_i] - \Pr[E_{i-1}]|\\ &= \sum_{i=1}^{k_{max}} (|\Pr[B_{i-1}(\mathbf{x}) = 1 \mid \mathbf{x} \in H_i^{k_{max}+1}]\end{aligned}$$

$$\begin{aligned} &- \Pr[B_{i-1}(\mathbf{x}) = 1 \mid \mathbf{x} \in H_{i-1}^{k_{max}+1}]|) \\ = {}& k_{max} \cdot \epsilon_{\mathrm{DDH}} \end{aligned}$$

Hence the advantage an attacker can get on distinguishing between the ciphertexts being encryptions of real ballots and encryptions of random group elements is $\mathrm{Adv}_{\mathrm{A}}^{dist} = k_{max} \cdot \epsilon_{\mathrm{DDH}}$. This is clearly a much better bound on the advantage of A. If we as before assume $k_{max} = l \approx 2^6$ and $\epsilon_{\mathrm{DDH}} = 2^{-60}$ we get $\mathrm{Adv}_{A}^{dist} = k_{max} \cdot \epsilon_{\mathrm{DDH}} = 2^6 \cdot 2^{-60} = 2^{-54}$. Now this number is fairly close to $\epsilon_{\mathrm{DDH}} = 2^{-60}$, so we don't get much of an advantage. Even though we have stated what we say is a small and large advantage without any reference, we still see that the advantage is dependent on the advantage on the DDH-problem. So a distinguisher between encryptions of real ballots and encryption of random group elements must give us an adversary for the DDH-problem.

CHAPTER 4

FULL PROTOCOL

In this chapter we describe the changes made to what Gjøsteen has called the full protocol, then we analyse the full protocol with these changes. The original full protocol is found in section 4 of [5].

4.1 Changes in the Full Protocol

In this section we comment on the changes made in the full protocol. We only comment on places where changes have been made.

Key generation

It is the electoral board that generates the secret election keys $\{a_{1i}\}$, $\{a_{2i}\}$ and $\{a_{3i}\}$, it also generates the per-voter keys. As in the original protocol we model this by a simple ideal functionality.

The original key generation functionality is found in Figure 7, p.18-19 in [5]. The key generation functionality with the changes made, is as follows:

```
Once (start) has been received from every electoral board player:
1: Choose the function $f : \mathcal{O} \to G$.
2: Choose random $a_{1i}$ and $a_{2i}$, and compute $a_{3i} = (a_{2i} + a_{1i}) \pmod{|G|}$ for $i \in \{1, \ldots, k_{max}\}$. Compute $y_{1i} = g^{a_{1i}}$, $y_{2i} = g^{a_{2i}}$ and $y_{3i} = g^{a_{3i}}$ for $i \in \{1, \ldots, k_{max}\}$.
```

```
3: For every voter $V$, choose a random exponent $s$ and a PRF instance $d$. Compute the per-voter commitment $\gamma = g^s$ and the set $\mathcal{RC} = \{(v, d(f(v)^s)) \mid v \in \mathcal{O}\}$. Send $(\mathsf{codes}, \mathcal{RC})$ to $V$.
4: Send $(\mathsf{keys}, \{y_{1i}\}, \{a_{2i}\}, \{y_{3i}\}, (V, s))$ to $B$, $(\mathsf{keys}, \{y_{1i}\}, \{y_{2i}\}, \{a_{3i}\}, (V, \gamma, d))$ to $R$, and $(\mathsf{keys}, \{y_{1i}\})$ to $A$.
5: For every computer P, send $(\mathsf{keys}, \{y_{1i}\}, f)$ to $P$.
Once $(\mathsf{count})$ has been received by a qualified majority of the electoral board players:
1: Send $(\mathsf{keys}, \sum_{i=1}^{k_{max}} a_{1i})$ to $D$.
```

Program 1: Updated ideal functionality for key generation

The Voter's Computer

The computer gets the voters ballot from the voter, encrypts the ballot, proves it knows the contents of the ciphertexts generated by encrypting, signs the encrypted ballot and the proof of knowledge and sends this to the ballot box via $\mathcal{F}_{eid}$. Then the computer waits for a receipt from B containing the receipt generator's signature that it has seen the ballot the computer sent to the ballot box. When this receipt is received the computer checks it and, if it is valid, outputs to the voter that the ballot was accepted.

The original program for the voter's computer is found in Figure 9, p. 20-21 in [5]. The voter's computer's program with changes made, is as follows:

```
On $(\mathsf{keys}, \{y_{1i}\}, f)$ from $\mathcal{F}_{key}$:
1: Store $\{y_{1i}\}$ and $f$.
On $(\mathsf{vote}, v_1, \ldots, v_k)$ from $V$:
1: Send $(\mathsf{establish}, V, B)$ to $\mathcal{F}_{eid}$ and wait for $(\mathsf{established}, sid)$.
2: Set $v_{k+1} = \cdots = v_{k_{max}} = 0$.
3: Choose random $t$ and compute $x = g^t$.
4: For $i$ from 1 to $k_{max}$: compute $w_i = y_{1i}^t f(v_i)$.
5: Send $(\mathsf{prove}, V, 1, g, x, t)$ to $\mathcal{F}_{pok}$ and wait for $(\mathsf{proof}, \ldots, \pi)$ from $\mathcal{F}_{pok}$.
6: Send $(\mathsf{sign}, V((x, w_1, \ldots, w_{k_{max}}), \pi))$ to $\mathcal{F}_{eid}$, and wait for $(\mathsf{signature}, \ldots, \sigma_V)$ from $\mathcal{F}_{eid}$.
7: Send $(\mathsf{send}, sid, (\mathsf{vote}, V((x, w_1, \ldots, w_{k_{max}}), \pi), \sigma_V))$ to $\mathcal{F}_{eid}$, and wait for $(\mathsf{recv}, sid, (\mathsf{receipt}, \sigma_R))$ from $\mathcal{F}_{eid}$.
8: Compute $h_b \leftarrow Hash(V, (x, w_1, \ldots, w_{k_{max}}), \pi)$.
9: Send $(\mathsf{verify}, R, h_b, \sigma_R)$ to $\mathcal{F}_{eid}$ and wait for $(\mathsf{verified}, R, h_b, \sigma_R)$.
10: Send $(\mathsf{accepted})$ to $V$.
```

Program 2: Updated program for the voter's computer

The Ballot Box

The ballot box does not do any work until it has received its codes. Then as computers wants to submit ballots, the ballot box connects to the computers, receives the encrypted and signed ballot, verifies the signature and proof of knowledge, does the appropriate computations, proves that it has computed correctly and sends the result to the receipt generator. Then it awaits the receipt generator's reply containing a signature of the hash of the ballot, checks the signature and sends the signature to the computer.

When the election closes, the ballot box is told to close, it then waits for ongoing submissions to complete, selects the ballots to be counted and sends them to the decryption service in a random order. Then it's final work is to send it's contents to the auditor.

The original program for the ballot box is found in Figure 10, p. 21-23 in [5]. The program for the ballot box with changes made, is as follows:

```
Do nothing until $(\mathsf{keys}, \{y_{1i}\}, \{a_{2i}\}, \{y_{3i}\}, \{(V, s)\})$ has been received
    from $\mathcal{F}_{key}$, then do:
1: Record $\{y_{1i}\}$, $\{a_{2i}\}$, $\{y_{3i}\}$ and the pairs $(V, s)$.
On $(\mathsf{established}, sid, V, P)$ from $\mathcal{F}_{eid}$:
1: Wait for $(\mathsf{recv}, sid, P, (((x, w_1, \dots, w_{k_{max}}), \pi), \sigma_V))$ from $\mathcal{F}_{eid}$.
2: Send $(\mathsf{verify}, V, ((x, w_1, \dots, w_{k_{max}}), \pi), \sigma_V)$ to $\mathcal{F}_{eid}$ and wait for
   $(\mathsf{verified}, \dots, \sigma_V)$ from $\mathsf{F}_{eid}$.
3: Send $(\mathsf{verify}, V, 1, g, x, \pi)$ to $\mathcal{F}_{pok}$ and wait for $(\mathsf{verified}, \dots, \pi)$ from $\mathcal{F}_{pok}$.
4: Look up the stored pair $(V, s)$ and place an exclusive lock on the pair (waiting
   for any other session to release it's exclusive lock).
5: Select the next sequence number $seq$.
6: Compute $\check{x}$ as specified on the next page.
7: for $i = 1$ to $k_{max}$ :
8:         Compute $(\check{w}_i, \check{\pi}_i)$ as specified on the next page.
9: Send $(\mathsf{ballot}, seq, V, x, \check{x}, ((w_1, \check{w}_1, \check{\pi}_1), \dots, (w_{k_{max}}, \check{w}_{k_{max}}, \check{\pi}_{k_{max}})), \pi, \sigma_V)$ to $R$,
   and wait for $(\mathsf{receipt}, seq, \sigma_R)$ from $R$.
10: Compute $h_b \leftarrow Hash(V, (x, w_1, \dots, w_{k_{max}}), \pi)$.
11: Send $(\mathsf{verify}, R, h_b, \sigma_R)$ to $\mathcal{F}_{eid}$ and wait for $(\mathsf{verified}, R, h_b, \sigma_R)$ from $\mathcal{F}_{eid}$.
12: Store $(seq, V, (x, w_1, \dots, w_{k_{max}}), \pi, \sigma_V)$ and release the lock on the record
    $(V, s)$.
13: Send $(\mathsf{receipt}, \sigma_R)$ to $P$.
On $(\mathsf{count})$ from $D$:
1: Stop processing $(\mathsf{established}, \dots)$ messages from $\mathcal{F}_{eid}$.
2: Stop any voting sessions that have not yet reached Step 4 and wait for re-
   maining sessions to terminate.
3: Send $(\mathsf{count})$ to $R$.
```

```
4: Let $S_{bb}$ be the list of all recorded entries $(seq, V, (x, w_1, \dots, w_{k_{max}}), \pi, \sigma_V)$.
5: For each voter $V$, find the recorded entry with the largest
   sequence number $seq$ and extract the ballot $(x, w_1, \dots, w_{k_{max}})$.
   Compute
       $w = \prod_{i=1}^{k_{max}} w_i$
   and add $(x, w)$ to the list $L$.
6: Sort $L$. Send $(\mathsf{decrypt}, L)$ to the decryption service $D$, and $(\mathsf{content}, S_{bb})$ to
   $A$.
```

Program 3: Updated program for the ballot box

Remark: To obtain the product $\prod_{i=1}^{k_{max}} f(v_i)$ we now multiply as follows:

$$\prod_{i=1}^{k_{max}} f(v_i) = w \cdot x^{-\sum_{i=1}^{k_{max}} a_{1i}}$$

We compute $\check{x}$ and the $\check{w}_i$'s as in section 3.3. The correctness proofs $\check{\pi}_i$ are made as follows:

Input: x, $\{w_i\}$, s and a_{2i}.

1. Compute $\bar{x} = x^s$.
2. For i from 1 to k_{max}: compute $\bar{w}_i = w_i^s$ and $\hat{w}_i = \bar{x}^{a_{2i}}$.
3. Send $(\mathsf{prove}, V, 1, (g, x, w_1, \dots, w_{k_{max}}), (g^s, \bar{x}, \bar{w}_1, \dots, \bar{w}_{k_{max}}), s)$ to $\mathcal{F}_{pok}$ and wait for $(\mathsf{proof}, \dots, \bar{\pi})$ from $\mathcal{F}_{pok}$.
4. For i from 1 to k_{max}: send $(\mathsf{prove}, V, 1, (g, \bar{x}), (y_{2i}, \hat{w}_i), a_{2i})$ to $\mathcal{F}_{pok}$ and wait for $(\mathsf{proof}, \dots, \hat{\pi}_i)$ from $\mathcal{F}_{pok}$.
5. Now the result is $(\check{x}, \check{w}_i) = (\bar{x}, \bar{w}_i\hat{w}_i)$, the proof is $\check{\pi}_i = (\bar{w}_i, \hat{w}_i, \bar{\pi}, \hat{\pi}_i)$.

To verify the proofs $\check{\pi}_i$ on input $(x, w_1, \dots, w_{k_{max}})$, $(\check{x}, \check{w}_1, \dots, \check{w}_{k_{max}})$, $\{y_{2i}\}$, γ do as follows:

1. For i from 1 to k_{max}: check that $\check{w}_i = \bar{w}_i\hat{w}_i$.
2. Send $(\mathsf{verify}, V, 1, (g, x, w_1, \dots, w_{k_{max}}), (\gamma, \bar{x}, \bar{w}_1, \dots, \bar{w}_{k_{max}}), \bar{\pi})$ to $\mathcal{F}_{pok}$ and wait for $(\mathsf{verified}, \dots)$ from $\mathcal{F}_{pok}$.
3. For i from 1 to k_{max}: send $(\mathsf{verify}, V, 1, (g, x), (y_{2i}, \hat{w}_i), \hat{\pi}_i)$ to $\mathcal{F}_{pok}$ and wait for $(\mathsf{verified}, \dots)$ from $\mathcal{F}_{pok}$.

Receipt Generator

The receipt generator receives from the ballot box the encrypted signed ballot with the voter's computer's proof of knowledge, together with a sequence number. It also receives the ciphertexts $(\breve{x}, \breve{w}_1, \ldots, \breve{w}_{k_{max}})$ made by the ballot box and the proof made by the ballot box for the ballot box computing correctly. It verifies the signature, all the proofs and the sequence number. Then the receipt generator decrypts $(\breve{x}, \breve{w}_1, \ldots, \breve{w}_{k_{max}})$ to obtain the receipt codes, and sends the obtained receipt codes to the voter. At last the receipt generator stores the voters name, the sequence number and a hash of the encrypted ballots. The signed hash of the encrypted ballots is also sent as a receipt to the ballot box.

The program for the original receipt generator is found in Figure 11, p. 23-24 in [5]. The new program for the receipt generator with changes made, is as follows:

```
On $(\mathsf{keys}, \{y_{1i}\}, \{y_{2i}\}, \{a_{3i}\}, \{(V, \gamma, d)\})$ from $\mathcal{F}_{key}$:
 1: Record $\{y_{1i}\}$, $\{y_{2i}\}$, $\{a_{3i}\}$ and the triples $(V, \gamma, d)$.
On $(\mathsf{ballot}, seq, V, x, \breve{x}, ((w_1, \breve{w}_1, \breve{\pi}_1) \ldots, (w_{k_{max}}, \breve{w}_{k_{max}}, \breve{\pi}_{k_{max}})), \pi, \sigma_V)$ from $B$:
 1: Compute $h_b \leftarrow Hash(V, (x, w_1, \ldots, w_{k_{max}}), \pi, \sigma_V)$ and $h'_b \leftarrow Hash(V,$
    $(x, w_1, \ldots, w_{k_{max}}), \pi)$.
 2: Look up the recorded tuple $(V, \gamma, d)$ and place an exclusive lock on the tuple
    (waiting for any other session to release it's exclusive lock).
 3: Verify that no record $(\cdot, \cdot, \cdot, h'_b)$ or $(V, seq', \cdot, \cdot)$ with $seq' \geq seq$ exists.
 4: Send $(\mathsf{verify}, V, ((x, w_1, \ldots, w_{k_{max}}), \pi), \sigma_V)$ to $\mathcal{F}_{eid}$ and wait for $(\mathsf{verified}$
    $, \ldots, \sigma_V)$.
 5: Send $(\mathsf{verify}, V, 1, g, x, \pi)$ to $\mathcal{F}_{pok}$ and wait for $(\mathsf{verified}, \ldots)$.
 6: for $i = 1$ to $k_{max}$ :
 7:         Verify the computation of $(\breve{x}, \breve{w}_i)$ using the proof $\breve{\pi}_i$ a described
            under the ballot box's changed program.
 8:         Compute $r_i = \breve{w}_i \breve{x}^{-a_{3i}}$. If $r_i \neq 1$, then $k = i$.
 9:         Compute $\breve{r}_i = d(r_i)$.
10: Send $(\mathsf{sign}, R, h_b,)$ to $\mathcal{F}_{eid}$ and wait for $(\mathsf{signature}, R, h_b, \sigma_R)$.
11: Record $(V, seq, h_b, h'_b)$. Send $(\mathsf{receipt}, seq, \sigma_R)$ to $B$.
12: Send $(\mathsf{receipt}, \breve{r}, \ldots, \breve{r}_k)$ to $V$.
On $(\mathsf{count})$ from $B$:
 1: Verify that all sessions have terminated.
 2: Send $(\mathsf{flush})$ to $\mathsf{F}_{sc}$.
 3: Let $S_R$ be the list of all recorded entries $(V, seq, h_b, h'_b)$. Send $(\mathsf{hashes}, S_R)$ to
    $A$.
```

Program 4: Updated program for the receipt generator

Decryption service

When the election is over, the decryption service receives the ballots to count from the ballot box in a random order. It then decrypts the ballots and shuffles them before output. Before the decryption service is allowed to decrypt, the auditor must accept the input as correct, this is done by the fact that the decryption service takes the hash of all ballots and sends the result to the auditor, then the auditor takes the hash of the ballots it has found should be counted based on what it has received from the ballot box and the receipt generator. If the two hashes are the same, the auditor tells the decryption service to proceed.

When finished, the decryption service must prove to the auditor that the encrypted ballots contain a permutation of the decrypted ballots it outputs. To prove that the encrypted ballots contain a permutation of the output ballots, the decryption service shuffles and rerandomize the encrypted ballots, and then decrypts the result. Correctness of the shuffle and decryption is proved to the auditor.

The decryption service also decodes the decrypted ballots. Since we have not changed anything about the decoding or shuffle proofs, we refer to the original protocol for more information on this.

The original decryption service program is found in Figure 12, p. 25 in [5]. The decryption service program with changes made, is as follows:

```
On $(\mathsf{keys}, \sum_{i=1}^{k_{max}} a_{1i})$ from $\mathcal{F}_{key}$:
1: Send $(\mathsf{count})$ to $B$.
2: Wait for $(\mathsf{decrypt}, (x_1, w_1), \ldots, (x_n, w_n))$ from $B$.
3: Compute $\chi \leftarrow Hash((x_1, w_1), \ldots, (x_n, w_n))$, send $(\mathsf{hash}, \chi)$ to $A$, and wait
   for $(\mathsf{proceed})$ from $A$.
4: Choose a permutation $\Pi$ on $\{1, \ldots, n\}$.
5: Let $y = \prod_{i=1}^{k_{max}} y_{1i}$.
6: for $i = 1$ to $n$ :
7:     Choose a random number $t_i$.
8:     Compute $x'_i = x_{\Pi(i)} g^{t_i} \qquad w'_i = w_{\Pi(i)} (\prod_{j=1}^{k_{max}} y_{1j})^{t_i}$.
9:     Compute $\mu_i = w'_i (x'_i)^{-\sum_{i=1}^{k_{max}} a_{1i}}$, send $(\mathsf{prove}, -, 1, (g, x'_i), (y, w'_i \mu^{-1}),$
       $\sum_{i=1}^{k_{max}} a_{1i})$ to $\mathcal{F}_{pok}$ and wait for $(\mathsf{proof}, \ldots, \pi_i)$.
10: Create a proof $\pi'$ that $(x'_1, w'_1), \ldots, (x'_n, w'_n)$ is a shuffle of $(x_1, w_1), \ldots,$
    $(x_n, w_n)$.
11: Send $(\mathsf{proofs}, (x'_1, w'_1, \mu_1, \pi_1), \ldots, (x'_n, w'_n, \mu_n, \pi_n), \pi')$ to $A$, then output
    $(\mathsf{ballots}, \phi(\mu_1), \ldots, \phi(\mu_n))$.
```

Program 5: Updated program for the decryption service

Auditor

The auditor verifies that the ballot box and the receipt generator has seen the same encrypted ballots. Then it verifies the selection of encrypted ballots sent for decryption, before it verifies the correction of decryption.

```
On $(\mathsf{keys}, \{y_{1i}\})$ from $B$:
 1: Store $\{y_{1i}\}$.
On $(\mathsf{content}, S_{bb})$ from $B$:
 1: Wait for $(\mathsf{hashes}, S_R)$ from $R$.
 2: Verify that every encrypted ballot in $S_{bb}$ has a corresponding hash in $S_R$, and
    vice versa.
 3: Verify the signatures and proofs of knowledge on the encrypted ballots.
 4: Select from $S_{bb}$ the ciphertexts that should be decrypted, sort the list and
    compute a hash $\chi$ of the list.
 5: Wait for $(\mathsf{hash}, \chi')$ from $D$. Verify that $\chi = \chi'$, then send $(\mathsf{proceed})$ to $D$.
 6: Wait for $(\mathsf{proofs}, (x'_1, w'_1, \mu_1, \pi_1), \dots, (x'_n, w'_n, \mu_n, \pi_n), \pi')$ from $D$.
 7: Let $y = \prod_{i=1}^{k_{max}} y_{1i}$.
 8: for $i = 1$ to $n$ :
 9:         Send $(\mathsf{verify}, -, 2, (g, x'_i), (y, w'_i\mu^{-1}), \pi_i)$ to $\mathcal{F}_{pok}$ and wait for
            $(\mathsf{verified}, \dots)$ from $\mathcal{F}_{pok}$.
10: Verify the proof $\pi'$.
11: Output $(\mathsf{accepted}, \phi(\mu_1), \dots, \phi(\mu_n))$.
```

Program 6: Updated program for the auditor

4.2 Security Analysis of the Full Protocol

In this section we analyse the security based on cases for corruption given on page 29 in [5]. The cases for corruption in the order we handle them are as follows:

- A subset of the voters and computers are corrupt, and possibly the ballot box.
- The receipt generator is corrupt.
- The decryption service is corrupt.
- The auditor is corrupt.

We use Kristian Gjøsteen's security analyses found in Chapter 5 in the original protocol as a basis for our security analyses.

Voters, Computers and the Ballot Box

Our starting point is, as it is in the original analysis made by Gjøsteen, the real protocol interacting with a real adversary that has corrupted a number of voters and computers, and possibly the ballot box. We use the games already made as a starting point and evolve them into games suitable for our protocol. We also give arguments for why every game is indistinguishable from the previous one. What we shall prove is the same as in the original protocol, which is (copied from [5]):

- If the ballot box is not corrupt, the auditor will not fail the election.
- For any honest voter that uses only honest computers, any ballot accepted as cast and not superseded should be counted if the auditor accepts the election. The ballot remains confidential regardless.
- If an honest voter uses a corrupt computer (not necessarily for voting), nothing can be guaranteed for voters that submit multiple ballots. However, for voters that submit exactly one ballot and accepts that ballot as cast, with high probability that ballot will be counted unless the voter observes an attack. If the ballot was submitted through an honest computer, the ballot remains confidential.

Game 1 In this game we wish to let a machine M simulate all honest players, this includes all honest voters and computers. Also since M simulate all honest players, M knows all decryption keys, and especially the secret keys $\{a_{1i}\}$, $\{a_{2i}\}$ and $\{a_{3i}\}$. Clearly this game is indistinguishable from the real protocol.

Game 2 The next step is to remove the need for shuffle proofs and proving correctness of decryptions. The reason for this is that we at some point want to start encrypting random elements and therefore will not be able to make a correct proof. Since B cannot see the communication between the decryption service and the auditor, this game is clearly indistinguishable from the previous one.

Game 3 Now we want to be able to assume that $Hash(\cdot)$ is a injective function, so we tell M to abort if it ever observes a collision in $Hash(\cdot)$. As long as $Hash(\cdot)$ is collision resistant, this game is indistinguishable from the previous one. From now on we assume $Hash(\cdot)$ is a injective function in this analysis.

Game 4 We will later need that the per-voter function $d : G \to C$ is a random function between G and C. Therefore we now sample it from all functions from

G to C. Since F is a pseudo-random function family, we conclude that this game is indistinguishable from the previous one.

Game 5 We change this game so that the return code is computed as $d(w_i x^{-a_{1i}})$ instead of using $d(\check{w}_i \check{x}^{-a_{3i}})$, which means M must now generate the receipt codes sent to v beforehand as $d(f(v))$ not $d(f(v)^s)$. The reason for this is to remove any use of $(\check{x}, \check{w}_1, \ldots, \check{w}_{k_{max}})$. Still d is a permutation and exponentiation is a permutation, so the only way these changes can be observable is if $(w_i x^{-a_{1i}})^s \neq (\check{w}_i \check{x}^{-a_{3i}})$. As before, if the ballot box proofs are valid, $(w_i x^{-a_{1i}})^s = (\check{w}_i \check{x}^{-a_{3i}})$ holds except with negligible probability. So we conclude this game is indistinguishable from the previous one. We can observe that we now use none of the receipt code generator's decryption keys $\{a_{3i}\}$.

Game 6 When an honestly generated encrypted ballot is made, M makes it and therefore knows the ballot contained in the encrypted ciphertexts. So when these ballots arrive at the simulated receipt generator we use the remembered ballot to generate the receipt codes instead of decrypting, and when these ballots are sent for final decryption at the decryption service we again use the remembered cleartext ballot instead of decrypting. This game is clearly indistinguishable from the previous one.

Game 7 Now we want to remove the use of a_{1i}. This is because we later on want to make a reduction on the subgroup membership problem $G^{k_{max}+1} \overset{?}{\longleftrightarrow} H_0^{k_{max}+1}$ using $\{y_{1i}\}$ as base together with g to randomize the ballots. If we were still using a_{1i} we would have a distinguisher for this problem since we have the private keys $\{a_{1i}\}$, this we do not want. The way to remove the use of a_{1i} is to use the witness that is given to $\mathcal{F}_{pok}$ to generate π to obtain the receipt codes. So when an adversarially generated encrypted ballot $((x, w_1, \ldots, w_{k_{max}}), \pi)$ reaches the simulated receipt generator, we use the witness p, that satisfies $x = g^p$, provided by the adversary to generate π. The witness p is used to decrypt $f(v_i) = w_i y_{1i}^{-p}$ instead of decrypting $f(v_i) = w_i x^{-a_{1i}}$.

M remembers the cleartext ballot for adversarially encrypted ballots that arrive and is decrypted at the receipt generator, and uses these stored ballots for decryption instead of decrypting ciphertexts when doing the final decryption.

As in the original protocol, by the properties of $\mathcal{F}_{pok}$, the computed decryptions are always correct. So this game is indistinguishable from the previous game.

Claim. The decryption keys $\{a_{1i}\}$ are not used in Game 7.

Proof. The same argument as the one used in [5] to argue that a_1 is not used in Game 7 there, is valid in this case also. □

Game 8 Now we want simulated honest computers to be able to generate ciphertexts as encryptions of ballots without actually knowing the contents or any witness for proving that you know the contents. We want this because in the next game we want to encrypt with random elements when simulating honest computers, and therefore will not be able to produce a witness for our encryptions. So now we no longer give $\mathcal{F}_{pok}$ a witness. $\mathcal{F}_{pok}$ is a trusted third party, and when checking a proof, $\mathcal{F}_{pok}$ just replies with verified or invalid, so $\mathcal{F}_{pok}$ does not actually use the witnesses as long as the encrypted ballot comes from a simulated honest voter since both are simulated by M. So therefore this game is indistinguishable from the previous one.

Game 9 Now we remove the connection between the ciphertexts and the ballots when it is a honest computer who generates the ciphertexts, this makes it easier for us to make the analysis. So instead of encrypting the encoded ballot options we encrypt random group elements. We already use cleartext ballots instead of decrypting in both the receipt generator and decryption service, so we only need to show that B cannot distinguish between encryptions of real ballots (the previous game) and random elements (this game).

Claim. A distinguisher between this and the previous game would give an adversary on the subgroup membership problem $G^{k_{max}+1} \overset{?}{\leftrightarrow} H_0^{k_{max}+1}$, where $H_0^{k_{max}+1} = \langle (g, y_{11}, \ldots, y_{1k_{max}}) \rangle$.

Proof. So given a tuple $\mathbf{x} = (x_0, \ldots, x_{k_{max}})$ we compute as follows:

$$(x, w_1, \ldots, w_{k_{max}}) = (g^t x_0^{t'}, y_{11}^t x_1^{t'} f(v_1), \ldots, y_{1k_{max}}^t x_{k_{max}}^{t'} f(v_{k_{max}}))$$

We now see that if $\mathbf{x} \in H_0^{k_{max}+1}$ the encryption is as in the previous game, and if $\mathbf{x} \in G^{k_{max}+1}$ the encryption is as in this game. Hence, if we have a distinguisher between this and the previous game, we get an adversary on the subgroup membership problem $G^{k_{max}+1} \overset{?}{\leftrightarrow} H_0^{k_{max}+1}$. $\square$

Analysis We are now in the same situation as in Game 9 of [5], and therefore the analysis done there is valid in our case also. Hence we are able to prove that the same conditions (listed earlier) still holds.

Receipt Generator

We start with the real protocol interacting with a real adversary. We wish to prove the following (this is copied from the original protocol, [5]):

- The corrupt receipt generator learns nothing about the submitted ballots, except what the receipt codes tell it.

Game 1 As we did in the previous section when analysing the voter, the voter's computer and the ballot box, we again want to make a machine M that simulate all honest players. Again the machine M has all decryption keys, and especially knows $\{a_{1i}\}$, $\{a_{2i}\}$ and $\{a_{3i}\}$. This game is indistinguishable to the previous one.

Game 2 Now we want to remove the use of the decryption keys $\{a_{1i}\}$. So, now every time M encrypts a ballot, M remembers the cleartext ballot corresponding to each encrypted ballot. So when the encrypted ballos are received by the decryption service, M uses the cleartext ballots instead of decrypting the ciphertexts. Clearly this game is indistinguishable to the previous one. Note that the decryption keys $\{a_{1i}\}$ are not used anymore.

Game 3 There is no need for M to be able to prove that it knows the contents of $(x, w_1, \ldots, w_{k_{max}})$ now. This is because we know M is honest and since $\mathcal{F}_{pok}$ only needs to output verified or invalid to R when R asks $\mathcal{F}_{pok}$ to verify the proof of knowledge. So $\mathcal{F}_{pok}$ now receives random witnesses. Since the witnesses are never used by $\mathcal{F}_{pok}$, this game is indistinguishable from the previous one.

Game 4 We will at a later game want to change the order in which we compute different ciphertexts, hence we now want to generate all ciphertexts at the same time so that $(\check{x}, \check{w}_1, \ldots, \check{w}_{k_{max}})$ can be computed before $(x, w_1, \ldots, w_{k_{max}})$ as long as the relationship between them is kept. So now M creates the messages to the receipt generator when encrypting the ballots, but does not send the messages to R before the corresponding encrypted ballots are received by the ballot box. This game is clearly indistinguishable from the previous one.

Game 5 In this game we change the order in which we compute the ballots. The reason for this is that we want to compute the ciphertexts B is going to compute first, and then relate the other ciphertexts to these. This way we can randomize the ciphertexts used to generate receipt codes in a later game, and still get the other ciphertexts to have the correct relationship. So we change the machine M's computation of the values $x, w_i, \bar{x}, \bar{w}_i, \hat{w}_i, \check{x}, \check{w}_i$ related to the option v_i as follows:

$$\begin{aligned}
\check{x} &= g^t & \check{w}_i &= \check{x}^{a_{3i}} f(v_i)^s \\
& & \hat{w}_i &= \check{x}^{a_{2i}} \\
\bar{x} &= \check{x} & \bar{w}_i &= \check{w}_i \hat{w}_i^{-1} \\
x &= \bar{x}^{s^{-1}} & w_i &= \bar{w}^{s^{-1}}
\end{aligned}$$

Where s^{-1} is the multiplicative inverse of s modulo $|G|$.

A straight-forward computation will show that the change in method of computation does not change the induced probability distributions. Therefore, this game is indistinguishable from the previous game.

Game 6 It is now time to remove the use of the keys $\{a_{2i}\}$. We do this to remove the connection between the x and the w_i's. This way we can in a later game randomize the w_i's so that they contain no information. So now we generate $\hat{w}_i$ as a random element from G instead of computing $\hat{w}_i = \check{x}^{a_{2i}}$. A similar argument as in Game 6 in [5] will give us an adversary against $G^{k_{max}+1} \overset{?}{\leftrightarrow} H_0^{k_{max}}$.

Game 7 At this step we randomize w_i so that it is just a random element from G, this way the only element containing information on the ballots is $\check{w}_i$. So we change the machine M's computation of $x, w_i, \bar{x}, \bar{w}_i, \hat{w}_i, \check{x}, \check{w}_i$ as follows:

$$\begin{aligned} x &= g^t & w_i &= g_i^{t'} \\ \bar{x} &= x^s & \bar{w}_i &= w_i^s \\ \check{x} &= \bar{x} & \check{w}_i &= \check{x}^{a_{3i}} f(v_i)^s \\ & & \hat{w}_i &= \check{w}_i \bar{w}_i^{-1} \end{aligned}$$

Claim. This game is indistinguishable to the previous one.

Proof. To prove this, we look at the elements as exponents of g, and compare the set of exponents. Then we show that we can make a one-to-one correspondence between the set of exponents. At last we explain why this one-to-one correspondence gives that the two computations have the same probability distributions. We only look at this for one i, this could be any i hence it holds for all i. Firstly, since g is a generator, there exists an $r \in \mathbb{Z}_{|G|}$ such that $f(v_i) = g^r$. Now we make a table, the entries in the table will be what g must be exponentiated with to get the indicated element at the top of the column. The table is as follows:

	x	$\check{x}$ $(\bar{x})$	w_i	$\bar{w}_i$	$\hat{w}_i$	$\check{w}_i$
Game 6	ts^{-1}	t	$a_{1i}ts^{-1} + r$	$a_{1i}t + rs$	$a_{2i}t$	$a_{3i}t + rs$
Game 7	t	ts	t'	$t's$	$a_{3i}ts-$ $t's + rs$	$tsa_{3i} + rs$

Note that it is not the same t in Game 6 and 7. By checking one can see that we get a one-to-one correspondence by sending $t \mapsto ts^{-1}$ and $t' \mapsto a_{1i}ts^{-1} + r$, where both maps go from Game 7 to Game 6. Now t and t' in Game 7 are random numbers from $\mathbb{Z}_{|G|}$. In Game 6 t is a random element, hence ts^{-1} and $a_{1i}ts^{-1} + r$ are random. Now if we generate random t and t' in Game 7 and use our one-to-one correspondence to generate the exponents in Game 6, or if we generate the random t in Game 6 and generate it we now see that the probability distributions for the exponents in Game 6 will be the same.

Likewise if we generate the random t in Game 6 and use the one-to-one correspondence to generate t and t' in Game 7, or if we generate random t and t' in

Game 7, we get the same probability distributions for the exponents in Game 7. So since we have a one-to-one correspondence between the exponents such that we can start in any of the games and generate the exponents of both games so that the probability distribution of each game is the same independent of which game you actually start with, we must have that the probability distributions for the exponents in the two games are the same. Hence the two games are indistinguishable. □

Game 8 Now that the only connection between the ciphertext and the cleartext ballot is contained in $\breve{w}_i$ we want to remove this last connection. We remove it by replacing the per voter functions $v \mapsto f(v)^s$ with a random function from $\mathcal{O}$ to G. A similar argument to the one given in [5] gives that this game and the previous game are indistinguishable.

Analysis We end up in the same situation as in [5], and hence conclude that still the receipt generator learns no unavoidable information about the submitted ballots.

Decryption Service

We have not made any changes in the protocol that will need any change of the analysis given in [5] for the decryption service.

Auditor

Gjøsteen proved that the following holds for the auditor in [5]:

- The submitted ballot remains confidential.

We want to prove that the same holds with the new changes.

Game 1 Firstly we let a machine M simulate all the honest players. This machine knows all private decryption keys, especially it knows $\{a_{1i}\}$, $\{a_{2i}\}$ and $\{a_{3i}\}$.

Game 2 Since M controls all players that want to make zero knowledge proofs, $\mathcal{F}_{pok}$ never needs to use the supplied witnesses. So now, instead of giving $\mathcal{F}_{pok}$ the actual witness, we instead give a random witness. Since they are never used, this game is indistinguishable from the previous one.

Game 3 M knows the cleartext ballot of all encrypted ballots, hence we now let M use the cleartext ballot instead of decrypting both in R and D. This means that the contents of the encrypted ballots are never used. These changes are not observable for the auditor, hence this game is indistinguishable from the previous one.

Game 4 Now we want to randomize the encrypted ballots, so that they contain no connection to the cleartext ballots. So now we encrypt random group elements instead of cleartext ballots. A straight-forward reduction to the problem $H_0^{k_{max}+1} \overset{?}{\longleftrightarrow} G^{k_{max}+1}$, with $H_0^{k_{max}+1} = \langle(g, y_{11}, \dots, y_{1k_{max}})\rangle$ will show that this game is indistinguishable to the previous one.

Game 5 In this final game the decryption service uses fresh random encryption in each round instead of rerandomizing when generating the shuffle proof. Again, a straight-forward reduction to the problem $H_0^{k_{max}+1} \overset{?}{\longleftrightarrow} G^{k_{max}+1}$, with $H_0^{k_{max}+1} = \langle(g, y_{11}, \dots, y_{1k_{max}})\rangle$ will show that this game is indistinguishable from the previous game.

Analysis Clearly we are in the same position here as in [5], and hence the analysis done there is valid also in our case.

CHAPTER 5

SUB PROTOCOLS

In this chapter we will describe the changes we want to make to the protocol. With these changes we will need to some times use commitments and Σ-protocols instead of zero-knowledge proofs to prove that different players have done their computations correctly. Our goal is to remove the private keys $\{a_{2i}\}$ from the protocol, and thereby also remove the dependency $a_{1i} + a_{2i} = a_{3i}$. To do this we make P compute two different ciphertexts, w_i and $\hat{w}_i$, for each choice v_i the voters computer computes as follows:

$$w_i = y_{1i}^t f(v_i)$$
$$\hat{w}_i = y_{3i}^t f(v_i)$$

We still compute $x = g^t$ as before. Now P sends this to B who stores $(x, w_1, \ldots, w_{k_{max}})$ and computes $\breve{x} = x^s$ and $\breve{w}_i = \hat{w}_i^s$ for all i before it sends $(\breve{x}, \breve{w}_1, \ldots, \breve{w}_{k_{max}})$ to R. In this brief description many details are left out. All the details can be found in the programs on later pages, but firstly we will show how we now will use Σ-protocols and commitments to prove that some of the computations are done correctly. The relation we are going to use for the Σ-protocols is as follows:S

$$\{(v, w) = ((\vec{r}, \vec{s}), w) \mid (\vec{r})^w = (r_0^w, \ldots, r_{k_{max}}^w) = \vec{s} = (s_0, \ldots, s_{k_{max}})\}$$

Firstly P will have to prove both to B and R that $(w_1, \ldots, w_{k_{max}})$ and $(\hat{w}_1, \ldots, \hat{w}_{k_{max}})$ contains the same ballot. Secondly B have to prove to R that we have produced $(\check{x}, \check{w}_1, \ldots, \check{w}_{k_{max}})$ correctly from $(x, \hat{w}_1, \ldots, \hat{w}_{k_{max}})$. Both of these proofs must be done without showing R $(\hat{w}_1, \ldots, \hat{w}_{k_{max}})$, since it then would be able to decrypt the ballot.

5.1 Proving *P* Computes Correctly

We begin by describing the communication between P and R to prove that $(w_1, \ldots, w_{k_{max}})$ and $(\hat{w}_1, \ldots, \hat{w}_{k_{max}})$ contains the same ballot. We simplify by doing computations for only one i. All the communication naturally goes trough B, and it also checks the proof to convince itself that $(w_1, \ldots, w_{k_{max}})$ and $(\hat{w}_1, \ldots,$ $\hat{w}_{k_{max}})$ contains the same ballot, but since it does not alter anything and checks the same as R, we simplify it to P and R communicating directly with eachother. It goes as follows:

$$
\begin{array}{lcr}
\mathbf{P} & - & \mathbf{R} \\
 & Common: & \\
 & g, \{y_{1i}\}, \{y_{3i}\} & \\
 & y_c, x, w_i, [\hat{w}_i] & \\
\text{Have:} & & \\
(\hat{x}, \hat{w}_i), t, \{r_i\}\ s.t. & & \\
[\hat{w}_i] = (g^{r_i}, y_c^{r_i} \hat{w}_i) & & \\
\text{Compute:} & & \\
r'_i, b \xleftarrow{r} \mathbb{Z}_{|G|} & & \\
\alpha_0 = g^b & & \\
\alpha_i = \left(\frac{y_{3i}}{y_{1i}}\right)^b & & \\
[\alpha_i] = (g^{r'_i}, y_c^{r'_i} \alpha_i) & \xrightarrow{a_0, [\alpha_i]} & \\
 & \xleftarrow{e} & e \xleftarrow{r} \mathbb{Z}_{|G|} \\
\text{Compute:} & & \\
z = te + b & & \\
R_i = r'_i + r_i e & \xrightarrow{z, R_i} & \text{Let} \\
 & & [w_i] = (1, w_i) \\
 & & \left[\frac{y_{3i}}{y_{1i}}\right] = \left(1, \frac{y_{3i}}{y_{1i}}\right) \\
 & & \text{Check:} \\
 & & \alpha_0 x^e g^{-z} = 1 \\
 & & [\alpha_i] \left(\frac{[\hat{w}_i]}{[w_i]}\right)^e \left[\frac{y_{3i}}{y_{1i}}\right]^{-z} = (g^{R_i}, y_c^{R_i})
\end{array}
$$

Figure 5.1: How to prove w_i and $\hat{w}_i$ contains the same ballot

Now we prove that this protocol gives us completeness, special soundness and special honest-verifier zero-knowledge (SHVZK), so we actually prove it is a Σ-protocol.

Completeness
We need to show that R will accept if both P and R follow the protocol with the given input. To show this we compute all the equalities to check that they are correct given that the protocol has been followed:

$$\begin{aligned}
\alpha_0 x^e g^{-z} &= g^b (g^t)^e g^{-(te+b)} = g^{b+te-te-b} = 1 \\
[\alpha_i] \left(\frac{[\hat{w}_i]}{[w_i]} \right)^e \left[\frac{y_{3i}}{y_{1i}} \right]^{-z} &= \left(g^{r'_i}, y_c^{r'_i} \left(\frac{y_{3i}}{y_{1i}} \right)^b \right) \left(g^{r_i}, y_c^{r_i} \frac{\hat{w}_i}{w_i} \right)^e \left(1, \frac{y_{3i}}{y_{1i}} \right)^{-z} \\
&= \left(g^{r'_i}, y_c^{r'_i} \left(\frac{y_{3i}}{y_{1i}} \right)^b \right) \left(g^{er_i}, y_c^{er_i} \left(\frac{y_{3i}}{y_{1i}} \right)^{te} \right) \\
&\quad \left(1, \left(\frac{y_{3i}}{y_{1i}} \right)^{-z} \right) \\
&= \left(g^{r'_i+er_i}, y_c^{r'_i+er_i} \left(\frac{y_{3i}}{y_{1i}} \right)^{b+te-z} \right) \\
&= \left(g^{R_i}, y_c^{R_i} \right)
\end{aligned}$$

Special Soundness
To show special soundness we will need to show that given two accepted conversations, with α_0 and $[\alpha_i]$ fixed, we can obtain the secret key t. So given two accepted conversations $(\alpha_0, [\alpha_i], e, z, R_i)$ and $(\alpha_0, [\alpha_i], \bar{e}, \bar{z}, \bar{R}_i)$ we compute as follows:

$$\begin{aligned}
\alpha_0 x^e g^{-z} &= 1 = \alpha_0 x^{\bar{e}} g^{-\bar{z}} \\
x^{e-\bar{e}} &= g^{z-\bar{z}} \\
x &= g^{(z-\bar{z})(e-\bar{e})^{-1}} \\
t &= (z-\bar{z})(e-\bar{e})^{-1} \pmod{|G|}
\end{aligned}$$

Special Honest-Verifier Zero-Knowledge
To prove SHVZK we need to make a simulator which given the common input and a random e can compute an accepting conversation with the same probability distribution as the conversation between a honest P and a honest R on the same common input. We do this by making a hybrid protocol between the real protocol and the simulator showing that they all give satisfactory conversations and that

the probability distributions are the same in the real protocol, the hybrid protocol and the simulator. It goes as follows:

Real	Hybrid	Simulator
Input:	Input:	Input:
e	e	e
Compute:	Compute:	Compute:
$r'_i, b \leftarrow \mathbb{Z}_{\lvert G\rvert}$	$R_i, z \leftarrow \mathbb{Z}_{\lvert G\rvert}$	$R_i, z \leftarrow \mathbb{Z}_{\lvert G\rvert}$
$z = te + b$	$b = z - te$	
$R_i = r'_i + r_i e$	$r'_i = R_i - r_i e$	
$\alpha_0 = g^b$	$\alpha_0 = g^z x^{-e}$	$\alpha_0 = g^z x^{-e}$
$\alpha_i = \left(\frac{y_{3i}}{y_{1i}}\right)^b$	$\alpha_i = \left(\frac{y_{3i}}{y_{1i}}\right)^b$	
$[\alpha_i] = (g^{r'_i}, y_c^{r'_i}\alpha_i)$	$[\alpha_i] =$	$[\alpha_i] =$
	$[1]_{R_i}\left[\frac{y_{3i}}{y_{1i}}\right]^z\left(\frac{[\hat{w}_i]}{[w_i]}\right)^{-e}$	$[1]_{R_i}\left[\frac{y_{3i}}{y_{1i}}\right]^z\left(\frac{[\hat{w}_i]}{[w_i]}\right)^{-e}$

Note that we have used that it is known that $[w_i] = (1, w_i)$ and $\left[\frac{y_{3i}}{y_{1i}}\right] = \left(1, \frac{y_{3i}}{y_{1i}}\right)$.

Now we argue that all of these give satisfactory conversations and the same probability distributions. Firstly we see that it is clear that the conversations in both the hybrid protocol and the simulator are satisfactory. So we argue that the probability distributions are the same in all three. We compare the probability distributions in the real protocol and the hybrid first.

Firstly we see that in the hybrid protocol we pick z and R_i random instead of r'_i and b, but since z and R_i are random, we see from the computations that r'_i and b are also random. And since r'_i and b are random in the real protocol, z and R_i are random in the real protocol.

Now $|G|$ is a group of prime order, so any non-identity element is a generator, and hence taking the power of a non-identity element in G by a random number is the same as picking a random element from G with uniform distribution. Furthermore, taking the product of random elements is still a random element, hence we see that α_0 and α_i are random elements in both the real and the hybrid protocol. By inspection one sees that still $[\alpha_i] = \text{com}_{y_c}(\alpha_i, r'_i)$ in the hybrid protocol, so $[\alpha_i]$ in the hybrid protocol has the same probability distribution as in the real protocol. Hence we see that the real protocol and the hybrid protocol has the same probability distributions.

Lastly we compare the probability distributions of the hybrid protocol and the simulator. R_i and z are generated as random elements in both and α_0 and $[\alpha_i]$ is computed in the same way in both. We also note that since we no longer use b and r'_i to generate the conversation, there is no point in generating them at all.

So the hybrid protocol and the simulator share the same probability distribution. Now, since both the real protocol and the simulator has the same probability distribution as the hybrid protocol, they must have the same probability distributions.

So we have made a simulator that makes a satisfactory conversation with the same probability distribution as a conversation in a real protocol with only the common information and e as input. Hence our protocol is SHVZK.

So since we have showed that the protocol satisfies completeness, special soundness and SHVZK, it is a Σ-protocol.

5.2 Proving B Computes Correctly

Now we describe how B proves to R that $(\check{x}, \check{w}_1, \ldots, \check{w}_{k_{max}})$ is computed correctly from $(x, \hat{w}_1, \ldots, \hat{w}_{k_{max}})$. Again we simplify to show how it goes for one i:

$$
\begin{array}{lcr}
\textbf{B} & - & \textbf{R} \\
 & Common: & \\
 & (\check{x}, \check{w}_i), \gamma, y_c, x & \\
 & g, [\hat{w}_i], [\check{w}_i] & \\
\text{Have:} & & \\
(\hat{x}, \hat{w}_i), r_i, q'_i \text{ s.t} & & \\
[\hat{w}_i] = (g^{r_i}, y_c^{r_i}\hat{w}_i) & & \\
[\check{w}_i] = (g^{q'_i}, y_c^{q'_i}\check{w}_i) & & \\
\text{Compute:} & & \\
\beta_1 = g^q, \beta_3 = x^q & & \\
\beta_{2i} = \hat{w}_i^q & & \\
[\beta_{2i}] = (g^{q''_i}, y_c^{q''_i}\beta_{2i}) & \xrightarrow{\beta_1,[\beta_{2i}],\beta_3} & \\
 & \xleftarrow{f} & f \overset{r}{\leftarrow} \mathbb{Z}_{|G|} \\
\text{Compute:} & & \\
z' = q + sf & & \\
R'_i = q''_i + fq'_i - z'r_i & \xrightarrow{z',R'_i} & \text{Check:} \\
 & & \beta_1\gamma^f g^{-z'} = 1 \\
 & & \beta_3\check{x}^f x^{-z'} = 1 \\
 & & [\beta_{2i}][\check{w}_i]^f[\hat{w}_i]^{-z'} = [1]_{R'_i} \\
 & & = com_{y_c}(1, R'_i)
\end{array}
$$

Figure 5.2: How to prove B has exponentiated with s

As before we prove that this protocol gives us completeness, special soundness and SHVZK, that is we prove it is a Σ-protocol as defined earlier.

Completeness
We show that if B and R are honest and follow the protocol, R will accept. The computations are as follows:

$$\begin{aligned}
\beta_1\gamma^f g^{-z'} &= g^q g^{sf} g^{-z'} = g^{z'-sf} g^{sf} g^{-z'} = 1 \\
\beta_3\check{x}^f x^{-z'} &= x^{z'-sf} x^{sf} x^{-z'} = x^{z'-sf+sf-z'} = 1 \\
[\beta_{2i}][\check{w}_i]^f[\hat{w}_i]^{-z'} &= (g^{q''_i}, y_c^{q''_i}\beta_{2i})(g^{q'_i}, y_c^{q'_i}\check{w}_i)^f(g^{r_i}, y_c^{r_i}\hat{w}_i)^{-z'} \\
&= (g^{q''_i}, y_c^{q''_i}\hat{w}_i^q)(g^{fq'_i}, y_c^{fq'_i}\check{w}_i^f)(g^{-z'r_i}, y_c^{-z'r_i}\hat{w}_i^{-z'}) \\
&= (g^{q''_i+fq'_i-z'r_i}, y_c^{q''_i+fq'_i-z'r_i}\hat{w}_i^{z'-sf}\hat{w}_i^{sf}\hat{w}_i^{-z'}) \\
&= (g^{R'_i}, y_c^{R'_i})
\end{aligned}$$

Special Soundness
We argue that given two conversations between B and R that are accepted, with β_1, $[\beta_{2i}]$ and β_3 fixed, will make us able to compute s. So assume $(\beta_1, [\beta_{2i}], \beta_3, f, z', R'_i)$ and $(\beta_1, [\beta_{2i}], \beta_3, \bar{f}, \bar{z}', \bar{R}'_i)$ are two accepted conversations, we compute as follows:

$$\begin{aligned}
\beta_1\gamma^f g^{-z'} &= 1 = \beta_1\gamma^{\bar{f}} g^{-\bar{z}'} \\
\gamma^f g^{-z'} &= \gamma^{\bar{f}} g^{-\bar{z}'} \\
\gamma^{f-\bar{f}} &= g^{z'-\bar{z}'} \\
\gamma &= g^{(z'-\bar{z}')(f-\bar{f})^{-1}}
\end{aligned}$$

So $s = (z' - \bar{z}')(f - \bar{f})^{-1} \pmod{|G|}$. And hence we have proved that we can find the secret s given two accepted conversations and the common input, so we have special soundness.

SHVZK
Again, to prove SHVZK, we need to make a simulator which given a random f and the common input can compute a satisfactory conversation with the same probability distribution as the conversations happening in the real protocol when both B and R are honest. As before, we make a hybrid protocol between the real protocol and the simulator to make it easier to argue that the probability distributions are the same. We get the following:

Real	Hybrid	Simulator
Input:	Input:	Input:
f	f	f
Compute:	Compute:	Compute:
$q, q_i'' \leftarrow \mathbb{Z}_{\vert G\vert}$	$R_i', z' \leftarrow \mathbb{Z}_{\vert G\vert}$	$R_i', z' \leftarrow \mathbb{Z}_{\vert G\vert}$
$z' = q + sf$	$q = z' - sf$	
$\beta_1 = g^q$	$\beta_1 = g^{z'}\gamma^{-f}$	$\beta_1 = g^{z'}\gamma^{-f}$
$\beta_{2i} = \hat{w}_i^q$	$\beta 2i = \hat{w}_i^{z'}\check{w}_i^{-f}$	
$[\beta 2i] = (g^{q_i''}, y_c^{q_i''}\beta_{2i})$	$[\beta 2i] =$	$[\beta 2i] =$
	$[1]_{R_i'}[\hat{w}_i]^{z'}[\check{w}]^{-f}$	$[1]_{R_i'}[\hat{w}_i]^{z'}[\check{w}]^{-f}$
$R_i' = q_i'' + fq_i' - z'r_i$	$q_i'' = R_i' - fq_i' + z'r_i$	
$\beta_3 = x^q$	$\beta_3 = x^{z'}\check{x}^{-f}$	$\beta_3 = x^{z'}\check{x}^{-f}$

Clearly both the real protocol, the hybrid protocol and the simulator generates satisfactory conversations, so we argue that the probability distribution of the different conversations are the same. We note that we only need to generate the messages B is supposed to send to R, that is we must generate β_1, $[\beta_{2i}]$, β_3, z' and R_i'. We do not need to generate numbers B uses to compute messages if we no longer need them in our computation of the messages.

In the real protocol we choose q, q_i'' at random. f is random, the product of a number with a random number is a random number and the sum of two random numbers is a random number. So z' and R_i' in the real protocol are random. In the hybrid protocol we pick R_i' and z at random, and use these to generate q and q_i''. Again, by the arguments just mentioned, q and q_i'' must be random.

In the real protocol β_1, β_{2i} and β_3 are random elements generated uniformly over G since q is a random number. In the hybrid protocol β_1, β_{2i} and β_3 are all the product of two random elements distributed uniformly over G (z' and f are random numbers), and hence random elements distributed uniformly over G. Certainly $[\beta_{2i}]$ is a random element from $G \times G$ with uniform distribution in the real protocol, and the same $[1]_{R_i'}$, $[\hat{w}_i]^{z'}$ and $[\check{w}_i]^{-f}$ is in the hybrid protocol. Multiplying together three random elements from $G \times G$ gives a random element in $G \times G$, hence $[\beta_{2i}]$ is a random element in $G \times G$ with uniform distribution in both the real and the hybrid protocol. So we see that the real protocol and the hybrid protocol have the same probability distributions.

Now we argue that the hybrid protocol has the same probability distributions as the simulator. Since every computation done in the simulator is exactly the same as the ones done in the hybrid protocol (we just don't generate q, q_i'' and β_{2i}), and z and R_i' is generated in the same way, it is obvious that the computed messages in the hybrid protocol and the simulator have the same probability distributions.

So the messages generated by the real protocol must have the same probability distributions as the messages generated by the simulator, since the messages from both have the same probability distributions as the messages generated in the hybrid protocol. The simulator only uses the input f and the common information, so we have SHVZK.

So we have shown that the last protocol has completeness, soundness and SHVZK, it is therefore a Σ-protocol.

5.3 Why The Protocols Fulfil Our Requirements

So we have proved that both protocols are Σ-protocols, but it can be somewhat difficult to see why our protocols realize our requirements. That is, why the ciphertexts have to contain the same ballots because of the first protocol and why B is forced to exponentiate by s because of the second protocol. We are not going to go into an elaborate proof of why this must be, but we will give an informal discussion for both cases here. This is to make it a little bit more clear why they realize the wanted requirements. We start with the first protocol, which is supposed to prove to R that P knows the contents of the ballots and that w_i and $\hat{w}_i$ contain the same contents.

Why P is Forced to put the Same Ballot in Both Ciphertexts

The soundness argument shows that given two accepting conversations $(\alpha_0, [\alpha_i], e, z, R_i)$ and $(\alpha_0, [\alpha_i], \bar{e}, \bar{z}, \bar{R}_i)$ we are able to find a $t' = (z - \bar{z})(e - \bar{e})^{-1}$ such that $g^{t'} = x$. Obviously $w_i = y_{1i}^{t'} m$ and $\hat{w}_i = y_{3i}^{t'} \hat{m}$ for some $m, \hat{m} \in G$, where m and $\hat{m}$ not necessarily is the same element.

Let us assume for a moment we had dropped the commitments and that we had obtained this t'. We also then have the equations $\alpha_i \left(\frac{\hat{w}_i}{w_i}\right)^e \left(\frac{y_{3i}}{y_{1i}}\right)^{-z} = 1$ and $\alpha_i \left(\frac{\hat{w}_i}{w_i}\right)^{\bar{e}} \left(\frac{y_{3i}}{y_{1i}}\right)^{-\bar{z}} = 1$. We then do the following calculations:

$$\begin{aligned}
\alpha_i \left(\frac{\hat{w}_i}{w_i}\right)^e \left(\frac{y_{3i}}{y_{1i}}\right)^{-z} &= \alpha_i \left(\frac{\hat{w}_i}{w_i}\right)^{\bar{e}} \left(\frac{y_{3i}}{y_{1i}}\right)^{-\bar{z}} \\
\left(\frac{\hat{w}_i}{w_i}\right)^e \left(\frac{y_{3i}}{y_{1i}}\right)^{-z} &= \left(\frac{\hat{w}_i}{w_i}\right)^{\bar{e}} \left(\frac{y_{3i}}{y_{1i}}\right)^{-\bar{z}} \\
\left(\frac{\hat{w}_i}{w_i}\right)^{e-\bar{e}} &= \left(\frac{y_{3i}}{y_{1i}}\right)^{z-\bar{z}} \\
\frac{\hat{w}_i}{w_i} &= \left(\frac{y_{3i}}{y_{1i}}\right)^{(z-\bar{z})(e-\bar{e})^{-1}}
\end{aligned}$$

$$\frac{\hat{w}_i}{w_i} = \left(\frac{y_{3i}}{y_{1i}}\right)^{t'}$$

But as we said above we also have $w_i = y_{1i}^{t'} m$ and $\hat{w}_i = y_{3i}^{t'} \hat{m}$, this gives us:

$$\begin{aligned} \frac{\hat{w}_i}{w_i} = \frac{y_{3i}^{t'}\hat{m}}{y_{1i}^{t'}m} &= \left(\frac{y_{3i}}{y_{1i}}\right)^{t'} \frac{\hat{m}}{m} \\ \left(\frac{y_{3i}}{y_{1i}}\right)^{t'} \frac{\hat{m}}{m} &= \left(\frac{y_{3i}}{y_{1i}}\right)^{t'} \\ \frac{\hat{m}}{m} &= 1 \\ \hat{m} &= m \end{aligned}$$

So we see that if $\alpha_i \left(\frac{\hat{w}_i}{w_i}\right)^e \left(\frac{y_{3i}}{y_{1i}}\right)^z = 1$ holds then $\hat{w}_i$ and w_i contains the same ballot. So if we could show that this equation must hold when $[\alpha_i]\left(\frac{[\hat{w}_i]}{[w_i]}\right)^e \left[\frac{y_{3i}}{y_{1i}}\right]^{-z} = [1]_{R_i} = (g^{R_i}, y_c^{R_i})$ holds, then we would be done. Now we can look at $[\alpha_i]$, $\frac{[\hat{w}_i]}{[w_i]}$, $\left[\frac{y_{3i}}{y_{1i}}\right]$ and $[1]_{R_i}$ as elements in $G \times G/\langle(g, y_c)\rangle$. The equation still holds in this group. Now define

$$\begin{cases} \psi : & G \times G/\langle(g, y_c)\rangle & \longrightarrow & G \\ & (g^r, y_c^r m) & \longmapsto & m \end{cases}$$

It can be checked that this is a group isomorphism. Since this is an isomorphism, and $[\alpha_i]\left(\frac{[\hat{w}_i]}{[w_i]}\right)^e \left[\frac{y_{3i}}{y_{1i}}\right]^{-z} = [1]_{R_i}$ holds in $G \times G/\langle(g, y_c)\rangle$ we must have that the equation $\alpha_i \left(\frac{\hat{w}_i}{w_i}\right)^e \left(\frac{y_{3i}}{y_{1i}}\right)^{-z} = 1$ must hold in G. So therefore $\hat{w}_i$ and w_i must contain the same ballot. And since we know t', we can find m. So P knows the contents of the ciphertexts.

Why B has to Exponentiate by s

By the soundness argument we can find $s' = (z' - \bar{z}')(f - \bar{f})^{-1}$ such that $\gamma = g^{s'}$ given that we have two accepted conversations $(\beta_1, [\beta_{2i}], \beta_3, f, z', R_i')$ and $(\beta_1, [\beta_{2i}], \beta_3, \bar{f}, \bar{z}', \bar{R}_i')$. Again we will assume for a while that we drop the commitments. Then we have the equations $\beta_3 \check{x}^f x^{-z'} = 1$, $\beta_3 \check{x}^{\bar{f}} x^{-\bar{z}'} = 1$, $\beta_{2i} \check{w}_i^f \hat{w}_i^{-z'} = 1$ and $\beta_{2i} \check{w}_i^{\bar{f}} \hat{w}_i^{-\bar{z}'} = 1$. We then compute as follows:

$$\begin{aligned}
\beta_{2i}\breve{w}_i^f \hat{w}_i^{-z'} &= \beta_{2i}\breve{w}_i^{\bar{f}} \hat{w}_i^{-\bar{z}'} \\
\breve{w}_i^f \hat{w}_i^{-z'} &= \breve{w}_i^{\bar{f}} \hat{w}_i^{-\bar{z}'} \\
\breve{w}_i^{f-\bar{f}} &= \hat{w}_i^{z'-\bar{z}'} \\
\breve{w}_i &= \hat{w}_i^{(z'-\bar{z}')(f-\bar{f})^{-1}} \\
\breve{w}_i &= \hat{w}_i^{s'} \\
\beta_3\breve{x}^f x^{-z'} &= \beta_3\breve{x}^{\bar{f}} x^{-\bar{z}'} \\
\breve{x}^{f-\bar{f}} &= x^{z'-\bar{z}'} \\
\breve{x} &= x^{(z'-\bar{z}')(f-\bar{f})^{-1}} \\
\breve{x} &= x^{s'}
\end{aligned}$$

And since B has no possibility to corrupt the γ R receives before the election starts we see that B must have exponentiated correctly by $s' = s$ if $\beta_{2i}\breve{w}_i^f \hat{w}_i^{-z'} = 1$ and $\beta_{2i}\breve{w}_i^{\bar{f}} \hat{w}_i^{-\bar{z}'} = 1$ holds. Using the same group isomorphism ψ we used to prove that w_i and $\hat{w}_i$ contains the same ballot we see that these two equations hold when $[\beta_{2i}][\breve{w}_i]^f[\hat{w}_i]^{-z'} = [1]_{R'_i}$ and $[\beta_{2i}][\breve{w}_i]^{\bar{f}}[\hat{w}_i]^{-\bar{z}'} = [1]_{\bar{R}'_i}$ holds. These two equations hold if the conversations gets accepted, hence B must have computed correctly and used s as exponent when generating $\breve{x}$ and the $\breve{w}_i$'s.

CHAPTER 6

CHANGES TO THE PROTOCOL

In this chapter we give the new programs, with both changes implemented, that together is the new protocol. Then we analyse this new protocol in the next chapter. We have used the programs given in Chapter 4 as basis for the changes.

6.1 Key Generation

The only change to the key generation functionality is that we do not generate $\{a_{2i}\}$ and $\{y_{2i}\}$ and that now every player gets $\{y_{1i}\}$ and $\{y_{3i}\}$. Therefore we do not mention anything more on the key generation functionality here.

6.2 The Voter's Computer

The voter's computer receives the ballot from the voter, encrypts it two times with different sets of public keys, generate the first part of the conversation in the Σ-protocol and signs on all of this. All this information is sent to B via $\mathcal{F}_{eid}$. Then the computer awaits the answer from R on the Σ-protocol (the answer goes via B), computes the answer to this, and returns his answer to B. At last B awaits the final receipt from R, checks it, and, if valid, outputs to the voter that the ballot was accepted. The new program for the voter's computer is as follows:

```
On $(\mathsf{keys}, \{y_{1i}\}, \{y_{3i}\}, f)$ from $\mathcal{F}_{key}$:
 1: Store $\{y_{1i}\}$, $\{y_{3i}\}$ and $f$.
On $(\mathsf{vote}, v_1, \dots, v_k)$ from $V$:
 1: Send $(\mathsf{establish}, V, B)$ to $\mathcal{F}_{eid}$ and wait for $(\mathsf{established}, sid)$.
 2: Set $v_{k+1} = \cdots = v_{k_{max}} = 0$.
 3: Compute $x = g^t$, $t \xleftarrow{r} \mathbb{Z}_{|G|}$.
 4: Compute $y_c \xleftarrow{r} G$.
 5: Pick $b \xleftarrow{r} \mathbb{Z}_{|G|}$.
 6: Compute $\alpha_0 = g^b$.
 7: for $i = 1$ to $k_{max}$ :
 8:         Compute $w_i = y_{1i}^t f(v_i)$ and $\hat{w}_i = y_{3i}^t f(v_i)$.
 9:         Compute $[\hat{w}_i] = com_{y_c}(\hat{w}_i, r_i), r_i \xleftarrow{r} \mathbb{Z}_{|G|}$.
10:         Compute $\alpha_i = \left(\frac{y_{3i}}{y_{1i}}\right)^b$.
11:         Compute $[\alpha_i] = com_{y_c}(\alpha_i, r'_i)$, $r'_i \xleftarrow{r} \mathbb{Z}_{|G|}$.
12: Send $(\mathsf{sign}, V, (x, w_1, \dots, w_{k_{max}}), ([\hat{w}_1], \dots, [\hat{w}_{k_{max}}]), ([\alpha_1], \dots, [\alpha_{k_{max}}]), \alpha_0)$ to
    $\mathcal{F}_{eid}$, and wait for $(\mathsf{signature}, \dots, \sigma_V)$ from $\mathcal{F}_{eid}$.
13: Send $(\mathsf{send}, sid, (\mathsf{vote}, V, (x, w_1, \dots, w_{k_{max}}), ([\hat{w}_1], \dots, [\hat{w}_{k_{max}}]),$
    $([\alpha_1], \dots, [\alpha_{k_{max}}]), \alpha_0, \sigma_V), (\hat{w}_1, \dots, \hat{w}_{k_{max}}), (r_1, \dots, r_{k_{max}}), y_c)$ to $\mathcal{F}_{eid}$,
    and wait for $(\mathsf{response}, sid, e)$ from $\mathcal{F}_{eid}$.
14: Compute $z = te + b$.
15: for $i = 1$ to $k_{max}$ :
16:         Compute $R_i = r'_i + r_i e$.
17: Send $(\mathsf{opening}, sid, V, z, (R_1, \dots, R_{k_{max}}))$, and wait for $(\mathsf{recv}, sid, (\mathsf{receipt}, \sigma_R))$
    from $\mathcal{F}_{eid}$.
18: Compute $h_b \leftarrow Hash(V, (x, w_1, \dots, w_{k_{max}}), ([\hat{w}_1], \dots, [\hat{w}_{k_{max}}]),$
    $([\alpha_1], \dots, [\alpha_{k_{max}}]), \alpha_0, \sigma_V)$.
19: Send $(\mathsf{verify}, R, h_b, \sigma_R)$ to $\mathcal{F}_{eid}$ and wait for $(\mathsf{verified}, R, h_b, \sigma_R)$.
20: Send $(\mathsf{accepted})$ to $V$.
```

Program 7: Updated program for the voters computer

6.3 The Ballot Box

The ballot box does nothing until it has received it's codes from KG. When a computer wishes to submit a ballot, the ballot box connects to the computer. Then the ballot box receives the encrypted ballot in two versions, and in addition B receives the first part of the conversation in the Σ-protocol proving that P has computed correctly. Note that all information that is going unchanged to R is signed on, so B must check the signature on this. Then B computes the ciphertexts R are going use to make the receipt codes, and the contents of the first conversation of the second Σ-protocol proving B has computed correctly. Then B sends to R the contents of the first part of the conversation P and R have to prove that P has computed correctly, the ciphertexts R is going to use to create the receipt codes and the contents of the first part of the conversation B makes with R to prove it has computed correctly. Then B awaits the response from R on both Σ-protocols, sends the response P needs to P and awaits the last part of the conversation of the Σ-protocol between P and R from P. It then checks the answer from P to see that it becomes a Σ-protocol, computes the answer in the Σ-protocol between B and R, and sends both responses back to R. He then awaits the receipt from R, checks it, and sends it to P.

When the ballot box is told to close, it awaits for ongoing submissions to complete, sends the ballots to be counted to D and send it's contents to A. The new program for the ballot box, with both changes implemented, is as follows:

```
Do nothing until $(\mathsf{keys}, \{y_{1i}\}, \{y_{3i}\}, \{(V,s)\})$ has been received from $\mathcal{F}_{key}$, then
    do:
 1: Record $\{y_{1i}\}$, $\{y_{3i}\}$ and the pairs $(V,s)$.
On $(\mathsf{established}, sid, V, P)$ from $\mathcal{F}_{eid}$:
 1: Wait for $(\mathsf{recv}, sid, P, ((x, w_1, \ldots, w_{k_{max}}), ([\hat{w}_1], \ldots, [\hat{w}_{k_{max}}]),$
    $([\alpha_1], \ldots, [\alpha_{k_{max}}]), \alpha_0, \sigma_V), (\hat{w}_1, \ldots, \hat{w}_{k_{max}}), (r_1, \ldots, r_{k_{max}}), y_c)$ from $\mathcal{F}_{eid}$.
 2: Send $(\mathsf{verify}, V, (x, w_1, \ldots, w_{k_{max}}), ([\hat{w}_1], \ldots, [\hat{w}_{k_{max}}]), ([\alpha_1], \ldots, [\alpha_{k_{max}}]), \alpha_0,$
    $\sigma_V)$ to $\mathcal{F}_{eid}$ and wait for $(\mathsf{verified}, \ldots, \sigma_V)$ from $\mathsf{F}_{eid}$.
 3: Look up the stored pair $(V,s)$ and place an exclusive lock on the pair (waiting
    for any other session to release it's exclusive lock).
 4: Select the next sequence number $seq$.
 5: Compute $\check{x} = x^s$.
 6: Compute $\beta_1 = g^q$, $q \xleftarrow{r} \mathbb{Z}_{|G|}$.
 7: for $i = 1$ to $k_{max}$ :
 8:         Compute $\check{w}_i = \hat{w}_i^s$.
 9:         Chose $q_i', q_i'' \xleftarrow{r} \mathbb{Z}_{|G|}$.
10:         Compute $[\check{w}_i] = \mathrm{com}_{y_c}(\check{w}_i, q_i')$.
11:         Compute $\beta_{2i} = \hat{w}_i^q$ and $[\beta_{2i}] = \mathrm{com}_{y_c}(\beta_{2i}, q_i'')$.
12: Compute $\beta_3 = x^q$.
13: Send $(\mathsf{ballot}, seq, V, ((x, w_1, \ldots, w_{k_{max}}), ([\hat{w}_1], \ldots, [\hat{w}_{k_{max}}]),$
    $([\alpha_1], \ldots, [\alpha_{k_{max}}]), \alpha_0, \sigma_V), (\check{x}, \check{w}_1, \ldots, \check{w}_{k_{max}}), ([\check{w}_1], \ldots, [\check{w}_{k_{max}}]), \beta_1,$
    $([\beta_{21}], \ldots, [\beta_{2k_{max}}]), \beta_3, y_c)$ to $R$ and wait for $(\mathsf{response}, seq, V, e, f)$ from $R$.
14: Send $(\mathsf{response}, sid, P, e)$ to $\mathcal{F}_{eid}$ and wait for $(\mathsf{opening}, sid, V, z,$
    $(R_1, \ldots, R_{k_{max}}))$ from $\mathcal{F}_{eid}$.
```

```
15: Compute $z' = q + sf$.
16: Check that $\alpha_0 x^e g^{-z} = 1$.
17: for $i = 1$ to $k_{max}$ :
18:         Let $[w_i] = (1, w_i)$ and $[y_i^*] = \left(1, \frac{y_{3i}}{y_{1i}}\right)$.
19:         Check that $[\alpha_i] \left(\frac{[\hat{w}_i]}{[w_i]}\right)^e [y_i^*]^{-z} = com_{y_c}(1, R_i)$.
20:         Compute $R'_i = q''_i + fq'_i - z'r_i$.
21: Send $(\mathsf{opening}, seq, V, z, z', (R_1, \ldots, R_{k_{max}}), (R'_1, \ldots, R'_{k_{max}}))$ to $R$ and wait
    for $(\mathsf{receipt}, seq, \sigma_R)$ from $R$.
22: Compute $h_b \leftarrow Hash(V, (x, w_1, \ldots, w_{k_{max}}), ([\hat{w}_1], \ldots, [\hat{w}_{k_{max}}]),$
    $([\alpha_1], \ldots, [\alpha_{k_{max}}]), \alpha_0, \sigma_V)$.
23: Send $(\mathsf{verify}, R, h_b, \sigma_R)$ to $\mathcal{F}_{eid}$ and wait for $(\mathsf{verified}, R, h_b, \sigma_R)$ from $\mathcal{F}_{eid}$.
24: Store $(seq, V, (x, w_1, \ldots, w_{k_{max}}), ([\hat{w}_1], \ldots, [\hat{w}_{k_{max}}]), ([\alpha_1], \ldots, [\alpha_{k_{max}}]), \alpha_0,$
    $\sigma_V, z, (R_1, \ldots, R_{k_{max}}))$ and release the lock on the record $(V, s)$.
25: Send $(\mathsf{receipt}, \sigma_R)$ to $P$.
On $(\mathsf{count})$ from $D$:
1: Stop processing $(\mathsf{established}, \ldots)$ messages from $\mathcal{F}_{eid}$.
2: Stop any voting sessions that have not yet reached Step 4 and wait for re-
   maining sessions to terminate.
3: Send $(\mathsf{count})$ to $R$.
4: Let $S_{bb}$ be the list of all recorded entries $(seq, V, (x, w_1, \ldots, w_{k_{max}}),$
   $([\hat{w}_1], \ldots, [\hat{w}_{k_{max}}]), ([\alpha_1], \ldots, [\alpha_{k_{max}}]), \alpha_0, \sigma_V, z, (R_1, \ldots, R_{k_{max}}))$.
5: For each voter $V$, find the recorded entry with the largest
   sequence number $seq$ and extract the ballot $(x, w_1, \ldots, w_{k_{max}})$.
   Compute
       $$w = \prod_{i=1}^{k_{max}} w_i$$
   and add $(x, w)$ to the list $L$.
6: Sort $L$. Send $(\mathsf{decrypt}, L)$ to the decryption service $D$, and $(\mathsf{content},$
   $S_{bb})$ to $A$.
```

Program 8: Updated program for the ballot box

6.4 The Receipt Generator

The receipt generator receives from the ballot box the contents of the first part of the conversation needed to prove that P has made two ciphertexts containing the same ballot, the ciphertexts it needs to generate receipt codes and the contents of the first part of the conversation needed to prove B has computed correctly. It checks the signature and the sequence number, and replies to B with the answers for the two Σ-protocols, then it awaits the response with the last part of the conversation of the two Σ-protocols. Then R checks that both Σ-protocols are fulfilled and thereafter generates the receipt codes, which are sent directly to the voter. The last step during a vote submission is that the receipt generator stores the name of the voter, the sequence number and a hash of the ballots and the first part of the conversation proving that P has computed correctly, and then signs on the hash and sends the signature to B.

When the election is finished and the receipt generator is told to close, it sends the list of voters, sequence numbers and hashes to the auditor. The new program for the receipt generator, with both changes implemented, is as follows:

```
On $(\mathsf{keys}, \{y_{1i}\}, \{a_{3i}\}, \{y_{3i}\}, \{(V, \gamma, d)\})$ from $\mathcal{F}_{key}$:
1: Record $\{y_{1i}\}$, $\{a_{3i}\}$ and the triples $(V, \gamma, d)$.
On $(\mathsf{ballot}, seq, V, ((x, w_1, \dots, w_{k_{max}}), ([\hat{w}_1], \dots, [\hat{w}_{k_{max}}]), ([\alpha_1], \dots, [\alpha_{k_{max}}]), \alpha_0,$
   $\sigma_V), (\check{x}, \check{w}_1, \dots, \check{w}_{k_{max}}), ([\check{w}_1], \dots, [\check{w}_{k_{max}}]), \beta_1, ([\beta_{21}], \dots, [\beta_{2k_{max}}]), \beta_3, y_c)$
   from $B$:
1: Compute $h_b \leftarrow Hash(V, (x, w_1, \dots, w_{k_{max}}, ([\hat{w}_1], \dots, [\hat{w}_{k_{max}}]),$
   $([\alpha_1], \dots, [\alpha_{k_{max}}]), \alpha_0, \sigma_V)$ and $h'_b \leftarrow Hash(V, x, w_1, \dots, w_{k_{max}})$.
2: Look up the recorded tuple $(V, \gamma, d)$ and place an exclusive lock on the tuple
   (waiting for any other session to release it's exclusive lock).
3: Verify that no record $(\cdot, \cdot, \cdot, h'_b)$ or $(V, seq', \cdot, \cdot)$ with $seq' \geq seq$ exists.
4: Send $(\mathsf{verify}, V, (x, w_1, \dots, w_{k_{max}}), ([\hat{w}_1], \dots, [\hat{w}_{k_{max}}]), ([\alpha_1], \dots, [\alpha_{k_{max}}]), \alpha_0,$
   $\sigma_V)$ to $\mathcal{F}_{eid}$ and wait for $(\mathsf{verified}, \dots, \sigma_V)$.
5: Chose $e, f \xleftarrow{r} \mathbb{Z}_{|G|}$.
6: Send $(\mathsf{response}, seq, V, e, f)$ to $B$ and wait for $(\mathsf{opening}, seq, V, z, z',$
   $(R_1, \dots, R_{k_{max}}), (R'_1, \dots, R'_{k_{max}}))$ from $B$.
7: for $i = 1$ to $k_{max}$ :
8:       Let $[w_i] = (1, w_i)$ and $[y^*_i] = \left(1, \frac{y_{3i}}{y_{1i}}\right)$.
9:       Check that $[\alpha_i] \left(\frac{[\hat{w}_i]}{[w_i]}\right)^e [y^*_i]^{-z} = com_{y_c}(1, R_i)$.
10:      Check that $[\beta_{2i}][\check{w}_i]^f [\hat{w}_i]^{-z'} = com_{y_c}(1, R'_i)$.
11: Check that $\alpha_0 x^e g^{-z} = 1$.
12: Check that $\beta_1 \gamma^f g^{-z'} = 1$.
13: Check that $\beta_3 \check{x}^f x^{-z'} = 1$.
14: Send $(\mathsf{sign}, R, h_b,)$ to $\mathcal{F}_{eid}$ and wait for $(\mathsf{signature}, R, h_b, \sigma_R)$.
15: Record $(V, seq, h_b, h'_b)$. Send $(\mathsf{receipt}, seq, \sigma_R)$ to $B$.
16: Send $(\mathsf{receipt}, \check{r}, \dots, \check{r}_k)$ to $V$.
On $(\mathsf{count})$ from $B$:
1: Verify that all sessions have terminated.
```

```
2: Send (flush) to $\mathsf{F}_{sc}$.
3: Let $S_R$ be the list of all recorded entries $(V, seq, h_b, h'_b)$. Send (hashes, $S_R$) to
   $A$.
```

Program 9: Updated program for the receipt generator

6.5 The Decryption Service

There is no change in the messages D receives compared to the program given earlier in this paper, and hence no changes in the functionality of D. So we do not mention it further here.

6.6 The Auditor

There are only two changes to the program we have given earlier in this paper, and that is that the auditor now also receives and stores $\{y_{3i}\}$ and, instead of checking the proofs of knowledge, must check that the equations $\alpha_0 x^e g^{-z} = 1$ and $[\alpha_i]\left(\frac{[\hat{w}_i]}{[w_i]}\right)^e [y*_i]^{-z} = com_{y_c}(1, R_i)$ hold, where still $[y_i^*] = \left(1, \frac{y_{3i}}{y_{1i}}\right)$. We do not comment more on this in this paper.

CHAPTER 7

ANALYSIS OF THE NEW PROTOCOL

We mention here that when we are doing the analysis of the new system, we model the generation of the Σ-protocols as something $\mathcal{F}_{pok}$ does, and at all times assume that $\mathcal{F}_{pok}$ is a trusted third party. The way we model it is that the player wanting to make a conversation gives $\mathcal{F}_{pok}$ the common input, and in addition the special input the prover has, including the secret the prover has (in our case the secret is either t or s). Then $\mathcal{F}_{pok}$ generates the proof and returns it as π. When verifying a proof the verifier sends the common input and π to $\mathcal{F}_{pok}$ and then $\mathcal{F}_{pok}$ responds with $\mathsf{verified}$ or $\mathsf{invalid}$. Commitments are still done by the players.

7.1 Voters, Computers and the Ballot Box

As before, we try to prove the same conditions as stated in the original protocol [5] by Gjøsteen. They are as follows:

- If the ballot box is not corrupt, the auditor will not fail the election
- For any honest voter that uses only honest computers, any ballot accepted as cast and not superseded should be counted if the auditor accepts the election. The ballot remains confidential regardless.

- If an honest voter uses a corrupt computer (not necessarily for voting), nothing can be guaranteed for voters that submit multiple ballots. However, for voters that submit exactly one ballot and accepts that ballot as cast, with high probability that ballot will be counted unless the voter observes an attack. If the ballot was submitted through an honest computer, the ballot remains confidential.

Game 1 We start by making a machine M that has all the information the honest players have, and that can simulate all the players. So in this game M plays the role of every honest player. Note that M has the private keys $\{a_{1i}\}$ and $\{a_{3i}\}$. Clearly this is indistinguishable from the real protocol.

Game 2 Since we know M is honest, there is no reason for M to do the mixing of encrypted ballots and proving the correctness of decryptions. In a later game we will make M encrypt random group elements, and then use cleartext ballots instead of decrypting ciphertexts. Then M will not be able to make the correctness proofs of decryption, hence we will need the fact that M does not do these proofs anymore. So now we decrypt the encrypted ballots, and then shuffle the decrypted ballots. An adversary cannot observe the communication between D and A, so therefore this game is indistinguishable from the previous one.

Game 3 We will later need to assume that we have a injective $Hash(\cdot)$ function. So we make M abort if it ever observes a collision in $Hash(\cdot)$. If $Hash(\cdot)$ is collision resistant (which we assume it is in the real protocol) this game is clearly indistinguishable from the previous game.

From now on we treat $Hash(\cdot)$ as a injective function in our analysis.

Game 4 Now we wish to randomize the function d used to generate receipt codes. This is done because in the next game we wish to change the input d gets to generate receipt codes. Then we will need the fact that d is a completely random function to argue indistinguishability when having changed the input. Since F is a pseudo-random function family, this game is indistinguishable to the previous one.

Game 5 In this game we want to eliminate the dependency to the per voter exponent s. So in this game M changes the return codes sent back to the voter to $d(f(v_i))$ instead of $d(f(v_i)^s)$. Now M computes the return code as $d(w_i x^{-a_{1i}})$.

As in the original protocol, these changes are only observable if $(w_i x_i^{-a_{1i}})^s \neq \check{w}_i \check{x}^{-a_{3i}}$. Now as long as the Σ-protocol proving correct behaviour when B computes $(\check{x}, \check{x}_1, \ldots, \check{x}_{k_{max}})$ holds and the Σ-protocol proving that P has created two ciphertexts containing the same ballots holds, the soundness argument of the two Σ-protocols together gives that this holds except with negligible probability. So this game is indistinguishable to the previous game. Note that now the decryption keys $\{a_{3i}\}$ are no longer in use.

Game 6 It is now time to change the computation of x, w_i and $\hat{w}_i$ that simulated honest computers do such that we can generate a random $\hat{w}_i$ in the next game and prove indistinguishability with a DDH-reduction. So for ballots encrypted by a simulated honest computer we change the computation of x, w_i and $\hat{w}_i$ to the following:

$$\begin{aligned} x &= g^t \\ w_i &= x^{a_{1i}} f(v_i) \\ \hat{w}_i &= y_{3i}^t f(v_i) \end{aligned}$$

In addition, simulated honest computers now provides $\mathcal{F}_{pok}$ with a random witness instead of t.

Clearly, this game is indistinguishable to the previous one as $\mathcal{F}_{pok}$ no longer uses the witness and the end results of the computations are the same.

Game 7 We are now in a position where we wish to compute random $\hat{w}_i$'s when simulated honest computers are encrypting ballots, to start randomizing what the ballot box receives from simulated honest computers. So we change the computations of x, w_i and $\hat{w}_i$ to the following for simulated honest computers:

$$\begin{aligned} x &\overset{r}{\leftarrow} G \\ w_i &= x^{a_{1i}} f(v_i) \\ \hat{w}_i &\overset{r}{\leftarrow} G \end{aligned}$$

Claim. A distinguisher between game 6 and 7 will result in an adversary for the DDH problem.

Proof. We will prove that a distinguisher between the two games gives an adversary for the subgroup membership problem $H_0^{k_{max}+1} \overset{?}{\leftrightarrow} G^{k_{max}+1}$ defined in Chapter 2, where $H_0^{k_{max}+1} = \langle (g, y_{31}, \ldots, y_{3k_{max}}) \rangle$. By Theorem 2.8 this gives an adversary for the DDH problem. So take a tuple $u = (u_0, \ldots, u_{k_{max}})$, where $u \in H_0^{k_{max}+1}$ or $u \in G^{k_{max}+1}$. Let $r, t \overset{r}{\leftarrow} \mathbb{Z}_{|G|}$, and compute as follows:

$$x = g^t u_0^r$$

$$\hat{w}_i = y_{3i}^t u_i^r f(v_i), \forall i \in \{1, \ldots, k_{max}\}$$

If $u \in H_0^{k_{max}+1}$, the computations are as in Game 6, and if $u \in G^{k_{max}+1}$, the computations go as in Game 7. So a distinguisher between the games gives an adversary for the problem $H_0^{k_{max}+1} \overset{?}{\leftrightarrow} G^{k_{max}+1}$. □

Game 8 It is now possible to start getting into a position where we no longer use the the decryption keys $\{a_{1i}\}$, this will be needed in order for us to later be able to randomize the w_i's. So in this game, when honest simulated computers encrypt ballots, M stores the cleartext ballot. When such ballots are received by the receipt generator, the receipt generator uses the cleartext ballots instead of decrypting and when encrypted ballots from simulated honest computers is sent for final decryption, the decryption service uses the cleartext ballots instead of decrypting.

When an adversarially generated encrypted ballot is received by the simulated R it verifies the proof π. This forces the the adversary to give $\mathcal{F}_{pok}$ a witness p such that $x = g^p$. So now when receiving encrypted ballots from a corrupt computer, we decrypt $f(v_i) = w_i y_{1i}^{-p}$ when generating receipt codes. This decryption is remembered by M and when the corresponding encrypted ballot is received by the decryption service, it uses the remembered decrypted ballot from R instead of decrypting. By the properties of $\mathcal{F}_{pok}$, the computations when decrypting are still correct, so this game is indistinguishable from the previous one. Note that now the only use of the decryption keys $\{a_{1i}\}$ is when generating w_i.

Game 9 We will need to remove the use of the decryption keys $\{a_{1i}\}$, so we now change back the computations of x and w_i that simulated honest computers do so that we in the next game can randomize w_i. So for simulated honest computers we compute as follows:

$$\begin{aligned} x &= g^t \\ w_i &= y_{1i}^t f(v_i) \\ \hat{w}_i &\overset{r}{\leftarrow} G \end{aligned}$$

The end result of the computations are just as in the previous game, hence this game is indistinguishable from the previous one.

Claim. The decryption keys $\{a_{1i}\}$ are no longer used in this game.

Proof. The same proof as in [5] holds in this case also. □

Game 10 In this last game, when simulated honest computers computes ciphertexts, we randomize the ciphertexts so that they contain no information. So we compute as follows:

$$\begin{aligned} x &\overset{r}{\leftarrow} G \\ w_i &\overset{r}{\leftarrow} G \\ \hat{w}_i &\overset{r}{\leftarrow} G \end{aligned}$$

Claim. A distinguisher between Game 9 and 10 will give us an adversary for the DDH problem.

Proof. We will prove that a distinguisher between the two games gives an adversary for the subgroup membership problem $H_0^{k_{max}+1} \overset{?}{\leftrightarrow} G^{k_{max}+1}$, where now $H_0^{k_{max}+1} = (g, y_{11}, \ldots, y_{1k_{max}})$. By Theorem 2.8 this gives an advantage on the DDH problem. So take a tuple $u = (u_0, \ldots, u_{k_{max}})$, where $u \in H_0^{k_{max}+1}$ or $u \in G^{k_{max}+1}$. Let $r, t \overset{r}{\leftarrow} \mathbb{Z}_{|G|}$, and compute as follows:

$$\begin{aligned} x &= g^t u_0^r \\ w_i &= y_{1i}^t u_i^r f(v_i), \forall i \in \{1, \ldots, k_{max}\} \end{aligned}$$

If $u \in H_0^{k_{max}+1}$ the computations will be as in Game 9, and if $u \in G^{k_{max}+1}$ the computations are as in Game 10. So a distinguisher will result in an adversary for the problem $H_0^{k_{max}+1} \overset{?}{\leftrightarrow} G^{k_{max}+1}$. □

Analysis We are now in the same situation as in the original protocol and the analysis there is valid in this case also.

7.2 The Receipt Generator

Now we make a security analysis with the receipt generator corrupted. We want to prove that the same condition as in the original protocol [5] holds here, the condition is as follows:

- The corrupt receipt generator learns nothing about the submitted ballots, except what the receipt codes tell it.

Game 1 In this game we want to make one player that has all the information the honest players have, and that can simulate all the players. So in this game

M plays the role of every honest player. Note that M has the private keys $\{a_{1i}\}$ and $\{a_{3i}\}$. Clearly this is indistinguishable from the real protocol.

Game 2 Now we want to get independent of the encryptions, that is we no longer want to be dependent on the encryptions being valid for D to output the correct ballot. So instead of decrypting the encrypted ballots, we use the cleartext ballots. Since M plays the role of both D and every P, it knows the cleartext ballots corresponding to every encryption. Clearly this game is indistinguishable from the previous one. Note that now $\{a_{1i}\}$ are never used.

Game 3 The next step in the process is to get independent of the proofs. Since all computers and the ballot box now are honest, $\mathcal{F}_{pok}$ never uses the witnesses when verifying the proofs for R. Therefore we now change the protocols so that B and all the P's give random witnesses to $\mathcal{F}_{pok}$. Since the witnesses are never used, this game is indistinguishable from the previous one.

Game 4 We change the computation of the $\hat{w}_i$'s so that we in the next game can randomize the w_i's. So we compute as follows:

$$\begin{array}{ll} x = g^t & w_i = y_{1i}^t f(v_i) \\ & \hat{w}_i = x^{a_{3i}} f(v_i) \\ \check{x} = x^s & \check{w}_i = \hat{w}_i^s \end{array}$$

The only change we have made is that now $\hat{w}_i$ is computed as $\hat{w}_i = x^{a_{3i}} f(v_i)$ instead of $\hat{w}_i = y_{3i}^t f(v_i)$. But as this give the same result, this game is indistinguishable from the previous one.

Game 5 Now we randomize the w_i's, so that they contain no information. So now when encrypting ballots we compute as follows:

$$\begin{array}{ll} x = g^t & w_i \overset{r}{\leftarrow} G \\ & \hat{w}_i = x^{a_{3i}} f(v_i) \\ \check{x} = x^s & \check{w}_i = \hat{w}_i^s \end{array}$$

Claim. A distinguisher between this and the previous game will give an adversary for the DDH problem.

Proof. We will prove that a distinguisher between the two games gives an adversary for the subgroup membership problem $H_0^{k_{max}} \overset{?}{\leftrightarrow} G^{k_{max}}$, where now $H_0^{k_{max}} = (y_{11}, \ldots, y_{1k_{max}})$. By Theorem 2.8 this will give an adversary for the

DDH problem. Let $u = (u_1, \dots, u_{k_{max}})$ be a tuple such that $u \in H_0^{k_{max}}$ or $u \in G^{k_{max}}$. Let $r, t \xleftarrow{r} \mathbb{Z}_{|G|}$, and compute as follows:

$$w_i = y_{1i}^t u_i^r f(v_i), \forall i \in \{1, \dots, k_{max}\}$$

Now if $u \in H_0^{k_{max}}$ the computations will be as in Game 3, and if $u \in G^{k_{max}}$ the computations are as in Game 4. So a distinguisher will result in an adversary for the problem $H_0^{k_{max}} \overset{?}{\leftrightarrow} G^{k_{max}}$. □

Game 6 Now we want the computations done by B to be done at the same time as generating the encrypted ballots, this way we can later alter the order in which we compute the different ciphertexts. It is important to note that the ballot box does not send the messages before the appropriate message has been received from P. Clearly this game is indistinguishable from the previous one.

Game 7 Now we wish to alter the way the $\breve{w}_i$'s and the $\hat{w}_i$'s are computed so that we at a later time can randomize the per voter functions $v \mapsto f(v)^s$. We now compute as follows:

$$\begin{array}{ll} x = g^t & w_i \xleftarrow{r} G \\ & \hat{w}_i \xleftarrow{r} G \\ \breve{x} = x^s & \breve{w}_i = \breve{x}^{a_{3i}} f(v_i)^s \end{array}$$

Since R only sees a commitment to the $\hat{w}_i$'s, but not the $\hat{w}_i$'s themselves and the $\breve{w}_i$'s end up as the same as before this game is indistinguishable from the previous one.

Game 8 In this last game we replace the per voter function $v \mapsto f(v)^s$ with random functions ϕ from $\mathcal{O}$ to G. So now we compute as follows:

$$\begin{array}{ll} x = g^t & w_i \xleftarrow{r} G \\ & \hat{w}_i \xleftarrow{r} G \\ \breve{x} = x^{t'} & \breve{w}_i = \breve{x}^{a_{3i}} \phi(v_i) \end{array}$$

Note that $\phi(\cdot)$ is a random function, so $\phi(v_i)$ is a random element in G.

Claim. A distinguisher for this and the previous game will give an adversary for the DDH-problem.

Proof. We will show that a distinguisher for this and the previous game gives an advantage on the subgroup membership problem $H_0^{|\mathcal{O}|} \overset{?}{\leftrightarrow} G^{|\mathcal{O}|}$, where $H_0^{|\mathcal{O}|} = \langle (l_1, \dots, l_{|\mathcal{O}|}) \rangle$. By Theorem 2.8 this will give an advantage on DDH. So our adversary gets as input $(l_1, \dots, l_{|\mathcal{O}|})$ and $(u_1, \dots, u_{|\mathcal{O}|})$. Further define $f(v_i) = l_i$, $i \in \{1, \dots, |\mathcal{O}|\}$. We then compute as follows for every voter:

$$\begin{aligned} g &= \prod_{j=1}^{|\mathcal{O}|} l_j^{r_j} & \gamma &= \prod_{j=1}^{|\mathcal{O}|} u_j^{r_j} \\ x &= \prod_{j=1}^{|\mathcal{O}|} l_j^{t_j} & w_i &\overset{r}{\leftarrow} G \\ & & \hat{w}_i &\overset{r}{\leftarrow} G \\ \check{x} &= \prod_{j=1}^{|\mathcal{O}|} u_j^{t_j} & \breve{w}_i &= \check{x}^{a_{3i}} u_i \end{aligned}$$

If there exists an s such that $u_j = l_j^s$, $\forall j \in \{1, \dots, |\mathcal{O}|\}$, then we have the same probability distributions as the previous game. If not, then we still have the same relationship as in this game between x, $\check{x}$ and $\breve{w}_i$, further u_i's are random elements of G and hence the function we have simulated used as encoding function is a random function (It must satisfy $(\phi(v_i) = u_i \ \forall i \in \{1, \dots, \mathcal{O}\}$, and hence could be any function). So we see that we have the same probability distributions as in this game. Hence we see that a distinguisher for this game and the previous one will give a distinguisher for the DDH-problem. □

Analysis Since the encrypted ballots received by R in the last game contain no information about the ballots and R must see the receipt codes, it learns no unavoidable information about the submitted ballots.

7.3 Decryption Service

Since the information D gets is not changed compared to what D gets in the original protocol [5], we conclude that the analysis in the original protocol is still valid. Therefore we do not comment on it here.

7.4 Auditor

Will prove the same as Gjøsteen [5] did in the original protocol, that is:

- The submitted ballots remain confidential.

Game 1 In this game we want to do as we always have done so far, namely simulate all the players by one machine M. M knows all decryption keys, especially M knows $\{a_{1i}\}$ and $\{a_{3i}\}$.

Game 2 We see that $\mathcal{F}_{pok}$ never uses the witnesses received from the players, since it knows the players are honest, hence we do not need to provide the real witnesses. So in this game we provide $\mathcal{F}_{pok}$ with random witnesses. This game is clearly indistinguishable from the previous one.

Game 3 Since M knows the cleartext of each encrypted ballot, we no longer decrypt in the decryption service or the receipt generator. Instead we use the remembered cleartext ballots. These changes are not observable for the auditor since the end result is the same, hence the game is indistinguishable from the previous one. Note that we no longer use the contents of the encrypted ballots.

Game 4 Now we encrypt random group elements instead of the cleartext ballots when creating x, the w_i's and the $\hat{w}_i$'s. Note that $\hat{w}_i$ and w_i is created with the same random group element. Since the auditor never sees the $\hat{w}_i$'s, only commitments to them, it cannot distinguish between generating them with encryptions of real ballots or random group elements. We also randomize w_i with the same random group element as $\hat{w}_i$. A straight-forward reduction will show that if we can distinguish between generating w_i with a real ballot or a random group element, we get an advantage on the problem $H_0^{k_{max}+1} \overset{?}{\longleftrightarrow} G^{k_{max}+1}$, with $H_0^{k_{max}+1} = \langle (g, y_{11}, \dots, y_{1k_{max}}) \rangle$. So this game is indistinguishable to the previous one.

Game 5 In this last game we generate new random encryption in each round of the shuffle proof instead of rerandomization. As in the previous game, a straight-forward reduction to the problem $H_0^{k_{max}+1} \overset{?}{\longleftrightarrow} G^{k_{max}+1}$, with $H_0^{k_{max}+1} = \langle (g, y_{11}, \dots, y_{1k_{max}}) \rangle$ will show that this game is indistinguishable to the previous one.

Analysis We are now in the same position as in the original protocol, and hence the analysis done there will hold in our case also.

CHAPTER 8

CONCLUSION

We have now proposed two changes in the protocol, and proved that it will not reduce the security. We had, though, a perfectly good protocol, and it is no point in changing it if we do not make any improvements to it. We have claimed that our first change would improve the performance of the protocol, and it is now time to find out exactly how much performance is improved because of this first change.

We will also look at what the second improvement has done to the number of expected exponentiations, even though the most important goal of that change was to remove $\{a_{2i}\}$. Throughout we assume that computing the exponentiation of a element g with a number r takes one time unit, computation of the sum of two elements, product of two elements and computation of a hash function is assumed to use negligible time.

First we analyse the cost of the different functionalities in the old system (without the changes) and sum up, then we analyse the cost of the functionalities in the new system (with the first change). At last we analyse the cost of the functionalities with the second change implemented, before we comment on the differences between them.

Cost, old system

Player	**Cost**
Key generation	$2N + 3$
The voter's computer	$3k_{max}$
Ballot box and receipt generator	$N(23k_{max})$
Counting	$N(k_{max} + 15)$

Cost, after first change

Player	**Cost**
Key generation	$2N + 3k_{max}$
The voter's computer	$k_{max} + 2$
Ballot box and receipt generator	$N(12k_{max} + 9)$
Counting	$17N$

Cost, after second change

Player	**Cost**
Key generation	$2N + 2k_{max}$
The voter's computer	$7k_{max} + 3$
Ballot box and receipt generator	$N(23k_{max} + 10)$
Counting	$N(4k_{max} + 17)$

Note that N is the number of voters and k_{max} is the maximum number of choices a voter can make. Further note that the number of exponentiations we have said the ballot box and the receipt generator uses is the number they use in total on receiving ballots (not including counting).

Firstly we see that even though we have counted them, we are not worried about that fact that the number of exponentiations the key generation functionality must do has increased. These computations are done before the election and are therefore not noticed by the voters. One could argue in the same way that the counting is done after the election, but as people want to know the election result as fast as possible, it is meaningful to look at the number of exponentiations one must do during counting.

One sees that the number of exponentiations we must do to count has changed. After the first improvement we see that the number of exponentiations done to count the ballots has gone down heavily. The reason for this is that the proof of knowledge to prove P has computed correctly done after the first improvement is much simpler than in the original protocol. After the second improvement we see that the number of exponentiations has increased, the number of exponentiations needed is almost 4 times as many as in the original protocol. This comes from the fact that it after the second change is more complicated to check if a ciphertext is correctly computed. Since both B and R checks the proofs of knowledge, one could consider if the auditor needs to check the proofs. Both B and R must be corrupt if a proof of knowledge is invalid, and if both B and R are corrupt nothing can be guaranteed.

Now we look at the number of exponentiations needed to submit a ballot. We see that with the first change (but not the second change) implemented the voter's computer must do about one third as many exponentiations as in the original protocol. Also the ballot box and the receipt generator uses about half as many exponentiations when processing a ballot with the first change implemented compared to the original protocol.

So we see that we have achieved a pretty good speed up when it comes to submitting votes. Especially the improvement in the number of exponentiations the ballot box and the receipt generator must do is worth noticing. In practice it will be defined a limit for the amount of time it should take from the voter's computer sends the encrypted ballot to the ballot box, until we should receive a receipt telling the voter's computer that the ballot is accepted or not accepted. Therefore if we use half as many exponentiations when submitting a ballot, we use about half as much computing power and hence we will need to purchase half as much hardware.

When we look at how the last improvement effects the number of exponentiations, we see that the number of exponentiations the ballot box and the receipt generator must do when receiving a ballot is about as many as in the original protocol. Further, we see that the number of exponentiations the voter's computer must do when submitting a ballot is in fact more than twice as many as in the original protocol and 7 times as many as after the first improvement. The reason for this is that we had to introduce commitments to make the protocol secure. But the goal of this improvement was not to improve the performance, but rather to increase security by removing the connection between the different decryption keys. And this we have managed.

Still it is interesting to look at how to overcome this increased number of exponentiations. There are many possible commitment schemes, so one thing one could look at is if one could find a commitment scheme that uses fewer exponentiations without losing any security measures. We have not had the time to look into this, but it is something that could be interesting to look at, at a later stage. If one could find such a scheme one might be able to reduce the number of exponentiations significantly as now the voter's computer uses $4k_{max}$ exponentiations per voter, the ballot box $2k_{max}$ exponentiations per voter and the receipt generator $4k_{max}$ exponentiations per voter on just generating commitments with this second change implemented. It would though be important that this commitment scheme is as secure as the one used now.

BIBLIOGRAPHY

[1] Dan Boneh. The decision diffie-hellman problem. *Algorithmic Number Theory*, pages 48–63, 1998.

[2] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. Cryptology ePrint Archive, Report 2001/085, 2001. `http://eprint.iacr.org/`.

[3] Ivan Damgard. On σ-protocols. *Lecture on Cryptologic Protocol Theory*, 2010.

[4] Ivan Damgard and Jesper Buus Nielsen. Commitment schemes and zero-knowledge protocols (2008). 2008.

[5] Kristian Gjøsteen. Analysis of an internet voting protocol. Cryptology ePrint Archive, Report 2010/380, 2010.

[6] Statistisk Sentralbyrå. Folkemengd, etter alder og fylke. absolutte tal. 1. januar 2011. `http://www.ssb.no/folkemengde/arkiv/tab-2011-02-24-01.html`, 2011.