NTNU
Norwegian University of
Science and Technology

Improving the Norwegian Internet Voting Protocol

Marianne Wiik Øberg

Master of Science in Mathematics
Submission date: June 2011
Supervisor: Kristian Gjøsteen, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

We have in this thesis looked at possible improvements with respect to security for the Norwegian Internet voting protocol analyzed by Gjøsteen in [Gjø10].

We have made a new protocol with independent secret keys, where all the encryptions of the votes are done by the voter's computer. We have also made two *Special-Honest-Verifier-Zero-Knowledge Arguments of Knowledge* for proving permutation *and* decryption of ElGamal ciphertexts, useful for the decryption service.

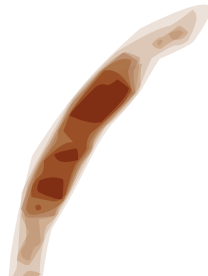
Acknowledgments

Thanks to my fantastic supervisor, Kristian Gjøsteen. You have always been available and patient, teaching me cryptography from scratch (over and over... and over again). There has always been an open door, a red pen and a smile. It has been a great pleasure working with you. Also thanks to the rest of the Institute for Mathematics for always helping and caring.

Thanks to Tea Toft, for holding hands and entering the abstract world together. Lucky me, our friendship $\in \mathbb{R}$. Thank you for teaching me how to never start the thinking process by trusting my own ideas. Thanks to Anders S. Lund for filling the "master-breaks" with academic discussions, at least that is what we like to tell our self. And for always knowing when it is the time for teaching me, and when it is the time for "just-do-it-for-me" when I was searching for your computer knowledge. The cute guys in the "driftsgutta"-office deserve thanks for holding out with me. This thesis would have been written by hand without you. Terje Buer did a great job reading through my thesis the very last night. Thank you!

So many friends have filled the time as student with so much more than studying. You have taught me how to always find a reason to celebrate ourself′. Thanks for all the great days/nights/mornings/evenings/coffee
breaks wherever we have been. Trondheim has given me nerds for life, that is priceless friendships! Also thanks to my anti-nerd friends in Oslo for keeping me quite normal after all!

Finally, thanks to my wonderful family for coffee on bed, and for always supporting and believing in me. I love the way we share everything (even though I have a feeling that you have never understood what I have been working with, you get the chance to try a bit harder with this paper). We keep going together!



Contents

Chapter 1

Introduction

Internet voting. The Norwegian government is planning a trial of Internet voting during the local government election September 2011. To build trust to the system, an important principle for the government is transparency. Most important documents including technical details are therefore made public. Gjøsteen [Gjø10] has described and analyzed the cryptographic protocol that will be used for the trial.

The protocol is designed by the Spanish electronic voting company Scytl. The voter submits a ballot by picking vote options on his computer. The protocol consists of several players encrypting and decrypting the ballot before the final result is made public. In a security analysis of a cryptographic protocol, we look at how the protocol acts when it is been attacked. An attacker is an non-honest player taking control over parts of the protocol.

According to [Gjø10], there are two main security problems in the Norwegian Internet voting system, compromised computers and coercion. As a defense against coercion, a voter will be allowed to submit multiple ballots where the last submitted ballot always will count. If a paper vote is submitted, this will be counted instead of any electronic vote.

Overview of the paper. We will in this thesis look at how it is possible to improve parts of the existing protocol by making it more secure.

Chapter 2 is a presentation of the theoretical background we need for the improvements.

The first improvement is presented in Chapter 3. It is a new vote submission where we make the secret keys independent, and where we move all the encryptions of the votes to the voter's computer. The new key generation is described, with a security analysis of different cases of attacks.

Chapter 4 contains a new proof for the decryption service. We have made two *Special-Honest-Verifier-Zero-Knowledge Arguments of knowledge* for proving permutation *and* decryption of ElGamal ciphertexts. The two suggestions are compared with respect to the performance in the end of the chapter.

We assume readers have basic knowledge in mathematics and cryptography.

Chapter 2

Theoretical Background

We will in this chapter present the theoretical background used in the improvements in Chap. 3 and Chap. 4. We present notation, definitions, theorems and a new subprotocol.

2.1 Notation

Picking a random element r from a group $\mathbb{Z}_q$ will be denoted by $r \leftarrow \mathbb{Z}_q$. For a probabilistic algorithm A, we write $y \leftarrow A(x)$ for A making the assignment $y = A(x; r)$ on input x, where r is picked random. $\Pr[A; B]$ denotes the probability for B to happen, given the probability space A.

2.2 Game Hopping

The basic ideas and definitions of *game hopping* and *security proofs* are from [Sho04], and also described by Gjøsteen in [Gjø].

Security is the fundamental goal of every cryptographic protocol. A well-founded confidence in the security of the protocol is essential for the trust in the voting system. But to show that an attacker can not break a system can often be difficult because we must assume the attacker has greater cryptanalytic resources than we have.

A *security proof* is a way to deal with this by isolating core components of the protocol and showing that an attack on the cryptosystem must lead to an attack on some of the components. Given such security proof, the confidence in the security of all the core components will be transferred to the cryptosystem. The advantage is that the core components are often easier to analyze and our

resources will be used more efficiently. But for even simple cryptosystems, the security proof can be complex and difficult to understand, and therefore difficult to verify the correctness of.

A proof technique to simplify the complexity in a system and make the proofs easier to understand and verify, is called *Game hopping*. The technique will be presented after some basic definitions.

Definition 2.2.1. A *Probability distribution* on a finite set S is a function $\mu : S \to \mathbb{R}$ such that (i) for all $s \in S$, $0 \leq \mu(s) \leq 1$, and (ii) $\sum_{s \in S} \mu(s) = 1$. A *probability space* X is a pair (S, μ) where S is a finite set and μ the probability distribution on S

A central part in Game hopping is *distinguishing probability spaces*. Given two spaces X and Y over the same set S, we have a distinguisher between the spaces if we can tell whether an element $s \in S$ is sampled from X or Y.

Let A be an algorithm with (X, Y, s) as input. We define the *Success probability* of the algorithm, $Succ(A)$, as a measure on "how good" the algorithm is to distinguish the probability spaces. Since the chance with a random guess is $\frac{1}{2}$, we define the *advantage* for the algorithm to be

$$Adv(A) = |Succ(A) - \frac{1}{2}|$$

We call X and Y indistinguishable if $Adv(A)$ is negligible.

Definition 2.2.2. A *Game* G is a collection of interactive, probabilistic machines cooperating to produce 0 or 1 as output. A Game essentially describes a probability space over $\{0, 1\}$, and we denote this space by G.

In *Game hopping*, we make a sequence of indistinguishable games. Indistinguishable games mean that the probability spaces the games describe are indistinguishable. We start with a game with the original attack with respect to a given adversary, and want to end up with a game which is easier to analyze.

2.3 Encryption

We want to use ElGamal encryption in our protocol. That is, the same encryption as we use in the existing protocol in [Gjø10]. Our group G will be a cyclic group of prime order q, with a_1 as a secret key and the public keys (g, y_1) such that $y = g^{a_1}$. The encryption of a message m will be

$$\xi : M \times R \to G \times G$$
$$\xi(m, r) = (g^r, y^r \cdot m)$$

where r is a randomizer sampled uniformly from R. We can show that the encryption is homomorphic when $(M_{pk},\cdot)$, $(R_{pk},+)$, $(G,\cdot)$ are abelian groups :

$$\begin{aligned}\xi(m_o, r_o) \cdot \xi(m_1, r_1) &= (g^{r_0}, y^{r_0} m_0)(g^{r_1}, y^{r_1} m_1) \\ &= (g^{r_0+r_1}, y^{r_0+r_1} m_0 m_1) = \xi(m_0 m_1, r_0 + r_1)\end{aligned}$$

The secret key a_1 will be used for decryption of the message.

2.4 Commitments

Commitments are a central part of many cryptographic protocols. Damgard and Nielsen describe in [DN08] a player committing to a value or message, by hiding it in a commitment, before passing the commitment to another player without revealing the content of it.

They define a *commitment scheme* as a probabilistic polynomial generator that outputs a public key ck from a keyspace K. For every ck, there will be a function that outputs the commitment:

$$com_{ck} : M \times R \to C$$

with a message $m \in M$ and a random parameter $r \in R$ as input. To open the commitment, the *opening* (m, r) will be revealed and the second player checks that $c = com_{ck}(m, r)$.

The commitment scheme is said to have the *binding property* if it is hard for the first player to change the value of the commitment after it has been sent. The value it has been committed to should be the only value that can be accepted as an opening. The commitment scheme has the *hiding property* if it is hard for the second player to gain any knowledge of the value before the first player gives his approval and reveals the key.

We can divide each of the properties of hiding and binding into *unconditional* and *computational*. Unconditional means that it is theoretical impossible to break the property, while computational means it should not be possible to break it with polynomial computing power. Intuitively, a commitment scheme both *unconditional hiding and binding* will give the best protection, but unfortunately, this is impossible as described in [DN08].

In our situation, the commitment schemes are used by a prover P who wants to convince a verifier V in a limited time. A *computational hiding and binding* commitment scheme should therefore be enough to preserve our security. However, we always want to accomplish the best security possible. As we will see several times later, first in Sect. 4.1.1, we need an argument that the verifier does not see the contents of the commitments. For this reason, we choose a commitment scheme with the unconditional hiding property that should satisfy the following definition:

Definition 2.4.1. [DN08] We have a commitment scheme with:
computational binding if for any probabilistic polynomial time algorithm P^* which takes as input a public key produced by a generator on input 1^l. Let $\epsilon(l)$ be the probability with which the algorithm outputs a commitment and two valid openings revealing distinct values. That is, it outputs $C, m, r, m^{'}, r^{'}$ such that $m \neq m^{'}$ and $com_{ck}(m, r) = C = com_{ck}(m^{'}, r^{'})$. Then $\epsilon(l)$ is negligible in l.
Unconditional hiding if for even infinitely powerful V and for correctly generated ck and random independent r, s, then $com_{ck}(m, r)$ and $com_{ck}(m, s)$ are statistically indistinguishable [1]. Furthermore, an honest P should accept an incorrectly generated ck with negligible probability.

Two extra properties are important in the choice of commitment scheme for our protocol, *Homomorphic Property* and *Root-extraction Property.*

Definition 2.4.2. [Gro10] We have a *homomorphic commitment scheme* with message space $(M_{ck}, +)$, randomizer space $(R_{ck}, +)$ and commitment space $(C_{ck}, \cdot)$ all abelian groups, if for all ck and for all $(m_0, r_0), (m_1, r_1) \in M_{ck} \times R_{ck}$ we have:

$$com_{ck}(m_o + m_1; r_0 + r_1) = com_{ck}(m_0; r_0) \cdot com_{ck}(m_1; r_1)$$

Definition 2.4.3. [Gro10] A commitment scheme has the *root-extraction property* if there exist a polynomial time algorithm that given a message M, randomizer R, and a non-trivial $e \in \mathbb{Z}_q$ such that

$$c^e = com_{ck}(M; R) \text{ , then it outputs } (m, r) \text{ as the opening of } c$$

We further require that the algorithm runs in polynomial time.

Pedersen commitment scheme

A commitment with the required properties and that will be used in our protocol is a *Pedersen commitment scheme* found in [Ped92]. The key generator KG generates random (g, h) and let

$$com_{ck}(m, r) = (g^m \cdot h^r)$$

We will show that this satisfies the properties:

The commitment is clearly *homomorphic* and we begin to show the *root-extraction property.* Assume we have an opening (M, R) of c^e:

$$c^e = (g^M \cdot h^R) = \underbrace{c \cdot c \ldots \cdot c}_{e} = \underbrace{(g^m \cdot h^r) \cdots (g^m \cdot h^r)}_{e} = (g^{m+\cdots+m} \cdot h^{r+\cdots+r})$$

where $m = M/e, r = R/e$ and (m, r) is the opening of c

[1] the statistical distance between the two are negligible

It is a *unconditionally hiding* commitment scheme. The random r makes it impossible to tell what m is, and therefore to distinguish two different messages. Assume we want to find a second opening (m', r') of a commitment c to break the *binding property*. To find the randomizer to the new message m', we get $h^{r'} = c \cdot g^{m'^{-1}}$, and we are left with the discrete log problem which should be impossible with only computational computing power. It therefore satisfies the *computational binding property*.

To save exponentiations, we can use *multi commitments* [Gro04]. The key generator will generate random $(g_1, g_2 \ldots, g_n, r)$ and let

$$c = com_{ck}(m_1, m_2 \ldots m_n, r) = (g_1^{m_1} \cdots g_n^{m_n} \cdot h^r)$$

In this way, we use $n + 1$ exponentiations instead of $2n$ for committing to n messages.

2.5 Zero Knowledge Argument

A *Zero-Knowledge Argument* is an argument where we want to convince a player without leaking any information out of the conversation. We will start to define an *Interactive Argument System* with two players, a prover P who wants to convince a verifier V. We assume both have polynomial time computing power.

The prover and verifier get a common input x, and the verifier will after an interactive conversation, output rejected or accepted to the current x. Given a binary language L, we say that (P, V) accepts if V is convinced that $x \in L$, and (P, V) rejects if V is convinced that $x \notin L$ [DN08]. We call a conversation between P and V for a *transcript* (tr). A cheating player will be denoted by P^* and V^*.

Definition 2.5.1. [DN08] The pair (P, V) is an *Interactive Argument* for L if it satisfies the following conditions:
Completeness: If $x \in L$, then the probability that (P, V) rejects x is negligible in the length of x.
Soundness: If $x \notin L$, then for any prover P^*, the probability that (P^*, V) accepts x is negligible in the length of x.

A variant of this is the *Argument of knowledge*. The prover will instead of arguing that $x \in L$, argue that it knows certain piece of information. We call this information for a *witness* w. Let R be a relation, typically a subset of $\{0,1\}^* \times \{0,1\}^*$. The input x can be a computational problem, where w is the solution of the problem. We say that $(x, w) \in R$ if w is the solution of x [DN08].

Definition 2.5.2. [Dam10] Let $\kappa()$ be a function from bit strings to the interval $[0, 1]$. The protocol (P, V) is said to be the *Argument of knowledge* for the relation R with knowledge error κ, if the following are satisfied:
Knowledge completeness:On common input x, if the honest prover gets a private input w such that $(x, w) \in R$, then the verifier always accepts.
Knowledge soundness: There exist a probabilistic algorithm M called the knowledge extractor. This M gets input x and rewindable black-box access to the prover and attempts to compute w such that $(x, w) \in R$. We require that the following holds: For any non-honest prover P^*, let $\epsilon(x)$ be the probability that V accepts on input x. There exist a constant c such that whenever $\epsilon(x) > \kappa(x)$, M will output a correct w in expected time at most

$$\frac{|x|^c}{\epsilon(x) - \kappa(x)}$$

where access to P^* counts as one step only.

Zero-Knowledge(ZK) is an extra property to the Interactive Argument or Argument of knowledge. In addition to completeness and soundness, we do not want V to be able to extract any extra information out of the conversation that he did not know before. We can show this by making a simulator for the prover without access to the witness, and showing that the arguments made by the real prover and the simulator are indistinguishable.

Definition 2.5.3. [DN08] An interactive argument system (P, V) for language L is *Zero-Knowledge Argument* if for every probabilistic polynomial time verifier V^* there is a simulator M_{V^*} running in expected probabilistic polynomial time, such that the output of M_{V^*} has the same probability distribution as the output from (P, V) on input $x \in L$ and arbitrary δ (as input to V^* only).

Similarly, we can define *Zero-Knowledge Argument of knowledge*. We will use several versions of Zero-Knowledge-arguments in this paper. *Honest-Verifier-Zero-Knowledge* (HVZK) means that the argument is ZK as long as the verifier is honest and follows the protocol. A stronger version is *Special-Honest-Verifier-Zero-Knowledge* (SHVZK).

Definition 2.5.4. [Gro10] An argument is called *Special-Honest-Verifier-Zero-Knowledge* for a relation R if there exists a simulator S such that for all non-uniform adversaries A we have

$$\begin{aligned} &Pr\Big[\sigma \leftarrow K(); (x, w, \rho) \leftarrow A(\sigma); tr \leftarrow \langle P^*(\sigma, x, w), V(\sigma, x; \rho)\rangle : \\ &\quad (\sigma, x, w) \in R \text{ and } \mathrm{A(tr)} = 1\Big] \\ &\approx Pr\Big[\sigma \leftarrow K(); (x, w, \rho) \leftarrow A(\sigma); tr \leftarrow S(\sigma, x; \rho) : (\sigma, x, w) \in R \text{ and } \mathrm{A(tr)} = 1\Big] \end{aligned}$$

The special case differ from the normal with the fact that it is HVZK on a given challenge ρ.

These cases are interesting even they demand the verifier to be honest. [Gro10] remarks that there are efficient techniques to transfer Honest-Verifier-Zero-Knowledge Arguments into Non-Interactive Zero-Knowledge Arguments, and [Gro04] describes such technique by all participants having access to a *random oracle*.

The *random oracle* acts like a random function R: $\{0,1\}^l \rightarrow \{0,1\}^t$. Since R is totally random, $R(a)$ will be uniform and random. $R(a)$ will also be independent of a and gives no information about $R(b)$ unless $a = b$. R will be fixed, such that the same $R(a)$ always will be returned on input a [Dam10].

With such oracle, we can execute a HVZK-protocol without the interactive conversation with V. The random oracle acts like an honest verifier, and we can replace the challenges V outputs on input a by $R(a)$. The prover will execute the protocol by generating the "conversation" with $R(a)$ as the challenge, and pass the whole conversation to V. The verifier will be able to accept or reject the argument by calling $R(a)$ from the same oracle. A SHVZK-argument with the random oracle behaving like an honest verifier, but without the ability to cheat, will according to [Gro04] yields as a Non-Interactive Zero-Knowledge–argument.

2.6 Witness-Extended Emulation

To complete a *Special-Honest-Verifier-Zero-Knowledge Argument of knowledge*, we need to show the *Knowledge soundness property* in Def. 2.5.2. The problem however, described in [DF02], is that the soundness property requires a 100 % chance for the emulator to extract the witness whenever a prover can convince the verifier. This will not always be the case, and they modified the definition to be a new knowledge soundness for a *Computationally convincing argument of knowledge*

Definition 2.6.1. [Gro04] A protocol (P, V) has the *Computational knowledge soundness* property for the relation R with knowledge error $\kappa(k)$ and failure probability $\nu(k)$ if there is a polynomial $p(k)$ and a probabilistic machine M such that for all deterministic polynomial time provers P and all probabilistic polynomial time adversaries A we have

$$Pr\Big[\sigma \leftarrow K(); (x,s) \leftarrow A(\sigma) : \epsilon_{P^*}(\sigma,x,s) > \kappa(k) \text{ and the expected running time of } M^{P^*}(\sigma,x) \text{ is larger than} \frac{p(k)}{\epsilon_{P^*} - \kappa(k)}\Big] < \nu(k)$$

where ϵ_{P^*} is the probability that a cheating prover P^* convinces V, and the running time of M is defined as ∞ if there is non-zero probability that it does not outputs a witness such that $(\sigma, x, w) \in R$.

Another property used in this paper is called *Witness-extended emulation.* That is, for all adversaries with probability ϵ to get an accepting argument, we want the emulator E to have the *same probability* to produce a similar argument *and* provide the witness.

Definition 2.6.2. [Gro10] An argument has *witness-extended emulation* if for all deterministic polynomial time P^* there exists an expected polynomial time emulator E such that for all non-uniform polynomial time adversaries A we have

$$Pr\Big[\sigma \leftarrow K(); (x,s) \leftarrow A(\sigma); tr \leftarrow \langle P^*(\sigma,x,s), V(\sigma,x)\rangle : A(tr) = 1\Big] \approx$$
$$Pr\big[\sigma \leftarrow K(); (x,s) \leftarrow A(\sigma); (tr,w) \leftarrow E^{\langle P^*(\sigma,x,s), V(\sigma,x)\rangle}(\sigma,x) :$$
$$A(tr) = 1 \text{ and if tr is accepting then} (\sigma,x,w) \in R\Big]$$

where s is the state of P^*, including the randomness. E has access to a transcript oracle $\langle P^*(\sigma,x,s), V(\sigma,x)\rangle$ that can be rewound to a particular round and run again and again with the verifier choosing fresh random coins.

We will define the emulator E to run $\langle P^*(\sigma,x,s), V(\sigma,x)\rangle$ over and over again until it finds the witness for every time V accepts the argument. With this in mind, we have the following theorem

Theorem 2.6.3. *[Gro04] An Interactive argument with Witness-extended emulation property is a Zero-Knowledge Argument of Computational knowledge with negligible knowledge error and negligible failure probability.*

We write out the sketch of proof from [Gro04]

Proof. Completeness follows from the completeness property of the argument. We want to show *computational knowledge soundness.* Assume we have an interactive argument with Witness-extended emulation property. Then we can assume from Def. 2.6.2 that there exists a negligible function $\delta(k)$ such that

$$Pr\Big[\sigma \leftarrow K() : (x,s) \leftarrow A(\sigma); tr \leftarrow \langle P^*(\sigma,x,s), V(\sigma,x)\rangle : A(tr) = 1\Big] -$$
$$Pr\Big[\sigma \leftarrow K() : (x,s) \leftarrow A(\sigma); (tr,w) \leftarrow E^{\langle P^*(\sigma,x,s), V(\sigma,x)\rangle}(\sigma,x) : A(tr) = 1$$
$$\text{and if it is accepting,then } (\sigma, x.w) \in R\Big] < \delta(k) \tag{2.1}$$

We are only interested in the probability to find the transcript tr when V *accepts*, otherwise E outputs $w = \perp$ and we do not need to find a witness either. Let

$$\epsilon_{p^*}(\sigma, x, s) = Pr[P^* \text{ convinces } V]$$
$$\alpha(\sigma, x, s) = Pr\big[\Sigma \leftarrow K(); (X, S) \leftarrow A(\Sigma) : \Sigma = \sigma, X = x, S = s\big]$$
$$\beta_E(\sigma, x, s) = Pr\big[w \leftarrow E^{\langle P^*(\sigma,x,s), V(\sigma,x)\rangle} : (\sigma, x, w) \in R\big]$$

and let d be the function $d(\sigma, x, s) = (\epsilon_{P^*} - \beta_E)(\sigma, x, s)$

We can then write out the probability of (2.1) to be

$$\begin{aligned}
&\sum_{(\sigma,x,s)} \alpha(\sigma, x, s) \cdot \epsilon_{P^*}(\sigma, x, s) \quad - \sum_{(\sigma,x,s)} \alpha(\sigma, x, s) \cdot \beta_E(\sigma, x, s) \\
&= \sum_{(\sigma,x,s)} \alpha \cdot (\epsilon_{P^*} - \beta_E)(\sigma, x, s) \\
&= \sum_{d(\sigma,x,s)>\kappa(k)} \alpha \cdot (\epsilon_{P^*} - \beta_E)(\sigma, x, s) \quad + \\
&\qquad \sum_{d(\sigma,x,s)\leq\kappa(k)} \alpha \cdot (\epsilon_{P^*} - \beta_E)(\sigma, x, s) < \delta(k) \qquad (2.2)
\end{aligned}$$

where $\kappa(k)$ is the *knowledge error*. We can argue that d is always positive, i.e $\epsilon_{P^*} \geq \beta_E$, in the way E is constructed. For E to find a witness such that $(\sigma, x, w) \in R$, it needs an accepting argument. That means that

$$\beta_E = \text{Pr[tr is accepting]} \cdot \text{Pr[E finds } w \text{ such that} (\sigma, x, w) \in R]$$

The first factor is ϵ_{P*} because E is running $\langle P^*(\sigma, x, s), V(\sigma, x)\rangle$ to get the argument. The second factor is a probability function $\in (0, 1)$ which implies $\beta_E \leq \epsilon_{P}*$, and d is positive. We can therefore conclude that both terms in (2.2) are positive and we have

$$\sum_{d(\sigma,x,s)>\kappa(k)} \alpha \cdot (\epsilon_{P^*} - \beta_E)(\sigma, x, s) < \delta(k) \qquad (2.3)$$
$$\sum_{d(\sigma,x,s)\leq\kappa(k)} \alpha \cdot (\epsilon_{P^*} - \beta_E)(\sigma, x, s) < \delta(k) \qquad (2.4)$$

Let $\nu(k)$ be the failure probability and $\nu(k) = \kappa(k) = \sqrt{\delta(k)}$. The witness-extractor M runs the emulator E^{P^*} until it gets a witness w such that $(\sigma, x, w) \in R$. To show computational knowledge soundness, we want to find the probability to get (σ, x, s) such that $\epsilon_{P^*} > \kappa(k)$ and M has an expected running time larger

than $\frac{p(k)}{\epsilon_{P^*}-\kappa(k)}$, and show that this probability is less than $\nu(k)$. To have M running more than $\frac{p(k)}{\epsilon_{P^*}-\kappa(k)}$ times means that the probability for E^{P^*} to find w such that $(\sigma, x, w) \in R$, $\beta_E(\sigma, x, s)$, is less than $\epsilon_{P^*} - \kappa(k)$. That is

$$\begin{aligned}\beta_E(\sigma, x, s) &< \epsilon_{P^*}(\sigma, x, s) - \kappa(k) \\ &\Rightarrow (\epsilon_{P^*} - \beta_E)(\sigma, x, s) > \kappa(k) \\ &\Rightarrow d(\sigma, x, s) > \kappa(k)\end{aligned}$$

This is exactly the elements in (2.3), and since $\kappa(k)$ is the lower bound of $(\epsilon_{P^*} - \beta_E)(\sigma, x, s)$ we have

$$\sum_{d(\sigma,x,s)>\kappa(k)} \alpha(\sigma, x, s)\kappa(k) < \delta(k)$$

which gives us the probability to get such triple to be

$$\sum_{d(\sigma,x,s)>\kappa(k)} \alpha(\sigma, x, s) = \frac{\delta(k)}{\kappa(k)} < \nu(k)$$

where $\nu(), \kappa()$ negligible by $\delta()$ negligible.

□

2.7 Encryption of Identity Subprotocol

We will in Sect. 4.1.1 need a *Special-Honest-Verifier-Zero-Knowledge* (SHVZK) *argument of knowledge* showing that

$$\xi_{id} = (x_0, w_0) = (g^r, y_1^r \cdot 1)$$

is an encryption of the identity where $y_1 = g^{a_1}$ and without knowing r. The idea is that if we have an encryption of the identity, then

$$\begin{aligned}\xi_{id} &= (x_0, w_0) = (g^r, y_1^r \cdot 1) \\ &\Rightarrow \log_{x_o} w_o = \log_g y_1 = a_1\end{aligned}$$

The subprotocol is shown in Fig.2.1 and we will show it is a SHVZK-argument of knowledge in the next sections.

PROVER		VERIFIER
	common input (g,y),(x_0, w_0)	
Private input: a_1 s.t $y_1 = g^{a_1}$ and $w_0 = x_0^{a_1}$		
$s \leftarrow \mathbb{Z}_q$		
$q_0 = x_0^s, q_1 = g^s$	$\xrightarrow{q_0, q_1}$	
		$e \leftarrow \mathbb{Z}_q$
	$\xleftarrow{e}$	
$z = a_1 e + s$	$\xrightarrow{z}$	Check that $q_0 w_0^e = x_0^z$ $q_1 y_1^e = g^z$

Figure 2.1: Subprotocol to show identity encryption

2.7.1 SHVZK

Completeness is straight forward to verify. Soundness follows from the Witness-extended emulation in the next subsection. We show it is *Special-Honest-Verifier-Zero-Knowledge* by making a simulator without access to the private input, and indistinguishable from the real prover. That is, on input e it outputs a transcript (q_0, q_1, e, z) with the same probability distribution as the transcription made by the the real prover. The differences between the simulator, hybrid and real prover will be described by colors in Fig. 2.2

Simulator	**Hybrid**	**Real Prover**
$z \leftarrow \mathbb{Z}_q$	$z \leftarrow \mathbb{Z}_q$	$s \leftarrow \mathbb{Z}_q$
$q_0 = x_0^z w_0^{-e}$	$s = z - a_1 e$	$z = a_1 e + s$
$q_1 = g^z y^{-e}$	$q_0 = x_0^z w_0^{-e}$	$q_0 = x_0^s$
	$q_1 = g^z y^{-e}$	$q_1 = g^s$

Figure 2.2: Differences between the Simulator and the Real Prover

The *simulator* and *hybrid* arguments differ by that s is computed in the hybrid argument. Since s is not used in the computation of q_0, q_1, the transcriptions will

be exactly the same. The *hybrid* and *Real prover* arguments differ by how z, s, q_0 and q_1 are computed. We can easily see that the computation of q_0 and q_1 gives the same result, and z has the same probability distribution since z is picked random in the hybrid argument and s is picked random in the real argument. From this, it follows that the transcripts made by the simulator and the real prover have the same probability distribution.

2.7.2 Witness-Extended Emulation

We want to create the emulator E, and show that E with the same probability that the prover P^* can convince V, will be able to extract the witness a_1 in expected polynomial time.

Let the emulator E run as the real conversation between $\langle P^*, V\rangle$. It gets out the transcription (q_0, q_1, e, z). If P^* fails, i.e V rejects, E outputs (tr, $\perp$). If V accepts, E runs $\langle P^*, V\rangle$ over and over again on the same q_0, q_1 until it gets another accepting argument $(q_0, q_1, e^{'}, z^{'})$. Thus we have $q_0 w_0^e = x_0^z$ and $q_0 w_0^{e^{'}} = x_0^{z^{'}}$. This gives us

$$\begin{aligned} w_0^{e-e^{'}} &= x_0^{(z-z^{'})} \\ w_0 &= x_0^{((z-z^{'})(e-e^{'})^{-1})} \end{aligned}$$

Want to show that $y = g^{((z-z^{'})(e-e^{'})^{-1})} = g^{a_1}$.
We have $q_1 y^e = g^z$ and $q_1 y^{e^{'}} = g^{z^{'}}$:

$$\begin{aligned} y^{e-e^{'}} &= g^{(z-z^{'})} \\ y &= g^{((z-z^{'})(e-e^{'})^{-1})} \end{aligned}$$

Thus, $a_1 = (z - z^{'})(e - e^{'})^{-1}$ and we have the witness.

It remains to argue that E uses expected polynomial time. Assume P^* has the probability ϵ to get an accepting argument. If he does and E needs to find the witness, we except $\frac{1}{\epsilon}$ runs to get the second argument. There will be one run and a non-accepting argument with a probability $(1 - \epsilon)$. In total, this gives us $(1 - \epsilon) \cdot 1 + (1 + \frac{1}{\epsilon}) \cdot \epsilon = 2$ expected runs.

Chapter 3

Improving the Vote Submission

We will in this chapter present a new and improved protocol for the Norwegian Internet voting system described in [Gjø10], with respect to security.

In the existing protocol, there are three connected secret keys spread out to three different players with one key each. The connection between the keys makes us vulnerable for two players cooperating, they can easily compute the third key. We will avoid this problem by making a new protocol with two independent secret keys.

The encryption of the votes are in the original protocol done by two different players. We will move the encryption to one of them, remove the secret key used by the other, and let the two remaining secret keys be independent of each other.

More technical, the protocol consists of several players. It is a set of voters V and the voter's computers P. We also have a set of infrastructure players. That is the ballot box B, the receipt generator R, the decryption service D and the auditor A, which monitors the entire process. An attacker is a non-honest player taking control over parts of the protocol. We call a part being controlled by an attacker for a corrupt player, and it will be denoted by the player's letter followed by *. The connection between the secret keys are $a_3 \equiv a_1 + a_2 \pmod{q}$.

Originally, the encryption of the votes was first done by the computer, followed by an re-encryption in B with the encryption key a_2 and a personal exponent s. We will move all the encryption to the computer, and make it without a_2. The receipt generator computes a receipt for each ballot with a_3, and sends it directly to the voter. He will control it against a *receipt code-set* received before the election starts. The decryption service decrypts the ballots, sent from the ballot box when the election is closed, with the decryption key a_1. With this encryption,

a corrupt B^* and R^* will no longer be able to compute a_1 and decrypt the votes.

We will begin the chapter with the new key generation functionality followed by the new vote submission. The following section will be a security analysis for different cases of attacks. And the last section discusses the improvements we have made.

3.1 Key Generation

We will in this section describe the new key generation KG. It is based on the key generation in [Gjø10], adapted to the changes made in our protocol. In the existing protocol, KG generates the per-voter exponent s and gives it directly to the ballot box through a secure channel.

In the improved protocol, the voter's computer needs the exponent s. For security reasons, the key generation closes *before* the election starts, and it will therefore not be able to send s directly to the computers *during* the election. The description of the key generation will therefore be followed by two suggestions for distribution of s to P.

Key generation Let $\mathcal{O}$ be the set of *options*, $\mathcal{C}$ the set of *receipt codes* and G a cyclic group of prime order q generated by g. KG generates random a_1, a_3 and makes (g, q, y_1, y_3) public, where $y_1 = g^{a_1}$ and $y_3 = g^{a_3}$. The decryption service gets a_1 and the receipt generator gets a_3. KG generates a public $f : \mathcal{O} \to G$ which encodes the vote options into group elements. In addition, it generates the per-voter-exponent s from $\mathbb{Z}_q$ and computes $\gamma = g^s$. The element γ is given to both B and R. We need a set of functions to generate the *receipt codes set*. That is d picked from a pseudo-random function family $F : G \to C$. The composition of f, the exponentiation map $x \longmapsto x^s$ and d gives a function $r : \mathcal{O} \to \mathcal{C}$ where $r(v) = d((f(v))^s)$. Before the election, the receipt code-set $\{(v; r(v)) \mid v \in \mathcal{O}\}$ is computed and given to V.

Intuitively, there are two ways to solve the distribution problem of s. We can use existing players and channels and pass s to P through B. The key generator will then use an encryption to hide s from B, where P with V will have the decryption key. The second way is to make a new player which will be open during the election. It will receive all the exponents from KG and keep them until a computer asks for it.

3.1.1 Distribution of the Exponent s Through Existing Players.

With a distribution of s through the ballot box, we will need to encrypt s. We will use a *public key encryption* with encryption key e_{pk} and a *smart card*, *SC*.

The smart card will be issued by the government, and we will assume it will be available for the improved protocol. We want the smart card, in addition to the signature and identification, containing a decryption key d_s such that $d_s(\xi_{e_{pk}}(s)) = s$. The generation and delivering of s *before* and *during* the election will be described in Fig. 3.1 and Fig. 3.2 and be like:

1. KG generates the per-voter exponent s, and does an encryption of s with a public encryption key e_{pk} given for every voter. The tuple $\left\{V, \gamma, \xi_{e_{pk}}(s)\right\}$ is given to the ballot box.

2. The contact between the voter's computer and ballot box will be establish by P using the smart card and passing the voters signature. B will respond on the signature by the corresponding $\xi_{e_{pk}}(s)$ from the list received from KG.

3. The computer uses the decryption key d_s to decrypt. It will now be in possession of s and ready to do the computation described in Sect. 3.2.

KG		B
For all voters V: $s \xleftarrow{r} \{0, ...q-1\}$		
Compute $\xi_{e_{pk}}(s)$	$\xrightarrow{\left\{V, \xi_{e_{pk}}(s)\right\}}$	Store $\left\{V, \xi_{e_{pk}}(s), \gamma\right\}$

Figure 3.1: Distribution of s from KG to P before the election

B	P		V/SC
Store $\left\{V, \xi_{e_{pk}}(s)\right\}$			
	$\xleftarrow{V_{id}}$	$\xleftarrow{V_{id}}$	v_{id}, d_s
	$\xrightarrow{\xi_{e_{pk}}(s)}$	$\xrightarrow{\xi_{e_{pk}}(s)}$	
		$\xleftarrow{(s)}$	
	s		$d_s(\xi_{e_{pk}}(s)) = s$

Figure 3.2: Distribution of s from KG to P during the election

The computer with access to the voter's smart card will be the only player in possession of s during the election. But during the election, we will have a problem to generate new smart cards if they get lost. A lost smart card should be

revoked to avoid it being abused, and a new smart card should therefore contain a *new* decryption key. However, the encryption of s is already given to B *before* the election starts, and a new encryption can not be generated because KG is closed.

In practice, with this distribution of s, a smart card lost during the election can not be replaced and a voter will in this case be forced to use paper vote.

3.1.2 Distribution of the Exponent s Through New Players

Our aim is to find a distribution of s in such way that a new smart card can replace a lost one. We need to introduce a new player which will be open during the election. We could let this extra player communicate directly to P, and in this way pass s to P without the encryption. However, for security reasons, we want as few players as possible be connected directly to the Internet, and an encryption of s sent through B is still to prefer. We will still use the smart card described in 3.1.1 for decryption.

To avoid an attacker taking control over the new extra player with the exponent s, we can divide this extra player into *multiple players*, where each player only has access to a piece of s. We will use a *Shamir treshold-scheme* to share s among the players.

Definition 3.1.1. [Sti06] Let t, w be positive integers, $t \leq w$. A *Shamir* (t, w)*-treshold scheme* is a method of sharing a key K among a set of w participants (denoted by P), in such a way that any t participants can compute the value of K, but no group of $t-1$ participants can do.

The Shamir (t, w)-treshold scheme is described in Fig. 3.3 from [Sti06] $A(x)$ will be of degree $t-1$ and on the form

$$A(x) = a_o + a_1 x + ...a_{t-1} x^{t-1}$$

t participants with $(x_i, A(x_i) = s_i)$ will have t equations, and will be able to determine all the coefficients of $A(x)$. We can then compute $K = A(0)$. The solution is unique by Lagrange interpolation formula described in [Sti06], which also gives us the formula to compute $A(x)$:

$$A(x) = \sum_{i=1}^{t} (s_i) \prod_{1 \leq k \leq t, k \neq i} \frac{x - x_i}{x_i - x_k} \pmod{q}$$

However, we do not need the whole polynomial since $K = A(0)$. Thus,

$$K = A(0) = \sum_{i=1}^{t} (s_i) \prod_{1 \leq k \leq t, k \neq i} \frac{-x_i}{x_i - x_k} \pmod{q}$$

Initialization Phase

1. The Dealer chooses w distinct, non-zero elements of $\mathbb{Z}_q$, denoted x_i, $1 \leq i \leq w$ (and $q \leq w+1$). For all i, the dealer gives the value x_i, all public, to the participants P_i

Share distribution

2. Suppose the dealer wants to share a key $K \in \mathbb{Z}_q$. The dealer secretly chooses (independently at random) $t-1$ elements in $\mathbb{Z}_q$ which are denoted $a_1 \ldots a_{t-1}$
3. For all i, the dealer computes $s_i = A(x_i)$, where

$$A(x) = K + \sum_{j=1}^{t-1} a_j x^j \pmod{q}$$

4. The dealer gives s_i secretly to all P_i

Figure 3.3: Shamir (t, w)-Threshold Scheme

and with

$$b_j = \prod_{1 \leq k \leq t, k \neq i} \frac{x_i}{x_k - x_i} \pmod{q}$$

we have

$$K = \sum_{i=1}^{t} s_i b_i \pmod{q} \tag{3.1}$$

We will use this scheme in our distribution of s. We assume we have several key generators. The main generator KG works like before, generates s, and will be closed before the election starts. In addition, we have several subgenerators KG_i where $i = 1, \ldots, w$. These will be open during the whole election and contain a public x_i.

The main KG plays the role of the *dealer*, and makes a function such that $A(0) = s$. He computes $s_i = A(x_i)$ and will pass s_i to KG_i for all i. Now, t participants will be able to compute s as in (3.1) by sharing $(s_i b_i)$.

However, we do not want the ballot box to be able to compute s, and $s_i b_i$ should therefore be encrypted. The distribution and encryption of s *before* and *during* the election will be explained in Fig. 3.4 and Fig. 3.5.

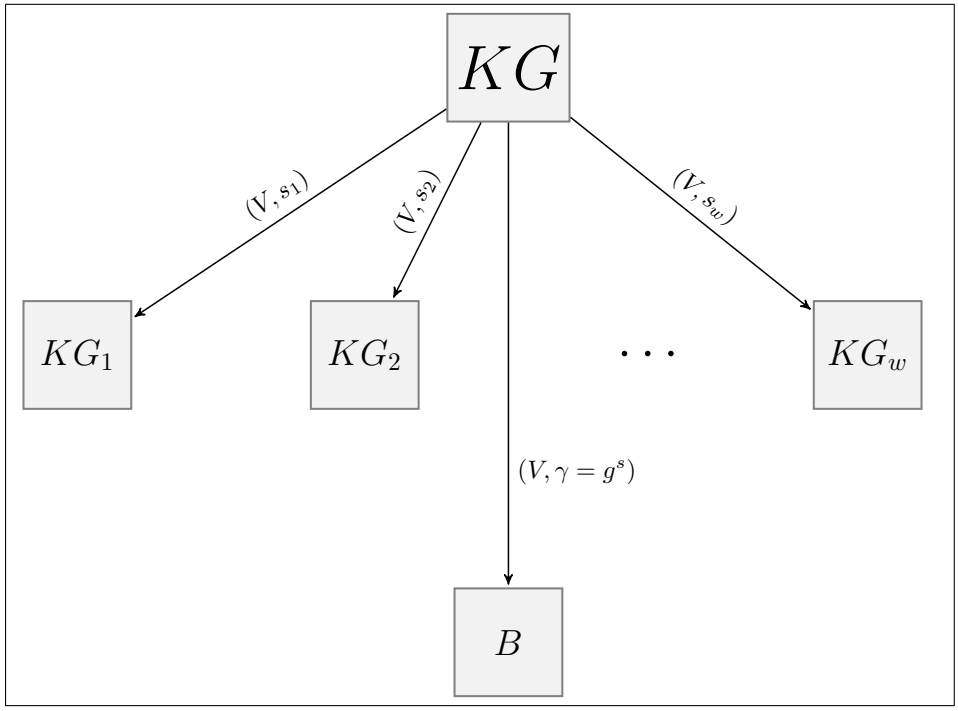


Figure 3.4: Distribution of s before election

KG_i will use an *public encryption key* ξ_{pk} received from the smart card together with the signature. KG_i encrypts $s_i b_i$ and $\xi(s_i b_i)$ will be sent to P through B. The computer will decrypt with the decryption key on the smart card, and compute s as in (3.1). With this distribution of s, P with access to the decryption key will still be the only player in possession of s during the election. A lost smart card can be replaced by a new one, containing a new decryption *and* encryption key. In practice, it means that a voter can lose and revoke as many smart card he wants during the election, and we can still generate new.

However, the more players introduced in the protocol, the more vulnerable for attacks. The advantage of using the Shamir (t, w)-treshold scheme is that the distribution of s will work even $(w - t)$ participants are out of action. The values of t and w depends on several factors. A bigger w allows us to have more players out of action, and demands less computing power from each player. The advantage to a smaller w is a smaller number of players to control, and it may be easier to implement in practice. We will not conclude with any numbers in this thesis.

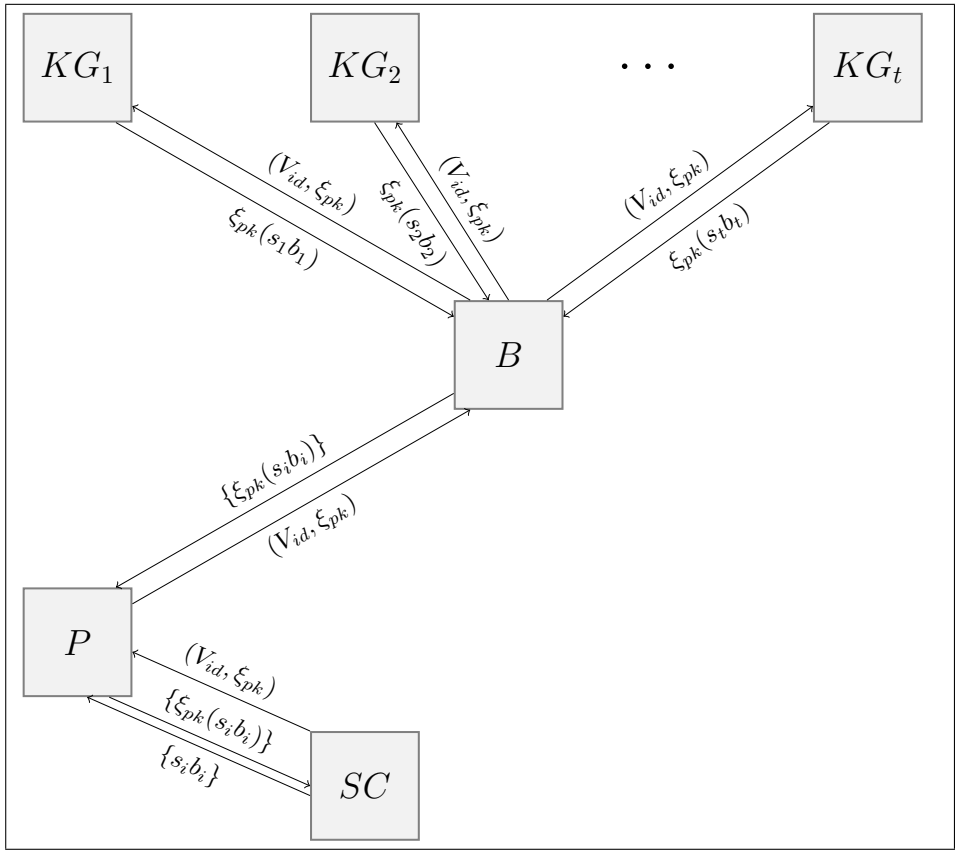


Figure 3.5: Distribution and encryption of s during election

3.2 Vote Submission

We want to describe how the vote submission works in the new protocol. Our aim is to make an improvement to the existing protocol, by making adjustments in the encryption of the votes, and in this way make it more secure. Except for these changes, we will mainly follow the programs described in detail for each player in Chap.4 in [Gjø10].

As long as nothing else is mentioned, we use the same ideal functionalities as well. They are described as trusted third parties which all the players can communicate securely with. They will replace needed subprotocols to simplify our analysis.

The *Ideal functionality for electronic identity* F_{eid} is to prove the identity of

the voter and *Ideal functionality for secure communication* F_{sc} is used for secure communication between the players.

The *Ideal functionality for discrete logarithm proof of knowledge* F_{pok} outputs a proof π of knowledge of the exponent ρ where $\eta = \epsilon^{\rho}$. It will output π on input (ϵ, η, ρ), with the assumption that (ϵ, η) is known for everyone.

The *ideal functionality for key generation* F_{key} generates the keys used in the protocol. The changes made in our F_{key} are described in the previous section.

The new vote submission will be as follows:

1. The voter selects his options $(v_1, \dots v_k)$ on the computer. Typically, v_1 will denote a political party, followed by candidates. The voter gives the computer access to his electronic identity and waits for a "ballot accepted"-message on his computer *and* receipts from the receipt generator out-of-band. He will before the election receive a *receipt code-set* with a table of receipt codes and options and will control the receipts against the receipt code-set.

2. As before, the computer sets $v_i = 0$ for $i = k+1, \cdots, k_{max}$. For $1 \leq i \leq k_{max}$, the computer chooses random t_i from $\{0, 1 \dots, q-1\}$. Now, for all i it encrypts the votes and computes the ballot:

   $(x_i, w_i) = (g^{t_i}, y_1^{t_i} f(v_i))$
   $(\hat{x}_i, \hat{w}_i) = (x_i^s, w_i^s)$
   $(\check{x}_i, \check{w}_i) = (g^{t_i s}, y_3^{t_i s} f(v_i)^s)$

   Note that $\check{x}_i = \hat{x}_i$. The computer signs the ballot on the voter's behalf.

3. The computer needs to show that the computation is done correctly, and prove the knowledge of the contents. The last proof is to prevent a corrupt ballot box feeding a non-honest voter with a ciphertext from an honest voter. If this is submitted, the receipt code sent to the non-honest voter will reveal the content of the vote. We will use F_{pok} to generate the proofs of knowledge. We can also use F_{pok} to prove equality of discrete logarithms, by proving knowledge of discrete logarithms in $G \times G \times G$. Thus, the computer
   Sends (g, x_i, t_i) to F_{pok} to show correctness of (x_i, w_i) and knowledge of v.
   Sends $(y_3 y_1^{-1}, \check{w}_i \hat{w}_i^{-1}, st)$ to F_{pok} to show correctness of $(\hat{w}_i, \check{w}_i)$. $\hat{w}_i$ and $\check{w}_i$ contain the same ballot if $\check{w}_i \hat{w}_i^{-1} = (y_3 y_1^{-1})^{st}$
   Sends $((g, x_i, w_i)(\gamma, \check{x}_i, \hat{w}_i), s)$ to F_{pok} to show equality of the exponents in $(\gamma, \check{x}_i, \hat{w}_i)$

   The computer sends $(x_i, \check{x}_i, w_i, \hat{w}_i, \check{w}_i)$ with the proofs and signature to B

and waits for R's signature that it has received the ballot before it informs the voter that the ballot has been accepted.

4. The ballot box verifies the signature and proofs. It enumerates the incoming ballots with ascending sequence numbers, this to make sure that only the last ballot will count from each voter. The ballot box will pass the entire ballot with signature, sequence number and proofs to R. The receipt generator replies with a signed hash for received ballot. When the election is over, B selects which ballots that should be decrypted. It computes for every voter j the product of the first ciphertexts, that is, $x_j = \prod_{i=1}^{k_{max}} x_i$ and $w_j = \prod_{i=1}^{k_{max}} w_i$ and passes it to D. The entire content of B is sent to A.

5. The receipt generator verifies the signatures and proofs from B, and signs a hash of the received ballot. It sends the signature back to B, which checks it, and passes it on to P. The receipt generator computes the receipts $\check{r}_i = d(\check{w}_i \check{x}_i^{-a_3})$ and sends $\check{r}_i$ directly to the voter. It makes a list of hashes of sign ballots to the auditor.

6. The decryption service receives a list of encrypted ballots from B. It waits for A's approval before it decrypts the ballots, and outputs them in a random order. The decryption service shows the correctness of this to the auditor.

7. The auditor monitors the protocol. It receives the entire content of B and the list of hashes from R. It verifies the content of B and controls it against the list from R to make sure that no ballots have been lost or inserted. A makes its own list over ballots that should be counted. It compares this list with the incoming ballots to the decryption service before it gives D its approval to decrypt.

The adjustments in our vote submission is the computation of $(\check{x}_i, \hat{w}_i, \check{w}_i)$. It was originally done by the ballot box, and we have moved it to the voter's computer. In addition, $\check{w}$ is changed and the protocol can now be executed without a_2. The proofs made by F_{pok} have also been adapted to our adjustments.

3.3 Security Analysis

We will in this section analyze the security for different cases of corruption:

- Receipt generator corrupt
- Ballot box corrupt

- Receipt generator and ballot box corrupt
- Computer corrupt

An analysis of the decryption service is presented in detail in Chap. 4, and an analysis of the auditor follows from this as well.

We want to show that the new protocol is an improvement of the existing. We show that the security is equal or better for each case of corruption, and analyze each of them, from the goals defined in [Gjø10]. We also look at some extra additional corruption scenarios in Sect. 3.3.4, not analyzed in the existing protocol.

3.3.1 Receipt Generator Corrupt

For the receipt generator, we use *game hopping* to show the goal defined in [Gjø10]:

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

We will in the games need a simulated F_{pok} indistinguishable from the real. We can make this by replacing the challenge given by the verifier by a random hash-function. In this way, it acts like an honest verifier. We can then use a HVZK-simulator with the correct challenge. A HVZK-simulator is clearly indistinguishable from the real and do not use the witness.

To simplify, we will assume in the description that there is only one vote per ballot. That is, $x_i = x, w_i = w$, etc. But the arguments should hold even when a ballot contains several votes. The games go like this:

Game 1
A machine M plays the role of every honest player and functionality. M also knows all secret keys. The encryption will be

$(x, w) = (g^t, y_1^t f(v))$
$(\hat{x}, \hat{w}) = (x^s, w^s)$
$(\check{x}, \check{w}) = (g^{ts}, y_3^{ts} f(v)^s)$

Game 2
Since M does all the work, it certainly knows which cleartext ballot corresponds to which encrypted ballot. In this game, the decryption service uses the cleartext instead of decrypting the ballots. Since an attacker can not see how D decrypts, this game is indistinguishable from the previous. Note that the decryption key

a_1 is no longer in use in D.

Game 3
We want F_{pok} to work even if the votes do not contain any contents. We will therefore use the simulated F_{pok}, and we can provide it a random witness. Because the simulated F_{pok} is indistinguishable from the real, the games are indistinguishable.

Game 4
We want to end up with a game without the exponent s, and where the components are independent and random. We want to remove the dependency between $\hat{w}$ and w and begin to express them in terms of $(x, \check{x})$ That is, change the computation of $(x, \check{x}, w, \hat{w}, \check{w})$ to :

$$\begin{array}{ll} x = g^t & w = x^{a_1} f(v) \\ & \hat{w} = \check{x}^{a_1} f(v)^s \\ \check{x} = x^s & \check{w} = \check{x}^{a_3} f(v)^s \end{array}$$

The games are indistinguishable since $y_1 = g^{a_1}$, and we still have the connection

$$\hat{w} = \check{x}^{a_1} f(v)^s = (x^s)^{a_1} f(v)^s = (x^{a_1} f(v))^s = w^s$$

Game 5
Further on, we also want $(x, \check{x})$ to be independent of each other. We compute $f(v_i)$ for all possible votes v_i, and let $\rho_{v_i} = f(v_i)^s$. That is for all v_i, we have the pairs $\{f(v_i), \rho(v_i)\}$. Let r_i be random, and we change the computation to

$$\begin{array}{ll} x = \prod f(v_i)^{r_i} & w = x^{a_1} f(v) \\ & \hat{w} = \check{x}^{a_1} \rho_v \\ \check{x} = \prod \rho_{v_i}^{r_i} & \check{w} = \check{x}^{a_3} \rho_v \end{array}$$

and since $\prod f(v_i)^{r_i} = g^t$ for some t, and

$$\check{x} = \prod \rho_{v_i}^{r_i} = \prod (f(v_i)^s)^{r_i} = (\prod f(v_i)^{r_i})^s = x^s$$

the games are indistinguishable.

Game 6
Let the computation be the same as in the previous game, but let ρ_{v_i} be picked random. We want $\rho_{v_i} \neq \rho_{v_j}$ for $i \neq j$. Assume an attacker can distinguish the two games. That means he will be able to determine whether ρ_{v_i} is a power of $f(v_i)$ or not. This is shown as hard in Game 8 Sect. 5.2 in [Gjø10]. Note that there is no longer a connection between x and $\check{x}$, nor between w and $\hat{w}$.

Game 7
Let t, u be random, and let the computation of $(x, \check{x}, w, \hat{w}, \check{w})$ be

$$\begin{array}{ll} x = g^t & w = y_1^t f(v) \\ & \hat{w} = y_1^u \rho_v \\ \check{x} = g^u & \check{w} = \check{x}^{a_3} \rho_v \end{array}$$

Since t, u are both random, and there is no longer a connection between x and $\check{x}$, the games are indistinguishable. We have also made the game independent of a_1 by replacing g^{a_1} by y_1.

Game 8
Our last step is to remove the connections between x and w and between $\check{x}$ and $\hat{w}$. We begin to define (g, y_1, b, c) as a DDH-tuple and let the computation be

$$\begin{array}{ll} x = g^t b^{t'} & w = y_1^t c^{t'} f(v) \\ & \hat{w} = y_1^u c^{u'} \rho_v \\ \check{x} = g^u b^{u'} & \check{w} = \check{x}^{a_3} \rho_v \end{array}$$

With a DDH-tuple, we have $\log_g y_1 = \log_b c = a_1$. We want to show that the games are indistinguishable. That is, we still have the connections $w = x^{a_1} f(v)$ and $\hat{w} = \check{x}^{a_1} \rho_v$.

Proof. Let

$$\begin{aligned} w &= y_1^t c^{t'} f(v) \\ &= g^{a_1 t} b^{a_1 t'} f(v) \\ &= (g^t b^{t'})^{a_1} f(v) \\ &= x^{a_1} f(v) \end{aligned}$$

Similarly, we have

$$\begin{aligned} \hat{w} &= y_1^u c^{u'} \rho_v \\ &= g^{a_1 u} b^{a_1 u'} \rho_v \\ &= (g^u b^{u'})^{a_1} \rho_v \\ &= \check{x}^{a_1} f(v) \end{aligned}$$

and the games are indistinguishable. □

Game 9
Let (g, y_1, b, c) now be a random tuple and the computation the same. We can show that a distinguisher between these two games are equivalent to a distinguisher for a DDH-tuple.

Proof. We sssume we have a distinguisher between Game 8 and Game 9 called A. That means we have an algorithm that on input (g, y_1, b, c) has a output depending on the input is a DDH-tuple or not. We therefore have a distinguisher for a DDH-tuple, by simply using the algorithm. This is known as hard and we can assume A do not exist. □

We want to have a sequence of indistinguishable games even when a ballot contains more than one vote. It is therefore important that we have two random variables for each component to make it indistinguishable with the next game. With only one random variable and several votes, we have dependency between the components and they will not be completely random as the next game requires.

We assume a ballot contains several votes from V, and only one variable. That is $x_i = b^{t'_i}, w_i = c^{t'_i} f(v)$, we get

$$x_i x_j^{-1} = g^{ut'} = b^{t'}$$
$$w_i w_j^{-1} = g^{vt'} = c^{t'}$$

and we have the connection

$$\log_b x_i x_j^{-1} = \log_c w_i w_j^{-1}$$

which makes them not independent.

With two random variables per component, we will show that the components are equal to

$$\begin{aligned} x_i &= g^{r_i} & x_j &= g^{r_j} \\ w_i &= y_1^{s_i} f(v), & w_j &= y_1^{s_j} f(v) \end{aligned} \tag{3.2}$$

with r_i, r_j, s_i, s_j all random. We have

$$\begin{aligned} x_i &= g^{t_i} b^{t'_i}, & x_j &= g^{t_j} b^{t'_j} \\ w_i &= y_1^{t_i} c^{t'_i} f(v), & w_j &= y_1^{t_j} c^{t'_j} f(v) \end{aligned}$$

With (g, y_1, b, c) a random tuple, we have for some random u, v, $b = g^u$ and $c = y^v$. We can compute

$$\begin{aligned} x_i &= g^{t_i} g^{ut'_i}, & x_j &= g^{t_j} g^{ut'_j} \\ w_i &= y_1^{t_i} y^{vt'_i} f(v), & w_j &= y_1^{t_j} y^{vt'_j} f(v) \end{aligned}$$

$\Longrightarrow$

$$x_i = g^{t_i + ut'_i}, \qquad x_j = g^{t_j + ut'_j}$$
$$w_i = y_1^{t_i + vt'_i} f(v), \qquad w_j = y_1^{t_j + vt'_j} f(v)$$

which gives us for each couple (x_i, x_j) and (w_i, w_j) two equations with two unknown,

$$t_i + ut'_i = r_i \qquad t_i + vt'_i = s_i$$
$$t_j + ut'_j = r_j \qquad t_j + vt'_j = s_j$$

and the components can be written as in (3.2), with exponents random since u and v are random.

Game 10
Let the final game be the computation with (t_1, t_2, u_1, u_2) random and

$$x = g^{t_1} \qquad w = g^{t_2}$$
$$\hat{w} = g^{u_2}$$
$$\check{x} = g^{u_1} \qquad \check{w} = \check{x}^{a_3} \rho_v$$

This is indistinguishable with previous game. With (t_1, t_2, u_1, u_2) all random, we do not have any connection between any of the components except between $\check{x}$ and $\check{w}$. We need this last connection for R to be able to give similar receipt codes for similar votes.

This game do not contain any information about the votes, and we have a final game equal to the final game in the analysis of R in [Gjø10]. R^* will therefore learn no unavoidable information about the submitted ballots. However, some untested cryptographic assumptions are used in the proof in Game 6, obtained from the existing protocol. Gjøsteen only conclude that the receipt generator will be *quite well* protected. We will from now on assume that this is good enough.

3.3.2 Ballot Box Corrupt

Our ballot box differ from the existing because our will not do any encryption or contain a secret key. We therefore define the goal for a corrupt ballot box from: *Any vote submitted through an honest computer should remain confidential* [Gjø10], and it will be

- A corrupt Ballot box should not be able to extract any information about the ballot

The ballot box receives the same information as R, but do not have any secret key to recognize similar receipt codes for similar votes. We will therefore continue from Game 10 in Sect. 3.3.1. That is

Game 10
Let (t_1, t_2, u_1, u_2) random and

$$\begin{array}{ll} x = g^{t_1} & w = g^{t_2} \\ & \hat{w} = g^{u_2} \\ \check{x} = g^{u_1} & \check{w} = \check{x}^{a_3}\rho_v \end{array}$$

We also let the the receipt generator use the cleartexts to compute the receipt codes instead of computing them out of $\check{x}_i, \check{w}_i$. In this way, a_3 is no longer in use in R. Since the attacker can not see how R makes his receipt codes, the games are indistinguishable.

Game 11
We want to remove the last connection between $\check{x}$ and $\check{w}$. Let (g, y_3, b, c) be a DDH-tuple and compute:

$$\begin{array}{ll} x = g^{t_1} & w = g^{t_2} \\ & \hat{w} = g^{u_2} \\ \check{x} = g^{u_1}b^{u_3} & \check{w} = y_3^{u_1}c^{u_3}\rho_v \end{array}$$

This game is indistinguishable from the previous game since $\log_g y_3 = \log_b c = a_3$ and similar to the proof made in Game 8.

Game 12
Let (g, y_3, b, c) be a random tuple. As before, we do not have any distinguisher for a DDH-tuple, and the games are indistinguishable. The components are now all random and do not contain any information about the votes. And B^* will not be able to extract any information about the votes.

3.3.3 Receipt Generator and Ballot Box Corrupt

We want to analyze the protocol when both the receipt generator and the ballot box are corrupt. The ballot box does not change or add anything, and the knowledge an attacker can extract from this situation is the same that he can extract when R^* is corrupt. It will follow from Sect. 3.3.1, and we can again conclude that it is *quite* well protected.

However, a corrupt ballot box and receiptgenerator will still give an attacker a chance to erase votes. Assume B^* receives a ballot with a vote v. R^* sees it and signs the hash which will be passed back to P through B. It also computes the

receipt code to V. The voter V will now see that the vote has been accepted on the computer, and received the right receipt code. Then the election is over, B^* and R^* can remove the ballot from the ballot box, *and* remove the corresponding sign hash on the list sent from R to A. In this way, neither the voter V or auditor A will notice an attack.

The advantage in our protocol compared with the existing is that the attacker will not know the contents of the ballots he may remove.

3.3.4 Computer Corrupt

A corrupt computer P^* will follow the analysis discussed in detail in Sect. 5.1 in [Gjø10] and will still have a chance to delay and reverse the order of the ballots submitted by an honest voter and there will be a chance that the wrong ballot will count. For this reason, Gjøsteen arguing that a voter receiving a wrong receipt code, should not vote electronic again, but rather use paper votes. We will analyze a corrupt P^* together with a corrupt R^* or B^*.

We will distinguish between a voter *using* a corrupt computer, and a voter that has been *in touch* with a corrupt computer. In the last situation, the votes are coming from an honest computer, but the attacker will still be in possession of the personal exponent s.

Corrupt P^* and R^*

For a voter *using* a corrupt computer P^*, and R^* is corrupt, an attacker will be able to forge votes. An honest voter chooses v on the corrupt computer. But P^* can submit whatever v^* he wants to B while R^* provides the "right" receipt code to V. R^* will also make a list for A matching the content of B and the voter and auditor will not notice. In this way, the attacker can control the whole protocol.

Assume V has been *in touch* with a corrupt computer, with receipt generator also corrupt. The attacker will then be in possession of both a_3 and s and will be able to decrypt the votes:

$$(\check{w}\check{x}^{-a_3})^{s^{-1}} = (f(v)^s)^{s^{-1}} = f(v)$$

However, even if an attacker with this access can identify votes, it will not be able to change or destroy votes.

Corrupt P^* and B^*

We have shown that a corrupt ballot box will not be able to extract any information about the votes. A corrupt B^* will not be able to change the content without being noticed by the auditor. A corrupt B^* and a voter *using* P^* will therefore not make any more damage than a corrupt P^* alone.

We can show that for a voter who has been *in touch* with a corrupt computer, and the ballot box corrupt, an attacker will still not learn anything new about the votes. Our goal is to make the final game without any content of the votes. Note that we need in this situation to preserve the "s"-connection between the components in the games because an attacker is in possession of it. Continue on the computation in Game 4 from 3.3.1, that is

Game 4

$$\begin{array}{ll} x = g^t & w = x^{a_1} f(v) \\ & \hat{w} = \check{x}^{a_1} f(v)^s \\ \check{x} = x^s & \check{w} = \check{x}^{a_3} f(v)^s = y_3^{ts} f(v)^s \end{array}$$

Game 5'
We want to hide a_1 to be able to remove the real content $f(v)$ from w. Introduce a_2 and define a_1 as $a_1 = a_3 - a_2 \pmod g$. We can then make $y_3 = y_1 g^{a_2}$. In this way, the computation can be done without a_1 and a_3.

$$\begin{array}{ll} x = g^t & w = y_1^t f(v) \\ & \hat{w} = w^s \\ \check{x} = x^s & \check{w} = \hat{w} \check{x}^{a_2} \end{array}$$

To show the games are indistinguishable, we want to show that $y_3 = y_1 g^{a_2} = g^{a_3}$ and $\check{w} = y_3^{ts} f(v)^s$ in Game 5'. Have

$$y_3 = y_1 g^{a_2} = y_1 g^{a_3 - a_1} = g^{a_1} g^{a_3 - a_1} = g^{a_3}$$

and

$$\begin{aligned} \check{w} = w^s \check{x}^{a_2} &= w^s x^{s(a_3 - a_1)} \\ &= y_1^{ts} f(v)^s g^{ts(a_3 - a_1)} \\ &= g^{a_1 ts} f(v)^s g^{ts(a_3 - a_1)} \\ &= g^{tsa_3} f(v)^s = y_3^{ts} f(v)^s \end{aligned}$$

Game 6'
Let (g, y_1, b, c) be a DDH-tuple. Let

$$\begin{array}{ll} x = g^t b^{t'} & w = y_1^t c^{t'} f(v) \\ & \hat{w} = w^s \\ \check{x} = x^s & \check{w} = \hat{w} \check{x}^{a_2} \end{array}$$

With $c = b^{a_1}$, we have

$$w = y_1^t c^{t'} f(v) = g^{a_1 t} b^{a_1 t'} f(v) = (g^t b^{t'})^{a_1} f(v) = x^{a_1} f(v) = y_1^t f(v)$$

as in the previous game, and they are indistinguishable.

Game 7'
Let the computation be the same, with (g, y_1, b, c) a random tuple. Indistinguishable since we still do not have a distinguisher for a DDH-tuple as shown in Sec 3.3.1.

Game 8'
We have now removed the connection between x and w, which makes w totally random. We can replace $f(v)$ by a random group element and we get

$$\begin{array}{ll} x = g^t & w = g^{t'} \\ & \hat{w} = w^s \\ \check{x} = x^s & \check{w} = \hat{w}\check{x}^{a_2} \end{array}$$

and the game is clearly indistinguishable from the previous game. A corrupt B^* with access to s will therefore not learn anything new about the votes.

3.4 Improvements

Our goal was to remove the connection between the secret keys to avoid a corrupt B^* and R^* to decrypt the votes. This is accomplished, and we have not found any other part with less security in the new protocol. The protection against a corrupt R^* or corrupt B^* will both still satisfy the goals defined in [Gjø10].

We will also compare the new protocol with the existing with respect to the performance. We have saved computations for the infrastructure players by moving all the encryption to the voter's computer. However, the proof of knowledge for all the ciphertexts will in the new protocol be verified by both R and B. Generating a proof of knowledge in F_{pok} for $(x_1 \dots x_{k_{max}})$ costs k_{max} exponentiations, while verifying costs $2k_{max}$. There are three proofs given to F_{pok} in Sect. 3.2. The total exponentiations for R and B to verify n votes will be given in Fig. 3.6. Note that $k_{max} = k$

Improved protocol

Operation	of	B	R	Total
Verify F_{pok}	(x_i, w_i)	$2kn$	$2kn$	
Verify F_{pok}	$(\check{w}_i, \hat{w}_i)$	$2kn$	$2kn$	
Verify F_{pok}	$(g, x, w)^s = (\gamma, \check{x}, \hat{w})$	$6kn$	$6kn$	
Computing	r_i		kn	
Sum		$10kn$	$11kn$	$21kn$

Figure 3.6: Exponentiations for R and B in the new protocol

The cost for R and B in the existing protocol,including encryption, generating, and verifying of proofs are given in Fig. 3.7. Note that $k = k_{max}$.

Existing protocol

Operation	of	B	R	Total
Encryption	$(\check{w}, \bar{w}, \bar{x})$	$3kn$		
Generating proof	$\bar{\pi}$	$3kn$		
Generating proof	$\check{\pi}$	$2kn$		
Verifying proof	π	$2kn$	$2kn$	
Verifying proof	$\bar{\pi}$		$6kn$	
Verifying proof	$\check{\pi}$		$4kn$	
Computing	r_i		kn	
Sum		$10kn$	$13kn$	$23kn$

Figure 3.7: Exponentiations for R and B in the existing protocol

It means we have a small improvement in the performance in the new protocol for the infrastructure players. The numbers of exponentiations for the computers has increased, but the computers will only do the exponentiations for one ballot at time.

The conclusion is that the new protocol is an improvement with respect to the security as wanted. In addition, we also have a small improvement of the performance.

Chapter 4

Improving the Decryption Service

The decryption service D decrypts the ballots. The auditor will see both the incoming encrypted ballots and the decrypted ballots. Because A has access to the whole content of B, it will be possible to connect which incoming ballot corresponds to which signature and voter. To avoid a corrupt A^* connecting incoming ballots with the decrypted ballots, we want the decryption service to output the ballots in random order. Furthermore, the decryption service does not only need to prove the correctness of the decryption, but also that the decrypted ballots actually are a permutation of the incoming.

In the existing proof to show the correctness of permutation and decryption, an attacker will according to [Gjø10] has $\frac{1}{100}$ chance of manipulating two votes. We want to replace this proof by a *Special-Honest-Verifier-Zero-Knowledge Arguments of knowledge*(SHVZK), which will increase the security. It will minimize the chance for D^* manipulating votes, and it will also give us a protection against A^*.

We have made two SHVZK-arguments of knowledge to the given problem, and will compare them in the end of the chapter.

4.1 Decryption Service 1

This first SHVZK-argument of knowledge is a modification of the SHVZK *Argument of Shuffle of Homomorphic Encryptions* by Groth in [Gro10]. While Groth wants to show that two sets of chiphertexts have the same contents, we want in addition to show that the second set actually is the decryption of the first.

We will use the same parameters used by Groth, and demand $2^{l_e+l_s} < q$. Groth also discusses the sizes of l_e, l_s to preserve the security. The encryption and commitments spaces are $\mathbb{Z}_q$.

The protocol for our new argument is shown in Fig. 4.1, where the prover P will play the role of D, and the verifier V the role of A. We will show that this is a *SHVZK argument of knowledge* in the next subsections.

We use two subprotocols. $\text{Arg}(\pi, \rho)$ is a *SHVZK argument of known content* from [Gro10]. It is an argument to show knowledge of a permutation π. For a known set $(m_1 \dots m_n)$, the prover convinces the verifier that a commitment c contains a permutation of $(m_1 \dots m_n)$.

In our case, $(\lambda i + t_i)$ is the known contents, while $c^{\lambda} c_d com(f_1 \dots f_n)$ is the commitment containing the permutation. The cost of this subprotocol is given as $3n$ exponentiations for the prover, and $2n$ for the verifier [Gro10].

The second subprotocol is $\text{Arg}(\xi_{id})$, described in 2.7. It shows that a tuple (x_0, w_0) is an encryption of the identity. With $(x_i, w_i) = (g^{t'_i}, y^{t'_i} m_i)$, the verifier will in the protocol compute

$$
\begin{aligned}
& \prod_{i=1}^{n} (x_i, w_i)^{-t_i} \xi(\prod_{i=1}^{n} M_i^{f_i}; 0) \cdot E_d \\
= \; & \prod_{i=1}^{n} (g^{t'_i}, y_1^{t'_i} m_i)^{-t_i} \cdot (g^0, y_1^0 \prod_{i=1}^{n} M_i^{f_i}) \cdot (g^{R_d}, y_1^{Rd} \prod_{i=1}^{n} M_i^{-d_i}) \\
= \; & (g^{\sum_{i=1}^{n} -t'_i t_i + R_d}, y_1^{\sum_{i=1}^{n} -t'_i t_i + R_d} \prod_{i=1}^{n} (m_i^{-t_i} M_i^{f_i} M_i^{-d_i})) \\
= \; & (g^{\sum_{i=1}^{n} -t'_i t_i + R_d}, y_1^{\sum_{i=1}^{n} -t'_i t_i + R_d} \prod_{i=1}^{n} (m_i^{-t_i} M_i^{t_{\pi(i)}} M^{d_i} M^{-d_i}))
\end{aligned}
$$

as long as $M_i = m_{\pi(i)} = w_{\pi(i)} x_{\pi(i)}^{-a_{\pi(i)}}$ and with no modular reduction of f_i it gives

$$
= \; (g^{\sum_{i=1}^{n} -t'_i t_i + R_d}, y_1^{\sum_{i=1}^{n} -t'_i t_i + R_d} \cdot 1) = \xi(1, \sum_{i=1}^{n} -t'_i t_i + R_d) = \xi_{id}
$$

$\text{Arg}(\xi_{id})$ shows that this is an encryption of the identity. To save exponentiations,

the prover computes the same $\xi_{id} = \xi(1, \sum_{i=1}^{n} -t'_i t_i + R_d)$ as

$$\begin{aligned}
& (\prod_{i=1}^{n} x_i^{-t_i} g^{R_d}, \prod_{i=1}^{n} w_i^{-t_i} y_1^{R_d}) \\
= \; & (\prod_{i=1}^{n} (g^{t'_i})^{-t_i} g^{R_d}, \prod_{i=1}^{n} (y_1^{t'_i})^{-t_i} y_1^{R_d}) \\
= \; & (g^{\sum_{i=1}^{n} -t'_i t_i + R_d}, y_1^{\sum_{i=1}^{n} -t'_i t_i + R_d}) \\
= \; & \xi(1, \sum_{i=1}^{n} -t'_i t_i + R_d) = \xi_{id}
\end{aligned}$$

PROVER	Common input: $(x_1, w_1) \dots (x_n, w_n)$ $M_1, \dots, M_n$	VERIFIER
Private input : π, a_1 s.t $M_i = m_{\pi(i)} = w_{\pi(i)} x_{\pi(i)}^{-a_{\pi(i)}}$		
$r, r_d \leftarrow \mathbb{Z}_q = R_{ck}$, $R_d \leftarrow \mathbb{Z}_q = R_{pk}$ $d_1, d_2, ..., d_n \leftarrow \{0,1\}^{l_s+l_e}$		
c=com$(\pi(1), \pi(2), ..., \pi(n); r)$ c_d=com$(-d_1, -d_2, \dots, -d_n; r_d)$		
$E_d = \xi(\prod_{i=1}^n M_i^{-d_i}; R_d)$ $= (g^{R_d}, y^{Rd} \prod_{i=1}^n M_i^{-d_i})$		
	$\xrightarrow{c,\ c_d,\ E_d}$	
		$t_i \leftarrow \{0,1\}^{l_e}$
	$\xleftarrow{t_1,\dots,t_n}$	
$f_i = t_{\pi(i)} + d_i$		
	$\xrightarrow{f_1,\dots,f_n}$	
		$\lambda \leftarrow \{0,1\}^{l_e}$
	$\xleftarrow{\lambda}$	
Arg$(\pi, \rho; c^\lambda c_dcom(f_1, \cdot\cdot, f_n; 0)$ =com$(\lambda\pi(1) + t_{\pi(1)}$ $, ..., \lambda\pi(n) + t_{\pi(n)}; \rho)$	$\longleftarrow$ $\longrightarrow$ $\longleftarrow$ $\longrightarrow$	
		Check that $c, c_d \in \mathbb{Z}_q, E_d \in \mathbb{Z}_q \times \mathbb{Z}_q$ $2^{l_e} \leq (f_1, .., f_n) < 2^{l_e+l_s}$ Verify **Arg**(π, ρ)
Compute $\xi_{id} = (x_o, w_o) =$ $(\prod_{i=1}^n x_i^{-t_i} g^{Rd}, \prod_{i=1}^n w_i^{-t_i} y_1^{Rd})$		Compute $\prod_{i=1}^n (x_i, w_i)^{-t_i} \cdot$ $\xi(\prod_{i=1}^n M_i^{f_i}; 0) \cdot E_d$ $= (x_o, w_o) = \xi_{id}$
Arg(ξ_{id})	$\longrightarrow$ $\longleftarrow$ $\longrightarrow$	Verify **Arg**(ξ_{id})

Figure 4.1: The protocol for Decryption service 1

4.1.1 SHVZK

Completeness is straight forward to verify when we do not have any modular reduction of f_i. We will not when $2^{l_e} \leq (f_1, .., f_n) < 2^{l_e+l_s}$. There is overwhelming probability for this as long as l_s is sufficiently large [Gro10]. The *Soundness* property will follow from the Witness-emulation in the next subsection. We create a simulator for the prover to show we have the *SHVZK-property*. The simulator takes e as input and has no access to the private inputs.

Simulator

$r, r_d, R', R_d \leftarrow \mathbb{Z}_q$
$f_1, .., f_n \leftarrow \mathbb{Z}_q$

$c = com(0, .., 0; r)$
$c_d = com(0, ..0; r_d)$
$E_d = \xi(1, R') \prod_{i=1}^{n} (x_i, w_i)^{t_i}$
$\xi(\prod_{i=1}^{n} M_i^{-f_i}; 0)$
$\xi_{id} = (\prod_{i=1}^{n} x_i^{-t_i} g^{Rd}, \prod_{i=1}^{n} w_i^{-t_i} y_1^{Rd})$
Sim Arg(π, ρ)
Sim Arg(ξ_{id})

Hybrid 1

$r, r_d, R', R_d \leftarrow \mathbb{Z}_q$
$f_1, .., f_n \leftarrow \mathbb{Z}_q$
$d_i = f_i - t_{\pi(i)}$
$c = com(\pi(1), .., \pi(n); r)$
$c_d = com(d_1, ..d_n; r_d)$
$E_d = \xi(1, R') \prod_{i=1}^{n} (x_i, w_i)^{t_i}$
$\xi(\prod_{i=1}^{n} M_i^{-f_i}; 0)$
$\xi_{id} = (\prod_{i=1}^{n} x_i^{-t_i} g^{Rd}, \prod_{i=1}^{n} w_i^{-t_i} y_1^{Rd})$
Sim Arg(π, ρ)
Sim Arg(ξ_{id})

Figure 4.2: Differences between Simulator and Hybrid 1

The differences between the *Simulator* and *Hybrid 1* in Fig. 4.2 are the contents of c and c_d. These are statistically indistinguishable because of the *unconditional hiding property* of the commitments. The simulated arguments, Sim Arg(π, ρ) and Sim Arg(ξ_{id}), are both independent of d_i and therefore still the same. We can conclude that they are indistinguishable.

Hybrid 1

$r, r_d, R', R_d \leftarrow \mathbb{Z}_q$
$f_1, .., f_n \leftarrow \mathbb{Z}_q$
$d_i = f_i - t_{\pi(i)}$
$c = com(\pi(1), .., \pi(n); r)$
$c_d = com(d_1, ..d_n; r_d)$
$E_d = \xi(1, R') \prod_{i=1}^{n} (x_i, w_i)^{t_i} \xi$
$(\prod_{i=1}^{n} M_i^{-f_i}; 0)$
$\xi_{id} = (\prod_{i=1}^{n} x_i^{-t_i} g^{Rd}, \prod_{i=1}^{n} w_i^{-t_i} y_1^{Rd})$
Sim Arg(π, ρ)
Sim Arg(ξ_{id})

Hybrid2

$r, r_d, R', R_d \leftarrow \mathbb{Z}_q$
$f_1, .., f_n \leftarrow \mathbb{Z}_q$
$d_i = f_i - t_{\pi(i)}$
$c = com(\pi(1), .., \pi(n); r)$
$c_d = com(d_1, ..d_n; r_d)$
$E_d = \xi(\prod_{i=1}^{n} M_i^{-d_i}; R_d)$

$\xi_{id} = (\prod_{i=1}^{n} x_i^{-t_i} g^{Rd}, \prod_{i=1}^{n} w_i^{-t_i} y_1^{Rd})$
Sim Arg(π, ρ)
Sim Arg(ξ_{id})

Figure 4.3: Differences between hybrid 1 and hybrid 2

Hybrid 1 and *Hybrid 2* differ in the way E_d is computed. We can show that they are indistinguishable. Note that $(x_i, w_i) = (g^{t_i'}, y^{t_i'} m_i)$. We have in Hybrid 1

$$
\begin{aligned}
E_d &= \xi(1, R') \prod_{i=1}^{n} (x_i, w_i)^{t_i} \xi(\prod_{i=1}^{n} M_i^{-f_i}; 0) \\
&= \xi(1, R') \prod_{i=1}^{n} (x_i, w_i)^{t_i} \xi(\prod_{i=1}^{n} M_i^{-t_{\pi(i)} - d_i}; 0) \\
&= (g^{R'}, y^{R'} \cdot 1) \prod_{i=1}^{n} (g^{t_i'}, y^{t_i'} m_i)^{t_i} (g^0, y^0 \prod_{i=1}^{n} M_i^{-t_{\pi(i)} - d_i}) \\
&= (g^{R'}, y^{R'} \cdot 1) (g^{\sum_{i=1}^{n} t_i' t_i}, y^{\sum_{i=1}^{n} t_i' t_i} \prod_{i=1}^{n} m_i^{t_i}) (g^0, y^0 \prod_{i=1}^{n} M_i^{-t_{\pi(i)} - d_i}) \\
&= (g^{R^*}, y^{R^*} \prod_{i=1}^{n} m_i^{t_i} M_i^{-t_{\pi(i)} - d_i}) = (g^{R^*}, y^{R^*} \prod_{i=1}^{n} M_i^{-d_i}) \\
&= \xi(\prod_{i=1}^{n} M_i^{-d_i}; R^*) \text{ where } R^* = R' + \sum_{i=1}^{n} t_i' t_i
\end{aligned}
$$

R^* is random since R' is random. Hybrid 1 and Hybrid 2 are therefore indistinguishable since $E_d = \xi(\prod_{i=1}^{n} M_i^{-d_i}; R_d)$ with R_d random.

Hybrid2	**Real prover**
$r, r_d, R', R_d \leftarrow \mathbb{Z}_q$	$r, r_d, R_d \leftarrow \mathbb{Z}_q$
$f_1, .., f_n \leftarrow \mathbb{Z}_q$	$d_1, d_2, ..., d_n \leftarrow \mathbb{Z}_q$
$d_i = f_i - t_{\pi(i)}$	$f_i = t_{\pi(i)} + d_i$
$c = com(\pi(1), .., \pi(n); r)$	$c = com(\pi(1), .., \pi(n); r)$
$c_d = com(d_1, ..d_n; r_d)$	$c_d = com(d_1, ..d_n; r_d)$
$E_d = \xi(\prod_{i=1}^n M_i^{-d_i}; R_d)$	$E_d = \xi(\prod_{i=1}^n M_i^{-d_i}; R_d)$
$\xi_{id} = (\prod_{i=1}^n x_i^{-t_i} g^{Rd}, \prod_{i=1}^n w_i^{-t_i} y_1^{Rd})$	$\xi_{id} = (\prod_{i=1}^n x_i^{-t_i} g^{Rd}, \prod_{i=1}^n w_i^{-t_i} y_1^{Rd})$
Sim Arg(π, ρ)	Arg(π, ρ)
Sim Arg(ξ_{id})	Arg(ξ_{id})

Figure 4.4: Differences between Hybrid 2 and the real P

$f_i's$ and $d_i's$ differ between *Hybrid 2* and the *Real prover* in Fig.4.4. They will be indistinguishable because the probability distribution is the same, no matter which order they are picked/computed. The *Simulated Arguments* for π and ξ_{id} are both SHVZK arguments by [Gro10] and Sect.2.7, and therefore indistinguishable from the real arguments. We can therefore conclude that the simulator exists and the argument is SHVZK.

4.1.2 Witness-Extended Emulation

We want to create an emulator E that whenever a deterministic polynomial time prover P^* can make a convincing argument, E extracts the witnesses in expected polynomial time. We use the emulator E' for Arg(π, ρ) in [Gro10], and the emulator E'' for Arg(ξ_{id}) by Sect.2.7 in our construction of E. It works like

E runs $\langle P^*, V \rangle$ as in the real protocol which outputs the transcript $(c, c_d, E_d, t_1.., t_n, f_1.., f_n, \lambda,\ \mathrm{tr}_{known},\ \mathrm{tr}_{\xi_{id}})$, where tr_{known} is the transcript of Arg(π, ρ) and $\mathrm{tr}_{\xi_{id}}$ the transcript for Arg(ξ_{id}). If P^* fails and V does not accept, E outputs (tr,$\perp$). Otherwise, we want to show that E extracts the witnesses π and a_1.

In order to extract the witnesses, we will rewind the transcript $(c, c_d, E_d, t_1.., t_n, f_1.., f_n, \lambda,\ \mathrm{tr}_{known})$ and run E'' on $Arg(\xi_{id})$ until we get the witness a_1. We can go further back and get two new transcripts on random chosen challenges $(t_i \dots t_n, \lambda)$ and $(t_i^{'} \dots t_n^{'}, \lambda')$ until E' gets the openings $(\pi_1, \rho_1), (\pi_2, \rho_2)$ of the commitments containing permutations of respectively $(\lambda i + t_i)$ and $(\lambda' i + t_i^{'})$. That means we have

$$c^{\lambda} c_d com(f_1, \dots, f_n; 0) = com(\lambda\pi_1(1) + t_{\pi_1(1)}, \dots, \lambda\pi_1(n) + t_{\pi_1(n)})$$

and

$$c^{\lambda^{'}} c_d com(f_1^{'}, \ldots, f_n^{'}; 0) = com(\lambda^{'}\pi_2(1) + t_{\pi_2(1)}^{'}, \ldots, \lambda^{'}\pi_2(n) + t_{\pi_2(n)}^{'})$$

Combining this:

$$c^{\lambda - \lambda^{'}} = com(f_1^{'} - f_1 + \lambda\pi_1(1) - \lambda^{'}\pi_2(1) + t_{\pi_1(1)} - t_{\pi_2(1)}^{'}, \ldots)$$

and we get an opening $(\mu_1, \ldots, \mu_n)$ of c by the *root property* of the commitments.

We want to argue that this really is a permutation of $\{1, \ldots, n\}$, that is $\mu_i = \pi(i)$. Assume we have two accepting argument which gives us the opening $(\mu_1, \ldots, \mu_n)$ of c with a probability ϵ^2. Then we have the probability ϵ^3 to get the third transcripts such that

$$\lambda\mu_i - d_i + f_i = \lambda\pi(i) + t_{\pi(i)}$$

$$\Rightarrow \lambda(\mu_i - \pi(i)) - d_i + f_i - t_{\pi(i)} = 0.$$

The f_i's are sent by the prover before it receives λ, and it is a overwhelming probability that $\mu_i = \pi(i)$ and $f_i = t_{\pi(i)} + d_i$.

It remains to argue that E uses expected polynomial time. We know that both $E^{'}$ and $E^{''}$ use expected polynomial time on their real arguments. The transcripts are indistinguishable from the real argument with the same probability distribution, and we can assume they will use the same computing time on the transcripts as well.

Assume that a cheating P^* has the probability ϵ to get an accepting argument on the whole protocol. $Arg(\pi, \rho)$ and $Arg(\xi_{id})$ are parts of the protocol, and we can assume ϵ is a lower bound for their probability for an accepting argument. Therefore, assuming ϵ will be the probability for $E^{'}$ and $E^{''}$ as well, we will get an upper bound on expected runs for E. Expected runs for $E^{'}$ getting 2 accepting arguments are $\frac{2}{\epsilon}$ and for $E^{''}$ getting 1 is $\frac{1}{\epsilon}$. There will only be one run if V does not accept, and the probability for this is $1 - \epsilon$. Total expected runs for E will therefore be $(1 - \epsilon) \cdot 1 + (1 + \frac{2}{\epsilon} + \frac{1}{\epsilon}) \cdot \epsilon = 4$ runs.

4.2 Decryption Service 2

We will in this section present the second suggestion of a *SHVZK-argument of knowledge*. This one is based on Groth's *Argument of Shuffle and Decryption of ElGamal Ciphertexts*[Gro10], and we will use the same parameters as in Sect. 4.1.

.

Groth's argument is based on mix-nets which is a multi-party protocol. It has several mix-servers which, one by one, mixing and part decrypting the ciphertexts.

Linking inputs and outputs is impossible if at least one mix-server is honest. In our case, there will only be one mix-server doing a permutation and a decryption. We will therefore make several modifications and the argument will be presented in the protocols in Fig.4.5 and Fig.4.6.

PROVER		VERIFIER
	Common input $(x_1, w_1)....(x_n, w_n)$ $M_1, ..., M_n$	
Private input : π, a_1 s.t $M_i = m_{\pi(i)} = w_{\pi(i)} \cdot x_{\pi(i)}^{-a_{\pi(i)}}$		
$r, r_d, r_1, r_2 \leftarrow \mathbb{Z}_q = R_{ck}$ $d_x, r_v, d_v, \leftarrow \mathbb{Z}_q$ $d_1, d_2, ..., d_n \leftarrow \{0,1\}^{l_s+l_e}$,		
c=com$(\pi(1), \pi(2), ..., \pi(n); r)$ c_d=com$(-d_1, -d_2, ..., -d_n; r_d)$		
$M = g^{r_v} \prod_{i=1}^{n} M_i^{-d_i}$ $D = g^{d_x}$, $c_1 = com(r_v; r_1)$, $c_2 = com(d_v; r_2)$		
	$\xrightarrow{c,\ c_d,\ E_d}$	
	$\xrightarrow{c_1,\ c_2,\ D, M}$	
		$t_i \leftarrow \{0,1\}^{l_e}$
	$\xleftarrow{t_1,...,t_n,}$	
$f_i = t_{\pi(i)} + d_i$ $U = g^{d_v}(\prod_{i=1}^{n} x_i^{-t_i})^{d_x}$		
	$\xrightarrow{f_1,...,f_n,U}$	
		$e, \lambda \leftarrow \{0,1\}^{l_e}$
	$\xleftarrow{\lambda, e}$	
Arg$(\pi, \rho; c^{\lambda} c_dcom(f_1, \ldots, f_n; 0)$ =com$(\lambda\pi(1) + t_{\pi(1)}$ $, ..., \lambda\pi(n) + t_{\pi(n)}; \rho)$	$\longleftarrow$ $\longrightarrow$ $\longleftarrow$ $\longrightarrow$	

Figure 4.5: The protocol for Decryption service 2. Part 1

PROVER	VERIFIER
$f = ea_1 + d_x$ $f_v = er_v + d_v$ $z_v = er_1 + r_2$	
$\xrightarrow{f, f_v, z_v}$	
	Check that $c, c_d, c_1, c_2 \in \mathbb{Z}_q$ $M, D, U \in \mathbb{Z}_q$ and that $2^{l_e} \leq f_1, \ldots, f_n < 2^{l_e + l_s}$ Verify $\mathbf{Arg}(\pi, \rho)$ Check that $\prod_{i=1}^{n}(x_i^{-t_i})^{-f}(\prod_{i=1}^{n} M_i^{f_i} \prod_{i=1}^{n} w_i^{-t_i} M)^e U = g^{f_v}$ $y^e D = g^f, c_1^e c_2 = com(f_v, z_v)$

Figure 4.6: The protocol for Decryption Service 2. Part 2

4.2.1 SHVZK

Completeness is straight forward to verify:

$$
\begin{aligned}
&\prod_{i=1}^{n}(x_i^{-t_i})^{-f}(\prod_{i=1}^{n} M_i^{f_i} \prod_{i=1}^{n} w_i^{-t_i} M)^e U \\
=&\prod_{i=1}^{n}(x_i^{-f} w_i^e)^{-t_i} \prod_{i=1}^{n}(M_i^{f_i} M)^e U \\
=&\prod_{i=1}^{n}(x_i^{-ea_1 - d_x} w_i^e)^{-t_i} \prod_{i=1}^{n} M_i^{ef_i}(M_i^{d_i} g^{r_v})^e (g^{d_v}(\prod_{i=1}^{n} x_i^{-t_i})^{d_x} = \\
=&\prod_{i=1}^{n}(x_i^{-ea_1 - d_x} w_i^e)^{-t_i}(x_i^{d_x})^{-t_i} \prod_{i=1}^{n} M_i^{et_{\pi(i)} + ed_i}(M_i^{ed_i} g^{r_v})^e g^{d_v} = \\
=&\prod_{i=1}^{n}(x_i^{-ea_1} w_i^e)^{-t_i} \prod_{i=1}^{n} M_i^{et_{\pi(i)}} g^{er_v} g^{d_v} = \\
=&\prod_{i=1}^{n}(x_i^{-a_1} w_i)^{-et_i} \prod_{i=1}^{n} M_i^{et_{\pi(i)}} g^{er_v + d_v} = \\
=&\prod_{i=1}^{n} m_i^{-et_i} M_i^{et_{\pi(i)}} g^{er_v} g^{f_v} = g^{f_v}
\end{aligned}
$$

It works when we do not have any modular reduction of f_i's, and we will not as long as $2^{l_e} \leq (f_1, .., f_n) \leq 2^{l_e+l_s}$. There is an overwhelming probability for this with l_s sufficiently large [Gro10].

We will show the argument has the SHVZK-property. We create a simulator for the prover with several hybrid protocols between the prover and the simulator. The simulator takes e as input and has no access to the private inputs.

Simulator

$f, f_v, z_v, r_1 \leftarrow \mathbb{Z}_q$
$f_1, .., f_n \leftarrow \mathbb{Z}_q$

$c = com(0, .., 0; r)$
$c_d = com(0, ..0; r_d)$
$c_1 = com(0; r_1)$
$c_2 = com(d_v, z_v)C_1^{-e}$
$M \leftarrow \mathbb{Z}_q$
$U = g^{f_v} \prod_{i=1}^{n} (x_i^{-t_i})^f \cdot$
$\quad (\prod M_i^{f_i} \prod_{i=1}^{n} w_i^{-t_i} M)^{-e}$
$D = g^f y^{-e}$
Sim Arg(π, ρ)

Hybrid 1

$f, f_v, z_v, r_1 \leftarrow \mathbb{Z}_q$
$f_1, .., f_n \leftarrow \mathbb{Z}_q$
$d_i = f_i - t_{\pi(i)}$
$c = com(\pi(1), \ldots, \pi(n); r)$
$c_d = com(-d_i, \ldots, -d_n; r_d)$
$c_1 = com(0; r_1)$
$c_2 = com(f_v, z_v)C_1^{-e}$
$M \leftarrow \mathbb{Z}_q$
$U = g^{f_v} \prod_{i=1}^{n} (x_i^{-t_i})^f \cdot$
$\quad (\prod_{i=1}^{n} M_i^{f_i} \prod_{i=1}^{n} w_i^{-t_i} M)^{-e}$
$D = g^f y^{-e}$
Sim Arg(π, ρ)

Figure 4.7: Differences between the Simulator and Hybrid 1

The differences between the *Simulator* and *Hybrid 1* in Fig. 4.7 is in the contents of the commitments on c, c_d. These are statistically indistinguishable because of the *unconditionally hiding property* of commitments, and the simulator and hybrid 1 are therefore indistinguishable.

Hybrid 1

$f, f_v, z_v, r_v, r_d, r_1 \leftarrow \mathbb{Z}_q$
$f_1, .., f_n \leftarrow \mathbb{Z}_q$
$d_i = f_i - t_{\pi(i)}$
$c = com(\pi(1), \ldots, \pi(n); r)$
$c_d = com(-d_i, \ldots, -d_n; r_d)$
$c_1 = com(0; r_1)$
$c_2 = com(f_v, z_v) C_1^{-e}$
$M \leftarrow \mathbb{Z}_q$
$U = g^{f_v} \prod_{i=1}^{n} (x_i^{-t_i})^f \cdot$
$(\prod_{i=1}^{n} M_i^{f_i} \prod_{i=1}^{n} w_i^{-t_i} M)^{-e}$
$D = g^f y^{-e}$
Sim Arg(π, ρ)

Hybrid 2

$f, f_v, z_v, r_v, d_v, d_x, r_1 \leftarrow \mathbb{Z}_q$
$f_1, .., f_n \leftarrow \mathbb{Z}_q$
$d_i = f_i - t_{\pi(i)}$
$c = com(\pi(1), \ldots, \pi(n); r)$
$c_d = com(-d_i, \ldots, -d_n; r_d)$
$c_1 = com(0; r_1)$
$c_2 = com(f_v, z_v) C_1^{-e}$
$M = g^{rv} \prod_{i=1}^{n} M_i^{-d_i}$
$U = g^{d_v} (\prod_{i=1}^{n} x_i^{-t_i})^{d_x}$

$D = g^f y^{-e}$
Sim Arg(π, ρ)

Figure 4.8: Differences between the Hybrid 1 and Hybrid 2

In Fig. 4.8, M is picked random in *Hybrid 1*, and it is indistinguishable from the one computed in *Hybrid 2* because of the randomness of r_v. Both U's have a random factor given by g^{f_v} and g^{d_v} which make them indistinguishable. Hybrid 1 and Hybrid 2 are therefore indistinguishable.

Hybrid 2

$f, f_v, z_v, d_v, r_v, d_x, r_1 \leftarrow \mathbb{Z}_q$
$f_1, .., f_n \leftarrow \mathbb{Z}_q$
$d_i = f_i - t_{\pi(i)}$
$c = com(\pi(1), \ldots, \pi(n); r)$
$c_d = com(-d_i, \ldots, -d_n; r_d)$
$c_1 = com(0; r_1)$
$c_2 = com(f_v, z_v) C_1^{-e}$
$M = g^{rv} \prod_{i=1}^{n} M_i^{-d_i}$
$U = g^{d_v} (\prod_{i=1}^{n} x_i^{-t_i})^{d_x}$
$D = g^f y^{-e}$
Sim Arg(π, ρ)

Real prover

$f, f_v, z_v, d_v, r_v, d_x, r_1, r_2 \leftarrow \mathbb{Z}_q$
$d_1, \ldots, d_n \leftarrow \mathbb{Z}_q$
$f_i = t_{\pi(i)} + d_i$
$c = com(\pi(1), \ldots, \pi(n); r)$
$c_d = com(-d_i, \ldots, -d_n; r_d)$
$c_1 = com(r_v; r_1)$
$c_2 = com(d_v, r_2)$
$M = g^{rv} \prod_{i=1}^{n} M_i^{-d_i}$
$U = g^{d_v} (\prod_{i=1}^{n} x_i^{-t_i})^{d_x}$
$D = g^{d_x}$
Arg(π, ρ)

Figure 4.9: Differences between hybrid 2 and the real prover

The $f_i's$ are picked random in *Hybrid 2* and computed with the random $d_i's$ in the *Real prover*. They and therefore indistinguishable in Fig. 4.9. Similar argument can be used on the $d_i's$. $D = g^f y^{-e}$ is random in the choice of f and $D = g^{d_x}$ in the choice of d_x. The contents of the commitments in c_1, c_2 are hide by the

unconditional hiding property of commitments and therefore indistinguishable. The last difference is $\text{Arg}(\pi, \rho)$, which is described as a SHVZK-argument by [Gro10]. It is therefore indistinguishable from the simulated $\text{Arg}(\pi, \rho)$ in hybrid 2.

We can conclude that the simulator exists. The Witness-extended emulation follows directly from the first part of Witness-extended emulation in *Argument of Shuffle and Decryption of ElGamal Ciphertexts* in [Gro10], and we can also conclude soundness. Thus, we have a SHVZK-argument of knowledge.

4.3 Performance

We have shown that both our suggestion are *Special-honest-verifier-zero-knowledge Arguments of knowledge*, and we can therefore conclude that they are equally secure. Because of the *soundness property*, D^* has negligible chances of cheating, and this is better than the existing proof.

Since the arguments are equally secure, we will compare the two SHVZK-arguments with respect to performance, i.e the computing power they need is measured in number of exponentiations. The existing proof needs around $15n$ exponentiations, where n is the number of votes. That is $8n$ to show the permutation is correct, and $7n$ for the decryption.

The number of exponentiations in the first suggestion called Decryption service 1:

Decryption service 1

Computation:	Prover	Verifier	Total
DecryptionM_i	n		
c	$n+1$		
c_d	$n+1$		
E_d	$n+2$		
$\text{Arg}(\pi, \rho)$	$3n$	$2n$	
ξ_{id}	$2n+2$	$3n$	
Sum	$9n+6$	$5n$	$14n+6$

The number of exponentiations in the second suggestion, called Decryption service 2:

Decryption service 2

Computation:	Prover	Verifier	Total
DecryptionM_i	n		
c	$n+1$		
c_d	$n+1$		
M	$n+1$		
D	1		
c_1, c_2	4		
U	$n+1$		
$\text{Arg}(\pi, \rho)$	$3n$	$2n$	
g^{f_v}		$3n$	
$com(f_v, r_v) = c_1^e c_2$		3	
$y^e D = g^f$		2	
Sum	$8n+9$	$5n+5$	$13n+14$

The second suggestion has the best performance, and should be preferred. The advantage to the second is also the number of rounds. But as described in Sect.2.5, we can turn Interactive HVZK-arguments into Non-Interactive ZK-arguments, and the number of rounds do not matter. Compared with the existing proof, our suggestion decrease the performance by around $2n$

4.4 Auditor Corrupt

We want to argue that a corrupt auditor satisfies the goal defined in [Gjø10], that is

- The submitted ballots remain confidential

The auditor has, as the ballot box, no secret keys. We can therefore use the game hopping from Sect. 3.3.2 and we see that a corrupt auditor will not be able to extract information out of the ciphertexts. That means the only way A^* can gain information about the votes are correlating the incoming ballots D gets, with the decrypted ballots. As shown is this chapter, the decryption service shows the correctness of the permutation by a SHVZK-argument of knowledge. And by the SHVZK property, A^* can not extract information about it. It will therefore not be able to connect the decrypted ballots with the incoming, and the ballots remain confidential.

Chapter 5

Conclusion

Our goal was to make an improved Internet voting protocol compared to the existing. We removed the connection between the secret keys to avoid a corrupt ballot box and a corrupt receipt generator decrypting the votes. We also wanted to make a new and better proof for the decryption service. This is both accomplished, and we have not found any other parts with less security in the new protocol.

The protection against a corrupt receipt generator or a corrupt ballot box will still both satisfy the goals defined in [Gjø10]. The new *Special-Honest-Verifier-Zero-Knowledge Argument of knowledge* for the decryption service preserves the confidentially for the votes with a corrupt auditor, and makes the chances for a corrupt decryption service manipulating votes negligible. We have therefore in both cases made improvements with respect to security. We also looked at the performance, and found some small improvements in both cases.

However, the Internet voting protocol is still not perfect. A corrupt ballot box and receipt generator can still remove votes, even if they no longer are able to decrypt the ciphertexts and see the contents of the ballots. Our main security problem is still compromised computers. We do not have any protection against P^* and R^* cooperating. Furthermore, as described in Chap. 6 in [Gjø10], there are also many requirements for a secure election system that have not been examined.

Bibliography

[Dam10] I. Damgard. On σ-protocols. *Lecture on Cryptologic Protocol Theory*, 2010.

[DF02] I. Damgård and E. Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. *Advances in Cryptology-ASIACRYPT 2002*, pages 77–85, 2002.

[DN08] I. Damgard and J. B. Nielsen. Commitment schemes and zero-knowledge protocols (2008). 2008.

[Gjø10] K. Gjøsteen. Analysis of an internet voting protocol. Cryptology ePrint Archive, Report 2010/380, 2010.

[Gjø] K Gjøsteen. A self-study introduction to provable security.

[Gro04] J. Groth. Honest verifier zero-knowledge arguments applied. Technical report, Citeseer, 2004.

[Gro10] J. Groth. A verifiable secret shuffle of homomorphic encryptions. *Journal of cryptology*, pages 1–34, 2010.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, 1992.

[Sho04] V. Shoup. Sequences of games: a tool for taming complexity in security proofs. Technical report, Citeseer, 2004.

[Sti06] D. R. Stinson. *Cryptography: theory and practice*. CRC press, 2006.