NTNU – Trondheim
Norwegian University of
Science and Technology

A Security Analysis of the Helios Voting Protocol and Application to the Norwegian County Election

Kristine Salamonsen

Master of Science in Physics and Mathematics
Submission date: June 2014
Supervisor: Kristian Gjøsteen, MATH
Co-supervisor: Anders Smedstuen Lund, IMF

Norwegian University of Science and Technology
Department of Mathematical Sciences

Preface

This paper represents my Master's thesis and last work at NTNU. I would like to thank my supervisor Anders Smedstuen Lund. Without your help this thesis would not have been written. Thank you for teaching me the theory in an understandable way and encouraging me to do my best. I would also like to thank Kristian Gjøsteen for guiding me at my project last fall and for inspiring me to study cryptography. At last I will say thanks to Thomas for reading through my thesis, correcting errors and discussing the paper (even though you probably did not understand the half of it).

Kristine Salamonsen
Trondheim, June 2014

Abstract

We present Helios, an end-to-end verifiable internet voting system. We describe the existing protocol, the encryption and decryption process with corresponding proofs, and analyse an attack against ballot secrecy.

Further we do some changes to the existing voting protocol. In the original protocol we are not able to prove soundness and extract the witness in the proof of correct encryption. Hence, we are not able to make a formal proof of ballot secrecy. We solve this problem by adding a different proof of correct encryption. In the second change we combine the ElGamal encryption with a generalisation of the Paillier encryption and analyse whether it can be used as an efficient voting protocol for the Norwegian county election.

Sammendrag

Vi presenterer Helios, et ende-til-ende verifiserbart internett valgsystem. Vi beskriver protokollen, krypteringen og dekrypteringen med tilhørende bevis for korrekthet. Vi analyserer også et angrep mot protokollen og ser på hvilke forbedringer som er gjort i denne sammenheng.

Videre utfører vi to endringer i den eksisterende valgprotokollen. I den originale protokollen klarer vi ikke å lage et formelt sikkerhetsbevis som viser at det man stemmer er hemmelig. Vi løser dette ved å legge til et ekstra krypteringsbevis i tillegg til det vi kaller et proof of knowledge. Til slutt kombinerer vi ElGamalkrypteringen med en generalisering av Pailliers kryptering og analyserer om det er en effektiv valgprotokoll for det norske fylkestingsvalget.

Contents

Chapter 1

Introduction

Electronic voting is upcoming and increases the flexibility of voting. It gives the opportunity of either remote voting where voters may submit their votes from home or voting electronically via computers in a voting booth at polling stations. In this thesis I have studied an example of a protocol for use in remote voting, namely Helios.

Along with the opportunity of remote voting over the internet follows a lot of challenges. In traditional paper elections voters do their choices secretly in a voting booth, making sure no one is looking. They submit their ballots into a locked ballot box, and make sure no one else is able to tamper with their ballots. Turning this into a digital process makes it all more complicated, introducing the need for techniques to guarantee privacy and integrity. The computations in the encryption and decryption process might also be tedious.

As mentioned, my work is based upon Helios, a publicly available internet voting system from 2008. This thesis is a continuation of my project the last fall, in which I introduced the existing Helios voting protocol. Now I dig a bit deeper and I make some changes to the protocol. These changes enable us to prove that any vote given in an election is kept secret, given some reasonable assumptions.

Helios supports homomorphic encryption allowing us to jointly decrypt the ciphertexts to reveal the sum of the votes. This could be very efficient, especially for simple elections, but for Helios where we use the exponential version of ElGamal this becomes hard when the exponent tends to grow large, since we have to compute the discrete logarithm.

We want to find a reasonably efficient voting protocol with homomorphic encryption for the Norwegian county election, and as Helios is today, it will not be efficient. ElGamal has some nice properties which we want to keep. Hence, we combine the ElGamal encryption with a generalisation of Paillier's public key system.

Overview of the Paper

Chapter 2 gives the theoretical background for this thesis. We discuss zero knowledge, Σ-protocols and proofs of knowledge.

Chapter 3 describes the voting protocol and the different players in it. We describe the voting phase with encryption and the counting phase with decryption, where we apply theory from the previous chapter.

Chapter 4 analyses an attack against the voting protocol. We will see how this effected the voter privacy and explain how the problem was solved. We will also propose another possible solution with unique identifiers.

In Chapter 5 we introduce a change to the voting protocol. We make a security analysis to make sure the new protocol still fulfills verifiability and we are now able to prove ballot secrecy.

In the last chapter we combine the exponential ElGamal cryptosystem with a generalisation of Paillier's public key system. The last thing we do is to apply Helios with the new encryption on the Norwegian county election and analyse the efficiency.

Chapter 2

Theory

This chapter will introduce the theory behind this thesis and will be useful in the work we do. We will first present the Decision Diffie-Hellman problem on which the security of the Helios voting protocol relies. Secondly we introduce the ElGamal Cryptosystem, which we use to encrypt and decrypt ballots. Then we look at zero-knowledge, Σ-protocols and proofs of knowledge, which we will use to show correct encryption and decryption in Helios. Lastly we present the Fiat-Shamir heuristic which makes our interactive proofs non-interactive.

2.1 The Decision Diffie-Hellman Problem

The security of our protocol relies on the complexity of the Decision Diffie-Hellman (DDH) problem. Consider a cyclic group G of prime order q and let g be a generator of the group. Generally speaking the DDH assumption claims that there exists no algorithm which efficiently distinguishes the distribution $\{g, g^a, g^b, g^{ab}\}$ from the distribution $\{g, g^a, g^b, g^c\}$ for random $a, b, c \in G$. More formally, we have:

Decision Diffie-Hellman Problem. *Given $(g_0, g_1) \in G \times G$ (where at least g_1 is sampled at random), decide if $(u_0, u_1) \in G \times G$ was sampled uniformly from the set $\{(g_0^t, g_1^t) \mid 0 \leq t < q\}$ or uniformly from $G \times G$.*

It can be proven (e.g [13], [10]) that the DDH problem is equivalent to the following problem:

***L*-DDH.** *Given $(g_0, g_1, \ldots, g_L) \in G^{L+1}$ (where at least $g_1, g_2, \ldots, g_L$ are sampled at random), decide if $(u_0, u_1, \ldots, u_L) \in G^{L+1}$ was sampled uniformly from the set $\{(g_0^t, g_1^t, \ldots, g_L^t) \mid 0 \leq t < q\}$ or uniformly from G^{L+1}.*

2.2 Encryption and Decryption

The encryption of ballots in Helios is based on the ElGamal encryption system. This section will specify the key generation, encryption and decryption for the exponential ElGamal Cryptosystem. We will also see some of the advantages of the additive homomorphic properties of this version of ElGamal.

Let p and q be primes such that $q \mid p-1$. Let G be a finite cyclic subgroup of $\mathbb{Z}_p^*$ with order q, and let g be a generator for the group, $G = \langle g \rangle$. The key generation is as follows:

- Select a at random, $a \in_R \mathbb{Z}_q^*$, and compute $y \leftarrow g^a (mod\ p)$. The private key is a while the public key is y, p and q.

Let $m \in \mathbb{Z}_q$, then we have the encryption of m:

- Select r at random, $r \in_R \mathbb{Z}_q^*$.
- A ciphertext is computed as $c = (\alpha, \beta) = (g^r\ (mod\ p),\ y^r g^m (mod\ p))$.

  The ciphertext is decrypted using the secret key, and we obtain
- $g^m = \beta y^{-r} = \beta (g^a)^{-r} = \beta \alpha^{-a}\ (mod\ p)$.

We choose the exponential version of ElGamal such that we get the plaintext m in the exponent, then we achieve an additive homomorphic cryptosystem, and we will exploit this further.

There is two common ways to anonymise the ballots; by homomorphic encryption or by a mix-net. By the homomorphic property of ElGamal we have that

$$Enc(m_1) \cdot Enc(m_2) = Enc(m_1 + m_2).$$

This allows us to multiply all the ballots together and then jointly decrypt them, revealing no single ballot. This explains why the exponential version of ElGamal is desirable. We obtain:

$$\prod c_i = \left(\prod g^{r_i}, \prod y^{r_i} g^{m_i} \right) = \left(g^{\sum r_i}\ (mod\ p), y^{\sum r_i} g^{\sum m_i}\ (mod\ p) \right).$$

Remark that we have to compute the discrete logarithm to obtain the final result $\sum m_i$, a problem assumed to be hard in our group G. This is indeed possible for small values of $\sum m_i$, by raising g to higher and higher powers until it equals $\beta \alpha^{-a}\ (mod\ p)$. But for large exponents this will be tricky.

The other way the ballots are made anonymous is by a mix-net. The mix-net shuffles and re-encrypts the encrypted votes and thus destroys the link between a voter and the corresponding vote. The shuffling is based on the Sako-Kilian protocol [14], a provable mix-net based on ElGamal re-encryption. The mix-net

was used in some previous versions of Helios, but this thesis will consider the homomorphic encryption and we will not describe the mix-net any further.

We remark that all calculations further in the chapter are done modulo p, we just omit the notation for simplicity and to save space.

2.3 Correct Encryption

Given a ciphertext (α, β) we want to prove that it contains a value between 0 and $q-1$ without revealing the value. This is done by a disjunctive proof of equality between two discrete logarithms. We prove that $(log_g\alpha = log_y\beta) \vee (log_g\alpha = log_y\frac{\beta}{g}) \vee \ldots \vee (log_g\alpha = log_y(\frac{\beta}{g^{q-1}}))$.

We will use a Σ-protocol, so we will first introduce zero-knowledge proofs, Σ-protocols and proofs of knowledge.

2.3.1 Zero-Knowledge, Σ-Protocols and Proofs of Knowledge

We consider a protocol between two parties, which we call prover and verifier. The goal is for the prover to show knowledge of some secret or convince the verifier of the validity of a statement without revealing the secret or anything beyond the validity of the statement. The idea is that the verifier should gain no (zero) additional knowledge than what he had before the execution of the protocol. We will now introduce a Σ-protocol, which is a zero-knowledge protocol.

A Σ-protocol is a 3-round protocol defined for a relation R. In a 3 round protocol the prover makes the first move and sends a message/commitment to the verifier. The verifier replies with a challenge and the last move is the prover's response.

Let $R = (v; w) \subseteq V \times W$ be a binary relation. $v \in V$ is the common input for the prover and the verifier, and $w \in W$ is the private input for the prover. We call w the witness. We define a language for the relation R: $L_R = \{v \in V : \exists_{w \in W}(v, w) \in R\}$. The definition of a Σ-protocol is given in [4], and we give it here:

Definition 1. *A protocol is said to be a Σ-protocol for a relation R if it satisfies the following properties:*

- ***Completeness.*** *In a conversation between an honest prover and an honest verifier and if $(v, w) \in R$, the verifier will always accept.*
- ***Special soundness.*** *Given the common input v and two accepting conversations (A, e, t) and (A, e', t') (with the same commitment but different*

challenges), a probabilistic polynomial time algorithm should be able to compute a witness w such that $(v, w) \in R$.

- ***Special honest-verifier zero-knowledge.*** *There exists a probabilistic polynomial time algorithm which, on common input (any given v and any challenge e), simulates an accepting conversation indistinguishable from a real conversation between an honest prover and an honest verifier.*

The proof of equality between two discrete logarithms which we use to prove correct encryption is a proof of knowledge, demonstrating knowledge of the randomness r used to create the ciphertext. Generally speaking, the protocol is a proof of knowledge if the prover convinces the verifier that he knows some information which is related to the common input v. In our case the prover knows the discrete logarithm r.

Technically a Σ-protocol is a proof of knowledge if there exists a probabilistic polynomial time algorithm that is able to extract the witness by interacting with a prover, with the same success probability as a prover convincing a real verifier. We call this algorithm a knowledge extractor. The extractor may rewind the prover to the point after the prover sent it commitments, send it different messages and play it several times, until it get two accepting conversation. From the requirement of special soundness there exists an algorithm which now is able to compute the witness.

2.3.1.1 OR-Protocol with n Options

Let us go back to what we wanted to prove, namely that (α, β) contains a value between 0 and $q-1$ without revealing the value. We use an OR-protocol which is a Σ-protocol for the relation $R_1 \vee R_2 = \{(v_1, v_2; w_1, w_2) : (v_1; w_1) \in R_1 \vee (v_2; w_2) \in R_2\}$. We extend this to an OR-protocol with n options and prove that $(log_g\alpha = log_y\beta) \vee (log_g\alpha = log_y\frac{\beta}{g}) \vee \ldots \vee (log_g\alpha = log_y\frac{\beta}{g^{n-1}})$. We call this protocol a n-OR-protocol and it is presented in Figure 2.1.

The common input is g, y, α and $\frac{\beta}{g^m}$, while the private input to the prover is r.

We show that the n-OR-protocol is a Σ-protocol.

Claim 1. *The n-OR-protocol is a Σ-protocol for the relation $\{((\alpha, \beta); 0, 1, \ldots, n-1) : (log_g\alpha = log_y\beta) \vee (log_g\alpha = log_y\frac{\beta}{g}) \vee \ldots \vee (log_g\alpha = log_y\frac{\beta}{g^{n-1}})\}$.*

Proof. • Completeness: Consider the case where $(\alpha, \beta) = (g^r, y^r g^i)$, $0 < i < n-1$, then:

$$g^{t_i} = g^{x_i + re_i} = A_i\alpha^{e_i},$$
$$y^{t_i} = y^{x_i + re_i} = B_i(\frac{\beta}{g^i})^{e_i}.$$

Clearly for $j \neq i$ we have $g^{t_j} = A\alpha^{e_j}, y^{t_j} = B_j(\frac{\beta}{g^j})^{e_j}$ by construction, and completeness holds.

Prover **Verifier**

Private input: r.
Assume:$(\alpha, \beta) = (g^r, y^r g^i)$,
$i \in \{0, 1, \ldots, n-1\}$.
$x_i \in_R \mathbb{Z}_q^*$.
$e_0, e_1, \ldots, e_{i-1}, e_{i+1}, \ldots, e_{n-1} \in_R \mathbb{Z}_q^*$.
$t_0, t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_{n-1} \in_R \mathbb{Z}_q^*$.
$A_0 \leftarrow g^{t_0} \alpha^{-e_0}$
$B_0 \leftarrow y^{t_0} \beta^{-e_0}$
$A_1 \leftarrow g^{t_1} \alpha^{-e_1}$
$B_1 \leftarrow y^{t_1} (\frac{\beta}{g})^{-e_1}$
$\vdots$
$A_{i-1} \leftarrow g^{t_{i-1}} \alpha^{-e_{i-1}}$
$B_{i-1} \leftarrow y^{t_{i-1}} (\frac{\beta}{g^{i-1}})^{-e_{i-1}}$
$A_i \leftarrow g^{x_i}$
$B_i \leftarrow y^{x_i}$
$A_{i+1} \leftarrow g^{t_{i+1}} \alpha^{-e_{i+1}}$
$B_{i+1} \leftarrow y^{t_{i+1}} (\frac{\beta}{g^{i+1}})^{-e_{i+1}}$
$\vdots$
$A_{n-1} \leftarrow g^{t_{n-1}} \alpha^{-e_{n-1}}$
$B_{n-1} \leftarrow y^{t_{n-1}} (\frac{\beta}{g^{n-1}})^{-e_{n-1}}$

$A_0, B_0, A_1, B_1, \ldots, A_{n-1}, B_{n-1} \longrightarrow$

$e \in_R \mathbb{Z}_q^*$

$\longleftarrow e$

$e_i \leftarrow e - e_0 - \ldots - e_{i-1} - e_{i+1} - \ldots - e_{n-1}$
$t_i \leftarrow x_i + r e_i$

$e_0, t_0, e_1, t_1, \ldots, e_{n-1}, t_{n-1} \longrightarrow$

Verify that:
$e \stackrel{?}{=} e_0 + e_1 + \ldots + e_{n-1}$
For all $i \in \{0, 1, \ldots, n-1\}$
check that:
$g^{t_i} \stackrel{?}{=} A_i \alpha^{e_i}$
$y^{t_i} \stackrel{?}{=} B_i (\frac{\beta}{g^i})^{e_i}$

Figure 2.1: Protocol n-OR.

- Special soundness: Given two accepting conversations

$$(A_0, B_0, A_1, B_1, \ldots, A_{n-1}, B_{n-1}, e; e_0, e_1, \ldots, e_{n-1}, t_0, t_1, \ldots, t_{n-1}),$$
$$(A_0, B_0, A_1, B_1, \ldots, A_{n-1}, B_{n-1}, e'; e'_0, e'_1, \ldots, e'_{n-1}, t'_0, t'_1, \ldots, t'_{n-1}),$$

where the commitments are the same and the challenges are different. We should now be able to compute the witness to prove special soundness.

Since $e = e_0 + e_1 + \ldots + e_{n-1} \neq e'_0 + e'_1 + \ldots + e'_{n-1} = e'$, we have $e_0 \neq e'_0$ or $e_1 \neq e'_1$ or $\ldots$ $e_{n-1} \neq e'_{n-1}$. We also have the following:

$$\begin{array}{llll}
g^{t_0}=A_0\alpha^{e_0}, & g^{t'_0}=A_0\alpha^{e'_0}, & y^{t_0}=B_0\beta^{e_0}, & y^{t'_0}=B_0\beta^{e'_0}, \\
g^{t_1}=A_1\alpha^{e_1}, & g^{t'_1}=A_1\alpha^{e'_1}, & y^{t_1}=B_1\left(\frac{\beta}{g}\right)^{e_1}, & y^{t'_1}=B_1\left(\frac{\beta}{g}\right)^{e'_1}, \\
\vdots & & & \\
g^{t_{n-1}}=A_{n-1}\alpha^{e_{n-1}}, & g^{t'_{n-1}}=A_{n-1}\alpha^{e'_{n-1}}, & y^{t_{n-1}}=B_{n-1}\left(\frac{\beta}{g^{n-1}}\right)^{e_{n-1}}, & y^{t'_{n-1}}=B_{n-1}\left(\frac{\beta}{g^{n-1}}\right)^{e'_{n-1}}.
\end{array}$$

From this follows:

$$\begin{array}{ll}
g^{t_0-t'_0} = \alpha^{e_0-e'_0}, & y^{t_0-t'_0} = \beta^{e_0-e'_0} \\
g^{t_1-t'_1} = \alpha^{e_1-e'_1}, & y^{t_1-t'_1} = \left(\frac{\beta}{g}\right)^{e_1-e'_1} \\
\vdots & \\
g^{t_{n-1}-t'_{n-1}} = \alpha^{e_{n-1}-e'_{n-1}}, & y^{t_{n-1}-t'_{n-1}} = \left(\frac{\beta}{g^{n-1}}\right)^{e_{n-1}-e'_{n-1}},
\end{array}$$

and we can extract the witness. Let $i \in \{0, 1, \ldots, n-1\}$ such that $e_i \neq e'_i$. Then we obtain the witness $r_i = \frac{t_i - t'_i}{e_i - e'_i}$, and we see that $r_i = log_g \alpha = log_y \frac{\beta}{g^i}$.

- Special honest-verifier zero-knowledge: We will again consider the case where $(\alpha, \beta) = (g^r, y^r g^i), i \in \{0, 1 \ldots n-1\}$. We have the distribution from the honest-verifier for any given challenge e:

$$\begin{array}{l}
\{(A_0, B_0, A_1, B_1, \ldots, A_{n-1}, B_{n-1}, e; e_0, e_1, \ldots, e_{n-1}, t_0, t_1, \ldots, t_{n-1}) : \\
x_i, e_0, e_1, \ldots, e_{i-1}, e_{i+1}, \ldots, e_{n-1}, t_0, t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_{n-1} \in_R \mathbb{Z}_q^*, \\
e_i \leftarrow e - e_1 - \ldots - e_{i-1} - e_{i+1} - \ldots - e_{n-1}, \\
t_i \leftarrow x_i + re_i, \\
A_0 \leftarrow g^{t_0}\alpha^{-e_0}, B_0 \leftarrow y^{t_0}\beta^{-e_0}, \\
A_1 \leftarrow g^{t_1}\alpha^{-e_1}, B_1 \leftarrow y^{t_1}\left(\frac{\beta}{g}\right)^{-e_1}, \\
\vdots \\
A_i \leftarrow g^{x_i}, B_i \leftarrow y^{x_i}, \\
\vdots \\
A_{n-1} \leftarrow g^{t_{n-1}}\alpha^{-e_{n-1}}, B_{n-1} \leftarrow y^{t_{n-1}}\left(\frac{\beta}{g^{n-1}}\right)^{-e_{n-1}}\},
\end{array}$$

and the simulated distribution:

$$\begin{aligned}
&\{(A_0, B_0, A_1, B_1, \ldots, A_{n-1}, B_{n-1}, e; e_0, e_1, \ldots, e_{n-1}, t_0, t_1, \ldots, t_{n-1}) : \\
&e_0, e_1, \ldots, e_{i-1}, e_{i+1}, \ldots, e_{n-1}, t_0, t_1, \ldots, t_{n-1} \in_R \mathbb{Z}_q^*, \\
&e_i \leftarrow e - e_1 - \ldots - e_{i-1} - e_{i+1} - \ldots - e_{n-1}, \\
&A_0 \leftarrow g^{t_0}\alpha^{-e_0}, B_0 \leftarrow y^{t_0}\beta^{-e_0}, \\
&A_1 \leftarrow g^{t_1}\alpha^{-e_1}, B_1 \leftarrow y^{t_1}\left(\frac{\beta}{g}\right)^{-e_1}, \\
&\vdots \\
&A_i \leftarrow g^{t_i}\alpha^{-e_i}, B_i \leftarrow y^{t_i}\left(\frac{\beta}{g^i}\right)^{-e_i}, \\
&\vdots \\
&A_{n-1} \leftarrow g^{t_{n-1}}\alpha^{-e_{n-1}}, B_{n-1} \leftarrow y^{t_{n-1}}\left(\frac{\beta}{g^{n-1}}\right)^{-e_{n-1}}\}.
\end{aligned}$$

It can be verified that these distributions are identical, and hence, we have special honest-verifier zero-knowledge, and the n-OR-protocol is a Σ-protocol. □

2.4 Correct Decryption

Proving the decryption correct is almost the same as proving correct encryption. But instead of proving knowledge of the randomness r used to create the ciphertext, we now want to prove knowledge of the secret key a. For integrity we prove equality between two discrete logarithms, namely $log_g y = log_\alpha \frac{\beta}{g^m}$. Hence, we see that this protocol is simpler than the OR-protcol used for encryption. The equality protocol is given in Figure 2.2.

The common input is g, y, α and $\frac{\beta}{g^m}$, while the private input to the prover is a. We show that this equality protocol is a Σ-protocol as well.

Claim 1. *The equality protocol is a Σ-protocol for the relation $\{(g, y, \alpha, \frac{\beta}{g^m}) : log_g y = log_\alpha \frac{\beta}{g^m}\}$.*

Proof.

- Completeness: Completeness follows from:

$$\begin{aligned}
g^t &= g^{x+ae} = Ay^e, \\
\alpha^t &= \alpha^{x+ae} = B\left(\frac{\beta}{g^m}\right)^e.
\end{aligned}$$

- Special soundness: Let (A, B, e, t) and (A, B, e', t') be two accepting transcripts with the same commitments A and B, and different challenges $e \neq e'$. We have

$$\begin{aligned}
g^t &= Ay^e, & g^{t'} &= Ay^{e'} \\
\alpha^t &= B\left(\frac{\beta}{g^m}\right)^e, & \alpha^{t'} &= B\left(\frac{\beta}{g^m}\right)^{e'}.
\end{aligned}$$

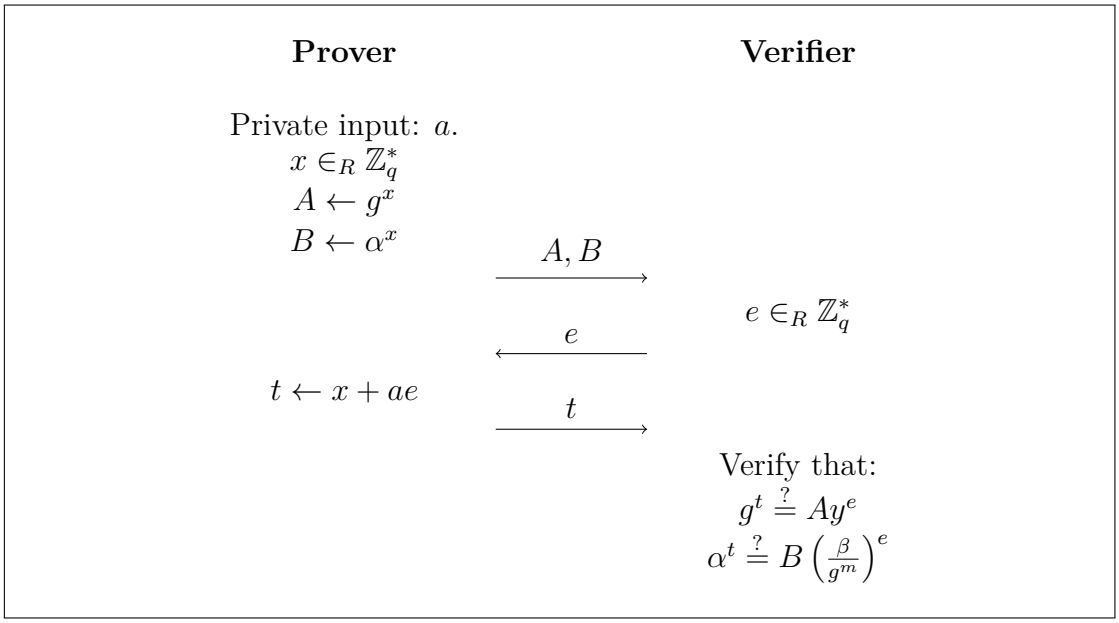

Figure 2.2: Equality Protocol.

From this follows:

$$g^{t-t'} = y^{e-e'}$$
$$\alpha^{t-t'} = \left(\frac{\beta}{g^m}\right)^{e-e'},$$

and we can extract the witness $a = \frac{t-t'}{e-e'}$. We have $a = log_g y$ and $a = log_\alpha \frac{\beta}{g^m}$ and it follows that $log_g y = log_\alpha \frac{\beta}{g^m}$.

- Special honest-verifier zero-knowledge: The distribution from the honest-verifier for any given challenge e:

$$\{(A, B, e, t) : x \in_R \mathbb{Z}_q^*, A \leftarrow g^x, B \leftarrow \alpha^x; t \leftarrow x + ae\},$$

and the simulated distribution:

$$\{(A, B, e, t) : t \in_R \mathbb{Z}_q^*, A \leftarrow g^t y^{-e}, B \leftarrow \alpha^t \left(\frac{\beta}{g^m}\right)^{-e}\}.$$

It can be checked that these distributions are identical, and we have special honest-verifier zero-knowledge, and the equality protocol is a Σ-protocol.

□

Remark that we only are able to prove zero-knowledge against an honest verifier, but that suffices for our purpose.

2.5 More on Soundness

We now argue that the relation $log_g y = log_\alpha \frac{\beta}{g^m}$ holds in the equality protocol given one accepting conversation, though we are not able to extract the witness.

Let (A, B, e, t) be a conversation such that $g^t = Ay^e$ and $\alpha^t = B(\frac{\beta}{g^m})^e$. g is a generator for the group, so we can write $g^t = g^\rho(g^\nu)^e$ for some ρ and $\nu \in \mathbb{Z}_q^*$. Similarly, $\alpha = g^r$ is a generator and we have that $\alpha^t = \alpha^{\rho+\delta}(\alpha^{\nu+\Delta})^e$. From this follows that $t = \rho + \nu e$ and $t = \rho + \delta + (\nu + \Delta)e$. Subtracting the first one from the second one gives $0 = \delta + \Delta e$. Since e is sampled after δ and Δ are chosen we have that $\delta = \Delta = 0$. This means that $log_g A = log_\alpha B$ and $log_g y = log_\alpha \frac{\beta}{g^m}$. Hence, given one accepting conversation the relation $log_g y = log_\alpha \frac{\beta}{g^i}$ always holds except with probability $(q-1)^{-1}$.

The same holds for the n-OR-protocol. Since A_i, B_i are defined before we know e, at least one e_i must be undefined. This gives us the same argument for $log_g \alpha = log_y \frac{\beta}{g^i}$. Looking at one specific i in the n-OR-protocol is just the same as one equality protocol. Hence we have that the relation $log_g \alpha = log_y \frac{\beta}{g^m}$ always holds with just one accepting conversation except with probability $(q-1)^{-1}$.

2.6 The Fiat-Shamir Heuristic

We use the Fiat-Shamir heuristic [7] to obtain a non-interactive zero knowledge proof. This is done by replacing the response of the verifier by a hash function H modelled as a random oracle. A random oracle could be viewed as a theoretical black box returning completely random answers for every query. It is also consistent, always returning the same answer on a given input. Consider every player to have access to this oracle. For more information about random oracles see [2].

As an example we look at the equality protocol in Figure 2.2. From the 3-round communication flow in the Σ-protocol, we now get a 1-round protocol. It works as follows for the decryption proof:

- The prover does the same as before, but instead of sending the commitments A, B to the verifier, he computes $e \leftarrow H(g, \alpha, y, \beta, A, B)$.
- The prover continues as before and computes $t \leftarrow x + ae$, and sends A, B, e, t to the verifier.
- At last the verifier checks if $e = H(g, \alpha, y, \beta, A, B)$ and if $g^t = Ay^e$ and $\alpha^t = B(\frac{\beta}{g^m})^e$.

The protocol is still complete and special honest-verifier zero-knowledge. The simulator works just as before; for any e choose t uniformly at random, and com-

pute $A \leftarrow g^t y^{-e}$ and $B \leftarrow \alpha^t (\frac{\beta}{g^m})^{-e}$. Then we can just reprogram the random oracle such that

$$H(g, \alpha, y, \beta, A, B) \leftarrow e.$$

We will further in this thesis use that H_{1i} is a hash function for an i-OR proof, that is $H_{1i} : G^{2i+4} \rightarrow \mathbb{Z}_q^*$ by $e \leftarrow H_{1i}(g, y, \alpha, \beta, A_0, B_0, A_1, B_1, \ldots, A_i, B_i)$. Further we will use that H_2 is a hash function for the equality protocol, that is $H_2 : G^6 \rightarrow \mathbb{Z}_q^*$ by $e \leftarrow H_2(g, \alpha, y, \beta, A, B)$.

The soundness on the other hand is more difficult to prove. In the non-interactive case we are no longer able to extract the witness. We can not longer rewind and obtain two accepting conversations with the same commitment but different challenges, since a random oracle will always return the same challenge on the same commitment query. There is no known attack against it, we just do not know how to prove it. In Chapter 5 we will add an extra proof to the knowledge proof and thus fix this problem, because we no longer need to extract the witness.

2.7 The Probability of Faking Decryption Proofs

In this section we take a look at the probability of faking the decryption proofs. We will need a bound for this event when we prove ballot secrecy later. We prove the following theorem.

Theorem 2.1. *Take any algorithm that outputs integers ν, e, and group elements g, α, y and $\frac{\beta}{g^m}$ such that $log_g y \neq log_\alpha \frac{\beta}{g^m}$. If the algorithm uses at most η queries to the random oracle H_2, then we have that the probability that*

$$A = g^t y^{-e},\ B = \alpha^t \left(\frac{\beta}{g^m}\right)^{-e} \text{ and } e = H(g, \alpha, y, \beta, A, B)$$

is at most $(\eta + 1)(q - 1)^{-1}$.

We use the following lemma from [9] to prove the theorem.

Lemma 2.2. *Let G be a group of prime order q, and let g be a generator for the group. Let ν and Δ be integers, and A, B, x_1, x_2 are group elements such that $x_1 = g^\nu$ and $x_2 = \alpha^{\nu+\Delta}$.*

Then if $\Delta \neq 0$ and e is an integer chosen uniformly at random from a set with $q - 1$ elements, the probability that there exists an integer t such that

$$Ax_1^e = g^t \text{ and } Bx_2^e = \alpha^t$$

is at most $(q - 1)^{-1}$.

Proof. We have $x_1 = g^\nu$ and $x_2 = \alpha^{\nu+\Delta}$. Suppose $\Delta \neq 0$, that is $log_g y \neq log_\alpha \frac{\beta}{g^m}$. Let e be chosen at random from $\mathbb{Z}_q^*$.

Since both g and α are generators of the group we have that $A = g^\rho$ and $B = \alpha^{\rho+\delta}$ for some ρ and $\delta \in \mathbb{Z}*_q$. We want to find the probability that there exists a t, given $A, B, g, \alpha, x_1, x_2$, such that

$$Ax_1^e = g^t \text{ and } Bx_2^e = \alpha^t.$$

We get $g^\rho g^{\nu e} = g^t$ and $\alpha^{\rho+\delta}\alpha^{(\nu+\Delta)e} = \alpha^t$. This gives $t = \rho+\nu e$ and $t = \rho+\delta+(\nu+\Delta)e$. Subtracting the first one from the second gives $\delta + \Delta e = 0$. Since $\Delta \neq 0$ and since δ and Δ was fixed before we choose e, there is only one e which satisfies this equation. e is chosen randomly from $\mathbb{Z}_q^*$, and from this follows that the probability of finding t such that the above holds is $(q-1)^{-1}$. □

Proof. (Proof of Theorem 2.1). If H_2 has not been queried at the relevant point, we have that $A = g^t y^{-e}$, $B = \alpha^t \left(\frac{\beta}{g^m}\right)^{-e}$ and $e = H_2(g, \alpha, y, \beta, A, B)$ holds with probability $(q-1)^{-1}$. From Lemma 2.2 we have that every time the algorithm queries H_2 with some input for which he cannot already create a forged proof, the probability that the random oracle replies with something that the adversary is able to use to forge a proof is $(q-1)^{-1}$. This is done η times.

This adds up to $(1+\eta)$ times where each event has probability $(q-1)^{-1}$. Let $\delta = (q-1)^{-1}$. Now we have that the probability that at least one of them happens is bounded by $1-(1-\delta)^{1+\eta} \leq 2(\eta+1)\delta$. □

Chapter 3

The Voting Protocol

The voting protocol consists of several cryptographic operations and different entities which we call players. In this chapter we will present the players and their interaction with each other. We will divide the election into three phases: Setup phase, voting phase and counting phase, and we describe the roles of the players in the different phases.

Helios has developed through four versions, and some of the improvements are done due to attacks discovered against the voting system. We will briefly mention some of these changes in this chapter, but the important improvements done to the protocol are discussed further in Chapter 4.

3.1 The Different Players

The voting protocol consists of several players interacting with one or several other players. The players we will consider in this protocol are the voter V, the voter's computer P, the election officer E, the bulletin board B and the decryptor D. Figure 3.1 presents the players and the communication channels between them.

The protocol works as follows: V submits a vote to P, which encrypts the vote and sends it further to E along with proofs of correct encryption. E verifies the proofs and if they are correct he publishes the encrypted vote and corresponding proofs on B. B is a publicly available website. When the election is done, E can compute the encrypted tally. All the ciphertexts are then homomorphically combined, and E sends the combination to D. D decrypts the ciphertexts, sends the result back to E along with a proof of correct decryption, and if the proof are correct, E publishes the final result on B with corresponding correction proofs.

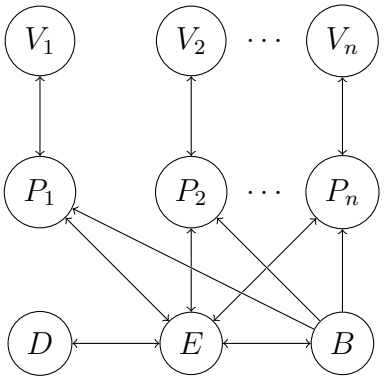

Figure 3.1: Overview of the Protocol Players.

3.2 Setup Phase

Anyone is able to create an election with Helios and invite people to vote. The person creating the election is considered as the administrator. He specifies the eligible voters, the questions or the candidates for the election, the opening and closing time (the voting phase), and controls E.

Helios generates a pair of keys, one public and one private. A set of n trustees are also selected. Helios supports threshold decryption, which means that the private key is shared among several trustees. Each trustee gets its own private key share a_i which is randomly chosen, $a_i \in_R \mathbb{Z}_q^*$. The private key is then the sum of all the key shares, $a = a_0 + a_1 + \ldots + a_{n-1}$.

Each trustee computes $y_i \leftarrow g^{a_i}$. Then $y \leftarrow y_0 \cdot y_1 \cdot \ldots \cdot y_n$ is the public key that is used for encryption and is publicly available at the bulletin board. This way, each trustee is committed to their key share by the value y_i. By a proof of knowledge as in Figure 2.2 (without showing equality) with a_i as the private input and only one commitment g^x, each trustee shows that y_i was correctly constructed. As we will see all the trustees are required for the decryption process.

The administrator can choose to have only one trustee as well, the Helios server itself.

When it is time, the administrator freezes the election. The voter list is now finished and the questions/candidates are unchangeable and ready for downloading.

3.3 Voting Phase

When the administrator freezes the election, the eligible voters receive an e-mail with information that the election is open and the questions/candidates are avail-

able for download. The mail also includes an individual username, a randomly generated password, a hash of the election parameters and the URL for the voting booth.

A voter V is first met by the ballot preparation. At this step anyone, not only eligible voters, is able to look and choose alternatives. Once every choice is made, they are immediately encrypted. A ballot contains a list of encrypted answers, where each encrypted answer represents one election question. Each encrypted answer has again a list of n ciphertexts $((\alpha_1, \beta_1), (\alpha_2, \beta_2), \ldots, (\alpha_n, \beta_n))$, where n is the number of possible choices or candidates for that question. Each encrypted answer contains also the product of the n ciphertexts $(\overline{\alpha}, \overline{\beta}) = (\alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_n, \beta_1 \cdot \beta_2 \cdot \ldots \cdot \beta_n)$.

The ciphertexts are constructed as described in Section 2.2. Let $m \in \{0, 1\}$ and let $(\alpha, \beta) = (g^r, y^r g^m)$ represents a ciphertext for some candidate. Then $m = 1$ means that the voter voted for the candidate, and $m = 0$ means that V did not vote for the candidate.

The product of ciphertexts $(\overline{\alpha}, \overline{\beta})$ for each question will contain a value between 0 and $n - 1$. In some cases the voter may be restricted to vote for at most one candidate, then the product of ciphertexts will contain a 0 or a 1.

When every plaintext is encrypted, a hash of the encryption appears and V can choose to audit or cast the ballot. The auditing reveals the ciphertexts for each possible answer along with proofs of correct encryption as described in Section 2.3.1.1. The proofs include individual proofs that each ciphertext contains a 0 or a 1 and overall proofs that the product of ciphertexts for each question contains a value between 0 and $n - 1$.

The auditing allows every voter to verify the encryption, though, some mathematical knowledge is required to do so. V should not be able to know the randomness used to encrypt the ballot, so he is asked to reconstruct the ballot. A new r is randomly generated, and hence, a new encryption is made. He can choose to audit as many times as he wants.

The voter is not asked to authenticate before he chooses to cast his ballot. This way, anyone is able to check the ballot preparation, audit the encryption and check that a ballot is well formed. This feature, separating the ballot preparation and casting, is based upon Benaloh's principle of ballot casting assurance [3], and it is important for the end-to-end verifiability Helios offers.

The voter authenticates by typing the username and the password received in the e-mail. The ciphertext is cast and the records of it are then deleted from the voter's computer. The encrypted ballot is published on the bulletin board along with V's ID or some alias, and proofs of correct encryption. Hence, V and everyone else can verify that V's ballot is included and that he cast a valid vote. V also receives a confirmation on e-mail, with the encrypted vote and the hash.

3.4 Counting Phase

When the election is closed and the voters have cast their votes, the cast votes are stored at the bulletin board. It is time for decryption and counting. To preserve anonymity, a ballot should never be decrypted alone. By the homomorphic properties of ElGamal described in Section 2.2, the ballots are multiplied together and then jointly decrypted.

In the case of several trustees, they are all required for decryption, where they use their private key share for a partial decryption. The partial decryption factor is $k_i = \alpha^{a_i}$, given (α, β). We stress that we use the inverse of the decryption factors in the decryption. Let k denote the product of all decryption factors. Then we have $g^m = \beta k^{-1}$.

The trusteees also need to show that the decryption is correct. This is done by proving equality between two discrete logarithms. Since every trustee generates their own proof, they prove that $log_g y_i = log_\alpha k_i$ by using the Σ-protocol for equality from Section 2.4.

When the homomorphic tallying is done the final result is made public at the bulletin board, along with proofs of correctness.

3.5 Development

The first version of Helios was published in 2008 and has by now developed to the fourth version. In 2010 Helios was attacked and major improvements were made due to this attack. The proofs were made more robust after the attack by including more parameters in the hash which generates the challenge in the knowledge proof for correct encryption. We will discuss the attack and the improvements further in Chapter 4.

Among other changes was the transition from mix-nets in the first version to homomorphic encryption in the second version. In the third version it became possible to authenticate in several new ways, e.g. Google, Facebook and Twitter.

We remark that the documentation of the fourth version is still in progress, as it has been since August 2012 [1].

Chapter 4

An Attack Against Ballot Secrecy

Helios has suffered from three known attacks. We will focus on the second one, an attack against voter privacy. The attack was first described and discussed in [18]. The attack is done by an adversary replaying another voter's ballot or a variant of it. Due to this attack, privacy has been taken care of and the improvements done are the greatest change in the voting protocol.

The attack exploits the lack of ballot independence. That is, an adversary is able to create a relating ballot by observing another voter's interaction with the system. This chapter will describe the attack, different variants of it and the improvements done to prevent similar future attacks.

4.1 Description of the Attack

In this section we will give a general description of the attack and in the next section we will give different variants of it. We remark that in the following attacks, the only parameters included in the hash functions generating the challenges, are the commitments. Otherwise, many of the attacks would be avoided.

For simplicity we consider an election with only three voters, V_1, V_2 and V_3, which makes it easy to see the impact of the attack. Assume that each voter is allowed to vote for at most one candidate, but the attack may be generalized to n candidates. We will illustrate that an adversary replaying another voter's ballot will destroy ballot secrecy in this particular election.

Consider two of the voters to be honest, say V_1 and V_2. The adversary V_3 is able to recast the same vote as one of the others. Let us say that V_3 recasts the same vote as V_1. We define a ballot for voter V_i, where $i \in \{1, 2\}$, in an election with n candidates as:

$$V_i,\ c_{i,1}, \ldots, c_{i,n},\ pok_{i,1}, \ldots, pok_{i,n},\ pok'_i,$$

where for all $i \in \{1, 2\}$ and $j \in \{1, \ldots, n\}$:

$$\begin{array}{lcl} c_{i,j} & = & (\alpha_{i,j}, \beta_{i,j}) \\ pok_{i,j} & = & (A_{i,j}, B_{i,j}, e_{i,j}, t_{i,j}, A'_{i,j}, B'_{i,j}, e'_{i,j}, t'_{i,j}) \\ pok'_i & = & (A_i, B_i, e_i, t_i, A'_i, B'_i, e'_i, t'_i) \end{array}$$

where $pok_{i,j}$ represents the proof of knowledge which demonstrates that each ciphertext $c_{i,j}$ contains a 0 or a 1, and pok'_i represents the overall proof which demonstrates that voter i has voted for at most one candidate.

When the voting phase is over and the election result is published, the adversary will immediately recognize what V_1 voted by observing the only candidate with at least two votes, which immediately destroys privacy. He will obviously also notice what V_2 voted, since this is the only remaining vote.

This attack can violate ballot secrecy in an election with more than three voters as well. An adversary observing an honest voter's ballot and ID at the bulletin board may interfere with other dishonest voters, making them all recast the same vote. This will result in a major contribution to the vote of the honest voter, which again will result in violated privacy.

Throughout this chapter we will assume that:

- There are three voters V_1, V_2 and V_3.
- Voters V_1 and V_2 are honest.
- There are n candidates.
- The adversary is using the ballot of voter k to violate privacy. The ballot is given as:
$$V_k, c_{k,1}, \ldots, c_{k,n}, pok_{k,1}, \ldots, pok_{k,n}, pok'_k.$$
- It is only possible to vote for zero or one candidate.

4.2 Variants of the Attack

An adversary may not necessarily recast a ballot, he can cast a variant of it. In this section we give several variants of the attack and we will see how the adversary, by exploiting ballot malleability, can construct new ballots depending on already existing ballots. These constructions prove the lack of ballot independence in Helios.

4.2.1 Integer Representation Attack

The first variant of the attack uses the possibility to change the response value in the encryption proof. The adversary constructs a new ballot without changing the vote. He can do so by adding a multiple of q to the response component in the knowledge proof. Let $\gamma_1, \gamma_1', \ldots, \gamma_n, \gamma_n', \gamma, \gamma' \in \mathbb{N}$. The adversary constructs the ballot:

$$c_{k,1}, \ldots, c_{k,n}, \overline{pok}_{k,1}, \ldots, \overline{pok}_{k,n}, \overline{pok}_k',$$

where for all $j \in \{1, \ldots, n\}$

$$\overline{pok}_{k,j} = (A_{k,j}, B_{k,j}, e_{k,j}, t_{k,j} + \gamma_j q, A'_{k,j}, B'_{k,j}, e'_{k,j}, t'_{k,j} + \gamma'_j q)$$

and

$$\overline{pok}_k' = (A_k, B_k, e_k, t_k + \gamma q, A'_k, B'_k, e'_k, t'_k + \gamma' q).$$

Hence, the ballot is different, while the vote remains unchanged. The proofs are still valid.

4.2.2 Permutation Attack

The second variant of the attack is done by a permutation of the ciphertexts in the ballot. Let π be a permutation on the set $\{1, 2, \ldots, n\}$ (not the identity map). The adversary constructs the ballot:

$$c_{k,\pi(1)}, \ldots, c_{k,\pi(n)}, pok_{k,\pi(1)}, \ldots, pok_{k,\pi(n)}, pok_k'.$$

Since we just have a permutation the proofs are still valid.

4.2.3 Malformed Ciphertext Attack

We will once again see how the adversary, by observing the bulletin board and the other ballots, may construct a new ballot which is related to an already existing ballot.

Let $v \in \{1, \ldots, n\}$, then the adversary can construct the following ballot:

$$\underbrace{(1,1), \ldots, (1,1)}_{\text{v-1 times}}, (\alpha_{k,v}, \beta_{k,v}), \underbrace{(1,1), \ldots, (1,1)}_{\text{n-v times}},$$
$$\overline{pok}_1, \ldots, \overline{pok}_{v-1}, pok_{k,v}, \overline{pok}_{v+1}, \ldots, \overline{pok}_n, pok_{k,v},$$

which represents a vote for candidate number v. $pok_{k,v}$ is the knowledge proof demonstrating that $(\alpha_{k,v}, \beta_{k,v})$ contains a 0 or a 1, described in Section 2.3.1.1. Following the protocol in Figure 2.1 with $n = 2$, we see that this is a valid proof.

For all $j \in \{1, \ldots, v-1, v+1, \ldots, n\}$ we have that the proof of knowledge for ciphertext j is given by $\overline{pok}_j = (A_j, B_j, e_j, t_j, A'_j, B'_j, e'_j, t'_j)$, where $e'_j, t'_j, t_j \in_R \mathbb{Z}_q^*$ and

$$\begin{aligned} A_j &\leftarrow g^{t_j} \\ A'_j &\leftarrow g^{t'_j} \\ B_j &\leftarrow y^{t_j} \\ B'_j &\leftarrow y^{t'_j} g^{e'_j} \\ e_j &\leftarrow H(A_j, B_j, A'_j, B'_j) - e'_j \end{aligned}$$

We have the ciphertext $(\alpha, \beta) = (1, 1)$. Then, $\alpha^{e_j} = 1$ and $\alpha^{e'_j} = 1$, and from this it follows that $A_j \alpha^{e_j} = A_j = g^{t_j}$ and $A'_j \alpha^{e'_j} = A'_j = g^{t'_j}$. Similarly, we have $(\frac{\beta}{g^0})^{e_j} = 1$ and $(\frac{\beta}{g^1})^{e'_j} = g^{-e'_j}$. Hence, $B_j(\frac{\beta}{g^0})^{e_j} = B_j = y^{t_j}$ and $B'_j(\frac{\beta}{g^1})^{e'_j} = B'_j g^{-e'_j} = y^{t'_j}$, and it follows that $\overline{pok}_j$ is a valid proof for the ciphertext (1,1).

If the honest V_k, voted for candidate number v, the adversary will cast the same vote. If V_k casts a vote of abstention, the adversary will also cast a vote of abstention. In any other case the adversary will cast a vote of abstention, which will be different from V_k who voted for any other candidate in $\{1, \ldots, v-1, v+1, \ldots, n\}$.

We illustrate that this form of the attack will also violate ballot secrecy. Consider the outcome of the election to be two votes for one candidate and one vote for another candidate (no abstention vote), then the adversary knows that V_k voted for the candidate with two votes, and privacy is again violated.

4.2.4 Homomorphic Attack

Another variant of the attack exploits the homomorphic properties of ElGamal. If V_k casts a vote of abstention, the adversary is able to cast the same vote. The adversary constructs the ballot

$$(\alpha_{k,1} \cdot \ldots \cdot \alpha_{k,n}, \beta_{k,1} \cdot \ldots \cdot \beta_{k,n}), \underbrace{(1,1), \ldots, (1,1)}_{\text{n-1 times}}, pok'_k, \overline{pok}_2, \ldots, \overline{pok}_n, pok'_k$$

As long as V_k does not cast a vote of abstention, we see that this ballot represents a vote for the first candidate.

$\overline{pok}_i$ for $2 \leq i \leq n$ is a valid proof for the ciphertext $(1, 1)$. This can be shown in the same way as we did in the malformed ciphertext attack. pok'_k is a valid proof for $(\alpha_{k,1} \cdot \ldots \cdot \alpha_{k,n}, \beta_{k,1} \cdot \ldots \cdot \beta_{k,n})$ as it is a valid overall proof for V_k's ballot.

As before this type of attack violates privacy. Consider the outcome of the election to be at least two votes of abstention, then it is clear for the adversary that V_k cast a vote of abstention, and if there are two votes for the same candidate and at least one vote for the first candidate, the adversary also knows what V_k voted.

4.2.5 Another Malleability Attack

The last variant we will present is another malleability attack. The adversary chooses $\gamma_1, \ldots \gamma_n \in \mathbb{N}$. Let $\gamma = \gamma_1 + \ldots + \gamma_n$. He constructs the ballot:

$$c_{k,1}, \ldots, c_{k,n}, pok_{k,1}, \ldots, pok_{k,n}, pok'_k,$$

such that for all $j \in \{1, \ldots n\}$ we have

$$\begin{array}{lcl} c_{k,j} & = & (\alpha g^{\gamma_j}, \beta y^{\gamma_j}) \\ pok_{k,j} & = & (A_{k,j}, B_{k,j}, e_{k,j}, t_{k,j} + \gamma_j e_{k,j}, A'_{k,j}, B'_{k,j}, e'_{k,j}, t'_{k,j} + \gamma_j e'_{k,j}) \\ pok'_k & = & (A_k, B_k, e_k, t_k + \gamma e_k, A'_k, B'_k, e'_k, t'_k + \gamma e'_k). \end{array}$$

The adversary will in this case cast a vote for the same candidate as V_k. The ballot will obviously be accepted by the election officier, we have just added some new randomness in the exponent and likewise we added $\gamma_j e_{k,j}$ in the proof.

Once again privacy is broken. If the outcome of the election is two votes for one candidate and one vote for another (or a vote of abstention), the adversary will learn what V_k voted.

4.3 Solving the Problem

After these attacks were discovered, improvements were made to the scheme to ensure voter privacy. We present the improvements here.

First of all the proofs were made more robust by including more parameters in the hash generating the challenge. Here we also have the option to bind any ballot to its respective voter, by including the identity in the hash. This was not done in Helios, but we will still present the option. First we will discuss several changes made in the protocol to prevent the different variants of the attack described in the previous sections.

To prevent the first type of attack, the integer representation attack, we want to prevent the adversary from tamper with the response value in the proof. We solve this by a simple modification in the protocol of the knowledge proof such that the response components are unchangeable. We explain how below. Another possible solution is to let the election officer reject all the ballots which contains a ciphertext that already exists on the bulletin board. This last method will also prevent the homomorphic attack.

The malformed ciphertext attack could also be avoided by rejecting already existing ciphertexts. Another alternative is to force the decryptor to only decrypt ciphertexts (α, β) where $\alpha, \beta \neq 1$, that is $r \neq 0$. This method will also be sufficient to avoid the homomorphic attack, which includes a lot of ciphertexts on the form $(1, 1)$.

The last attack described, another malleability attack, can be avoided the same way as the first attack, by making it impossible to change the response value.

Overall a lot more contexts were included in the hash generating the challenge. Especially, α, β were included. In the previous versions of Helios, the hash generating the challenge only used the commitments as input. Including more parameters will make it more difficult to fake the proofs for an adversary. This way the response components are unchangeable. Note that when we presented the non-interactive version in Section 2.6 we took the changes into account and included (g, y, α, β) in the hash.

In addition we can also bind the ballots to voters by including the voter identity in the hash. This technique uses unique identifiers, first proposed by [8], and include the voter's identity within in the hash that generates e. In the non-interactive proof we have that $e \leftarrow H_{12}(g, y, \alpha, \beta, A_0, B_0, A_1, B_1)$. To make the ballot unique to every voter we include their identity $e \leftarrow H_{12}(g, y, \alpha, \beta, A_0, B_0, A_1, B_1, ID)$. Then, we have for $i \in \{0, 1\}$

$$e_i \leftarrow H_{12}(g, y, \alpha, \beta, A_0, B_0, A_1, B_1, ID) - e_{j \neq i},$$

where $j \in \{0, 1\}$.

This method prevents an adversary from recasting a ballot or a variant of it, because he is not able to include his ID in an already existing ballot. Despite this, unique identifiers were not implemented in Helios due to Benaloh's principle of ballot casting assurance. Use of the unique identifiers will require the voters to sign in before preparing their ballots, since their ID will be included in the proof. Then there is no longer possible for people not eligible to vote, to audit the ballot preparation and encryption. Hence, this change would have weakened the auditability and verifiability in the protocol. The creators of Helios consider Benaloh's principle of ballot casting assurance and verifiability as more important, and recommend other systems if voter privacy is of great importance.

Chapter 5

Changing the Protocol

This chapter will introduce some changes we have done to the protocol. The main changes is done in the proof of correct encryption, where we introduce a new generator and add one more proof of correction. In Section 2.6 we mentioned that we are not able to prove soundness in the non-interactive proof. Because of this we are not able to give a formal proof of ballot secrecy. We will solve this problem by adding a new correction proof to the knowledge proof, then we no longer need to extract the witness.

We will further make a security analysis to make sure that the new protocol satisfies verifiability as the old protocol did, and then make a formal proof of ballot secrecy.

5.1 The New Protocol

This section will present the new encryption proof. The encryption remains the same, so an encrypted vote is still given as $(\alpha, \beta) = (g^r, y^r g^m)$. The Σ-protocol in Figure 2.1 proving that each ciphertext contains a 0 or a 1 and the Σ-protocol proving that the product contains a value between 0 and $n-1$ are still included. We will further only point out the changes, what is not mentioned remains the same as before.

We introduce a new generator $\overline{g}$ which is chosen randomly from $\mathbb{Z}_q^*$ and must be generated during key generation. During encryption we add the element $\overline{\alpha} \leftarrow \overline{g}^r$ to the ciphertext. We use a Σ-protocol for equality for $log_g \alpha = log_{\overline{g}} \overline{\alpha}$, defined in Section 2.4, and add it in the proof of correctness, in addition to the Σ-protocols used in the previous version. Let H_2 be the hash function in this proof. Hence,

we add the following to the proof

$$\begin{aligned}&\text{Let } x \in \mathbb{Z}_q^*,\\ &A \leftarrow g^x, \overline{A} \leftarrow \overline{g}^x,\\ &e \leftarrow H_2(g, \overline{g}, \alpha, \overline{\alpha}, A, \overline{A}), t \leftarrow x + re,\end{aligned}$$

and verify that

$$g^t \stackrel{?}{=} A\alpha^e \text{and } \overline{g}^t \stackrel{?}{=} \overline{A}\overline{\alpha}^e.$$

There is no known way to prove that the non-interactive proof is a proof of knowledge. As described in Section 2.3.1 proving the encryption proof to be a proof of knowledge is done by rewinding. In order to extract the witness, the extractor needs to rewind the prover to the point after the prover has sent his commitments, send the prover different challenges and play it several times, until the extractor gets two accepting conversations. This works fine in the interactive case, but with a hash function it will never work. The hash function will never find two different challenges and hence not two accepting conversations, corresponding to the same commitments.

Remember that, as mentioned in Section 2.5, we are still able to prove the equality between two logarithms with just one accepting conversation in the non-interactive proof, though we are not able to rewind and extract the witness.

In the security analysis in the following section we introduce a trick based upon ideas from [17], which enables us to cheat in a way so that we avoid extracting the witness.

5.2 Security Analysis

The original version of Helios offers end-to-end verifiability and ballot secrecy (though we are not able to formally prove the last one). To avoid lack of security it is important that our new protocol fulfills these requirements as well. The following sections will give an analysis of the security requirements in the new protocol. First will we give some formal definitions.

5.2.1 Defining Verifiability

Helios offers end-to-end verifiability meaning that the voter and anyone else can audit every step from ballot preparation to the final tally.

Verifiability gives the opportunity to verify that every ballot is included in the tally and counted correctly. It is a part of the integrity in the system making sure that no one tampers with the ballots. The votes coming into the system, being encrypted, combined and decrypted, should be consistent with the final result.

What we really want when a voter is submitting a vote is that the vote is *cast as intended*. Then the vote is stored at the bulletin board, and we want it to be *stored as cast*. At last the the ballot should be *counted as stored*.

The original version of Helios fulfills these requirements. With the ballot casting assurance everyone is able to audit the encryption process. When voters have cast their votes, the encrypted ballots appear on the bulletin board along with proofs and ID's or aliases of the voters. This way anyone can check that the vote is included and that the proofs are correct.

The final tally is published on the bulletin board along with proofs of correctness, available for anyone to verify. Hence, Helios offers end-to-end verifiability.

5.2.2 Defining Ballot Secrecy

We will in this section discuss the notion of ballot secrecy. Briefly speaking, ballot secrecy means that no one should be able to see your vote.

Consider the system in Figure 5.1, representing a voting system. The final output is known to everyone, that is, the sum of all the votes. Then, no one should be able to tell what you voted from this output, and this is the meaning of a secret ballot.

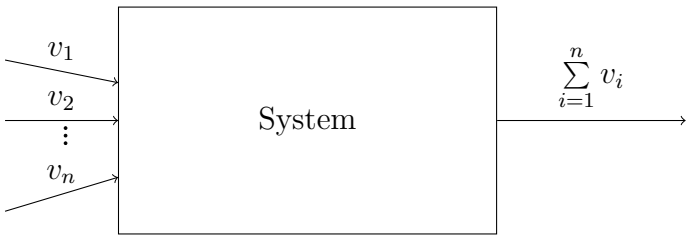

Figure 5.1: Voting system.

Now, assume that some of the input is known. Say, there are n votes in total and m of these are known. As long as $n - m \geq 2$, we should still not be able to say anything about the remaining votes.

Consider a smaller scenario, an election with four voters V_1, V_2, V_3 and V_4, and their corresponding votes v_1, v_2, v_3 and v_4. This scenario is represented in Figure 5.2. After the election is done and the final result is counted and published, we have the sum of the four votes $\sum_{i=1}^{4} v_i$. Assume that we know v_1 and v_2, that is, the input to V_1 and V_2. Then, the only remaining information is the sum of v_3 and v_4, and we are still not able to say anything about what V_3 and V_4 voted.

Now, consider two voters V_0 and V_1, both honest, and two possible candidates d_0 and d_1. We claim that it should be impossible to distinguish the scenario where V_0 is voting for d_0 and V_1 is voting for d_1, from the scenario where V_0 is voting for

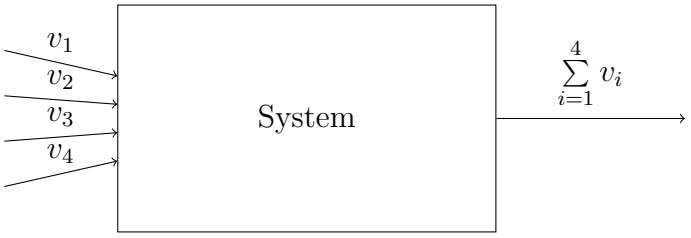

Figure 5.2: Ballot secrecy.

d_1 and V_1 is voting for d_0. We generalize it, and give a formal definition of ballot secrecy. We play the following game between a simulator and an adversary.

Ballot Secrecy. *The simulator chooses a bit* $b \in_R \{0,1\}$*, generates* $g, \overline{g} \in_R \mathbb{Z}_q^*$ *and the keypair* (a, y)*, and sends* $g, \overline{g}$ *and* y *to the adversary. The adversary queries the encryption oracle for encryptions of* $m_1, m_2, \ldots, m_n$ *where* $m_i = m_{i0}$ *or* m_{i1} $\forall$ $i \in \{1, 2, \ldots, n\}$*. The simulator computes*

$$c_{ij} \leftarrow ((g^{r_{i0}}, \overline{g}^{r_{i0}}, y^{r_{i0}} g^{\pi_b(m_{i0})}), (g^{r_{i1}}, \overline{g}^{r_{i1}}, y^{r_{i1}} g^{\pi_b(m_{i1})}))$$

$\forall$ $i \in \{1, 2, \ldots, n\}$ *and sends* c_{ij} *to the adversary along with the proof of correct encryption. That is, he encrypts* $m_1, m_2, \ldots, m_n$ *or a permutation of them.*

The adversary may at all times query a decryption oracle for decryptions of arbitrary ciphertexts c^**, as long as* $c^* \neq c_{ij}$ $\forall$ $i \in \{1, 2, \ldots, n\}$*. The simulator verifies the proofs included in the ciphertexts and if they are valid, he responds with the decryption, otherwise he outputs* $\perp$*.*

At last the adversary outputs b' *and wins if* $b = b'$*. Then the advantage of the adversary is given as*

$$\epsilon = |Pr[E_0] - \frac{1}{2}|.$$

The reason why we subtract the half is that an adversary may always guess with success probability $\frac{1}{2}$, but we want to find the advantage which is better than just guessing.

Note that since the proofs are checked, the adversary will not be able to create ciphertexts related to the c_i's.

5.2.3 Verifiability in the New Protocol

The changes we have made to the protocol will not affect the end-to-end verifiability. As before the voter and anyone else are able to run the ballot preparation as many times as they want and verify that the encryption is correct. The public bulletin board enables a voter to see that his vote is included and anyone else is also able to follow the votes appearing on the bulletin board and verify the proofs.

When the election is ended, the ballots are jointly decrypted and the result is published at the bulletin board along with proofs of correctness. Hence, anyone is able to verify that the decryption was correct.

So, the new system still offers end-to-end verifiability.

5.2.4 Ballot Secrecy in the New Protocol

In this section we will prove that our new system offers ballot secrecy. We prove the following theorem.

Theorem 5.1. *If Decision Diffie-Hellman is (ϵ_{DDH}, T)-hard, then any adversary against ballot secrecy has advantage at most $\epsilon_{DDH} + \frac{5n}{q-1}n_q + \frac{2(\eta+1)}{q-1}$, where n_q is the number of hash queries made by the adversary and η is the number of queries to H_2.*

The proof of Theorem 5.1 is a sequence of games between a simulator and an adversary. We start with the initial game described in the definition in section 5.2.2. In the end we want to make sure that no information about b leaks to the adversary, so we start to change the initial game and make small modifications through several games.

The first change made is just some bookkeeping to remember the ciphertexts for later use, and then we forge the encryption proofs of honestly generated ciphertexts by using the honest verifier simulation.

Further we remove the secret key, so that the adversary has no chance to learn it. Before we remove it, we introduce a little trick based upon ideas from [17]. This in addition to the new encryption proof give us another way to decrypt ciphertexts.

At last we modify the encryption process, first by encrypting using DDH tuples, then by encrypting using random tuples. This way no information about b is available to the adversary.

Let σ be the set of all the proofs with commitments, challenges and responses. In the following proof let a ciphertext be given as $c_{ij} = (\alpha_{ij}, \overline{\alpha}_{ij}, \beta_{ij}, \sigma_{ij})$ and let E_i denote the event that the adversary outputs $b = b'$ in game i.

Proof. **Game 0** The initial game is exactly the same as described in the definition of ballot secrecy, in Section 5.2.2.

Assume that the adversary sends n messages, uses time at most T and queries n_q times to the hash oracle.

The advantage of the adversary is

$$\epsilon = |Pr[E_0] - \frac{1}{2}|.$$

Game 1 This game proceeds exactly as the previous one, we just add a little bookkeeping. We will later remove the secret key and change the decryption process, and we will need to remember the content of any honestly generated ciphertext. So in this game, every time an honest computer encrypts a valid ballot, we store the content of the ciphertext for later use.

It is not possible to observe this additional bookkeeping and the game is indistinguishable from Game 0. Hence

$$Pr[E_0] = Pr[E_1].$$

Game 2 This game is exactly as Game 1, but now we are forging the proofs for every honestly generated ciphertext. This is done the same way as the honest verifier simulation in Section 2.3.1.1. We simulate the individual proofs, the overall proof and the new equality proof. Then, for challenges e_{1i}, e_2 and e_{3i}, where $i \in \{1, 2, \ldots, n\}$, we reprogram the random hash functions such that $H_{12}(g, y_i, \alpha_i, \beta_i, A_{0i}, B_{0i}, A_{1i}, B_{1i}) \leftarrow e_{1i}$, $H_{1n}(g, y, \alpha, \beta, A_0, B_0, A_1, B_1, \ldots, A_{n-1}, B_{n-1}) \leftarrow e_2$ and $H_2(g, \overline{g}, \alpha_i, \overline{\alpha}_i, A_i, \overline{A}_i) \leftarrow e_{3i}$.

This modification may be observable. If any of the hashes is already asked for by the adversary, we have a problem. A random oracle is consistent, asking for the same hash will always return the same answer. That means, if the adversary has already queried for the hash, we are not able to reprogram the random hash function, and hence not able to fake the proofs. From this it follows that the difference between this game and the previous is dependent on whether or not the adversary has queried the hash. The challenge is asked for $2n$ times in the individual proofs, n times for the overall proof and $2n$ times for the new protocol, which adds up to $5n$. Since the number of queries to the hash oracle is n_q and a challenge is in the group $\mathbb{Z}_q^*$, we have

$$|Pr[E_1] - Pr[E_2]| \leq \frac{5n}{q-1} n_q.$$

Game 3 This game proceeds as before. In addition we pick a random value l from the group $\mathbb{Z}_q^*$ and we set $\overline{g} \leftarrow y^l$. Given a valid ciphertext c_{ij} and the soundness property we have $log_g \alpha_{ij} = log_{\overline{g}} \overline{\alpha}_{ij}$, then $\alpha_{ij} = g^{r_{ij}}$ and $\overline{\alpha}_{ij} = \overline{g}^{r_{ij}}$. From this follows $\overline{\alpha}_{ij}^{1/l} = \overline{g}^{r_{ij}/l} = y^{r_{ij} l/l} = y^{r_{ij}}$.

Introducing this is not observable and we have

$$Pr[E_2] = Pr[E_3].$$

Game 4 We change the decryption and decrypt by $\beta\overline{\alpha}^{(-1/l)}$. Then we remove the secret key a, and from now on the public key y is just randomly chosen from $\mathbb{Z}_q^*$.

These changes can only be observed if the adversary manages to fake the equality proofs. Theorem 2.1 gives that the probability of this is $2(\eta+1)(q-1)^{-1}$, where η is the number of queries to H_2. Note that $\eta \leq n_q$. From this follows

$$|Pr[E_3] - Pr[E_4]| \leq \frac{2(\eta+1)}{q-1}.$$

Game 5 We continue as before, but now we make a modification to the encryption process. We want to encrypt using 3-DDH tuples. Let $(g, \overline{g}, y, u_1, \overline{u}_1, u_2)$ be a 3-DDH tuple, where $u_1 = g^{t_{ij}}$, $\overline{u}_1 = \overline{g}^{t_{ij}}$ and $u_2 = y^{t_{ij}}$ for some t_{ij} in $\mathbb{Z}_q^*$. We use this tuple to encrypt, and for $r_{ij}, s_{ij} \in_R \mathbb{Z}_q^*$ we get

$$(\alpha_{ij}, \overline{\alpha}_{ij}, \beta_{ij}) = (g^{r_{ij}} u_1^{s_{ij}}, \overline{g}^{r_{ij}} \overline{u}_1^{s_{ij}}, y^{r_{ij}} u_2^{s_{ij}} g^{m_{ij}}).$$

This modification is unobservable and

$$Pr[E_4] = Pr[E_5].$$

Game 6 In the last game we want to change the encryption a little more. Now we let $(g, \overline{g}, y, u_1, \overline{u}_1, u_2)$ be a random tuple where $u_1 = g^{t_{ij}}$, $\overline{u}_1 = \overline{g}^{t'_{ij}}$ and $u_2 = y^{t''_{ij}}$ for some $t_{ij}, t'_{ij}, t''_{ij}$ in $\mathbb{Z}_q^*$, and $t_{ij} \neq t'_{ij} \neq t''_{ij}$. Then, for $r_{ij}, s_{ij} \in_R \mathbb{Z}_q^*$ we have

$$(\alpha_{ij}, \overline{\alpha}_{ij}, \beta_{ij}) = (g^{r_{ij}} u_1^{s_{ij}}, \overline{g}^{r_{ij}} \overline{u}_1^{s_{ij}}, y^{r_{ij}} u_2^{s_{ij}} g^{m_{ij}}).$$

Note that the adversary will not notice that $log_g \alpha_{ij} \neq log_{\overline{g}} \overline{\alpha_{ij}}$ from the proofs, since they are forged.

We observe that the only difference between this game and the previous one is whether or not we use the 3-DDH tuple for encryption. We prove the following lemma.

Lemma 5.2. *Suppose 3-DDH is (ϵ_{DDH}, T)-hard, then*

$$|Pr[E_5] - Pr[E_6]| \leq \epsilon_{DDH}.$$

Proof. Let $\mathcal{D}$ be a distinguisher between Game 5 and Game 6 and let $\mathcal{A}$ be the adversary against ballot secrecy. $\mathcal{A}$ runs $\mathcal{D}$ as a subroutine and it is

important to notice that $\mathcal{A}$ controls what $\mathcal{D}$ sees. $\mathcal{A}$ will use $\mathcal{D}$ to distinguish 3-DDH tuples from random tuples.

Let $(g, \overline{g}, y, u_1, \overline{u}_1, u_2)$ be a random instance of the decision Diffie-Hellman problem. If $u_1 = g^{t_{ij}}, \overline{u}_1 = \overline{g}^{t_{ij}}$ and $u_2 = y^{t_{ij}} = g^{at_{ij}}$, the tuple was drawn from the distribution G^4, and if $u_1 = g^{t_{ij}}, \overline{u}_1 = \overline{g}^{t'_{ij}}$ and $u_2 = y^{t''_{ij}} = g^{at''_{ij}}$, the tuple was drawn from the distribution of random tuples.

The job of $\mathcal{A}$ is to distinguish between these two distributions.

$\mathcal{A}$ makes encryptions $(g^{r_{ij}} u_1^{s_{ij}}, \overline{g}^{r_{ij}} \overline{u}_1^{s_{ij}}, y^{r_{ij}} u_2^{s_{ij}} g^{m_{ij}})$, simulates the proofs and sends the ciphertexts to $\mathcal{D}$. Since $\mathcal{A}$ runs $\mathcal{D}$, $\mathcal{A}$ controls the hash values which $\mathcal{D}$ will use to verify the proofs. This way $\mathcal{A}$ is always able to fake proofs such that they will always be accepted by $\mathcal{D}$. At last $\mathcal{A}$ waits for $\mathcal{D}$ to distinguish whether the encryption came from Game 5 or from Game 6.

If $(g, \overline{g}, y, u_1, \overline{u}_1, u_2)$ is drawn from the 3-DDH distribution, $\mathcal{A}$ simulates the ciphertexts from Game 5. If $(g, \overline{g}, y, u_1, \overline{u}_1, u_2)$ is drawn from the distribution of random tuples, $\mathcal{A}$ simulates the ciphertexts from Game 6. Eventually $\mathcal{D}$ makes a guess and outputs a bit, then $\mathcal{A}$ outputs the same bit. Since the ciphertexts perfectly model Game 5 and Game 6 respectively, we must have

$$|Pr[E_5] - Pr[E_6]| \leq \epsilon_{DDH},$$

which proves the lemma. □

Game 7 In this game we go back and let everything be as in Game 2, but instead of encrypting the message m_{ij}, we encrypt random group elements γ_{ij}.

In the previous game the β_{ij}'s are encryptions of random elements from G, since u_2 is random. In this game we encrypt random elements γ_{ij}. Hence, the distribution of β_{ij} is equal in Game 6 and Game 7, and the adversary will not see any difference between these games.

$$Pr[E_6] = Pr[E_7].$$

Analysis We observe that we encrypt random group elements in the last game. Therefore the ciphertexts contain no information about the bit b, and the adversary has no more advantage than guessing, that is

$$Pr[E_7] = \frac{1}{2}.$$

Finally, we have that the advantage of any adversary is given by

$$\begin{aligned}\epsilon &= |Pr[E_0] - \tfrac{1}{2}| \\ &= |Pr[E_0] - Pr[E_1] + Pr[E_1] - \ldots - Pr[E_7] + Pr[E_7] - \tfrac{1}{2}|,\end{aligned}$$

using the triangle inequality we get

$$\epsilon \leq |Pr[E_0] - Pr[E_1]| + |Pr[E_1] - Pr[E_2]| + \ldots + |Pr[E_7] - \frac{1}{2}|.$$

From this follows

$$\epsilon \leq \epsilon_{DDH} + \frac{5n}{q-1} n_q + \frac{2(\eta+1)}{q-1},$$

which proves the theorem.

□

Chapter 6

Applying Helios to the Norwegian County Election

So far we have encrypted using the exponential ElGamal cryptosystem. This chapter introduces a combination of ElGamal and the generalisation of Paillier's public key system. We want to see if we can find a reasonably efficient voting protocol with homomorphic encryption for the Norwegian county election.

The current version of Helios with the exponential ElGamal works perfectly fine for elections where the exponents are relatively small. As previously mentioned we have to compute the discrete logarithm to obtain the final result. This becomes hard in elections with homomorphic encryption where the exponents tend to grow large. Homomorphic encryption is desirable, but in the Norwegian county election the exponents will grow large. In the new system it is possible to efficiently compute discrete logarithms to a base $(1 + n)$.

We will first describe the system and the encryption and decryption process. Secondly we will apply the voting scheme to the Norwegian county election and at last we will use a numerical example with data from Hordaland and analyse if it is an efficient voting scheme.

6.1 The New Encryption Scheme

We introduce the new encryption scheme where we combine ElGamal with a generalisation of Paillier's public key system [12]. With this new system we obtain the properties from both ElGamal and Paillier. With ElGamal follows the Σ-protocol which enables us to prove correct encryption, and we are also able to make a formal security proof, as in the previous chapter. Blending this with Paillier, we obtain a system where we are able to compute discrete logarithms, and hence get the final result even if the exponent is large. In this section we give some theoretical

background and explain the encryption and decryption.

Let $p = 2p' + 1$ and $q = 2q' + 1$, where p, p', q and q' are primes. We have that $gcd(p-1, q-1) = 2$. Let $n = pq$. We will work in the ring $\mathbb{Z}^*_{n^{s+1}}$ which has some nice properties. A property we will use is that the discrete logarithm to base $(1+n)$ is easy to compute. We will soon explain how.

We fix the element g in $\mathbb{Z}^*_{n^{s+1}}$ and let $|g| = p'q'$. We have some T such that $T \geq |g|^2$ (one alternative is to choose $T = n^2$). Let $x \in_R \{0, 1, \ldots, T\}$ and $y = g^x$. Then $\mathbb{Z}^*_{n^{s+1}}, g$ and y are the public keys while x is the private key.

To encrypt we choose a random r_i from $\{0, 1, \ldots, T\}$. The encryption of a message $m_i \in \mathbb{Z}_{n^s}$ is then given by

$$c_i = (\alpha_i, \beta_i) = (g^{r_i}, y^{r_i}(1+n)^{m_i}) \; mod \; n^{s+1}.$$

The encryption is homomorphic, which is of our interest. We see that by multiplying the ciphertexts together, the m_i's, for $i \in \{1, 2, \ldots, k\}$, are added in the exponent, and we obtain

$$c = \prod_{i=1}^{k} c_i = \left(\prod_{i=1}^{k} \alpha_i, \prod_{i=1}^{k} \beta_i \right) = \left(\prod_{i=1}^{k} g^{r_i}, \prod_{i=1}^{k} y^{r_i}(1+n)^{m_i} \right)$$

$$= \left(g^{\sum_{i=1}^{k} r_i}, y^{\sum_{i=1}^{k} r_i}(1+n)^{\sum_{i=1}^{k} m_i} \right) \; mod \; n^{s+1}.$$

To obtain the original plaintext we compute

$$(1+n)^{\sum_{i=1}^{k} m_i} = \prod_{i=1}^{k} \beta_i y^{-\sum_{i=1}^{k} r_i} = \left(\prod_{i=1}^{k} \beta_i \alpha_i \right)^{-x}.$$

In the original protocol we had $g^{\sum m_i}$ and in that group it was hard to compute discrete logarithms. Now we are in the ring $\mathbb{Z}^*_{n^{s+1}}$ and we have $(1+n)^{\sum m_i}$ where we are able to compute $\sum m_i$ [5]. We explain how to do it.

Set $a = \sum m_i$ and we want to find $a \, (mod \, n^{s+1})$. The idea is to compute $a_j = a \, (mod \, n^j)$ for $j = 1, 2, \ldots, s+1$. Let $L()$ be a function such that $L(b) = \frac{b-1}{n}$. We apply the function on $(1+n)^a$, and we will find a way to extract a. First of all we have

$$(1+n)^a = \sum_{j=0}^{a} \binom{a}{j} n^j = 1 + an + \binom{a}{2} n^2 + \ldots + \binom{a}{a-1} n^{a-1} + n^a.$$

Applying the function $L()$ we get

$$L((1+n)^a \; mod \, n^{s+1}) = (a + \binom{a}{2} n + \ldots + \binom{a}{s} n^{s-1}) \; mod \, n^s.$$

Assume we are in the j' th step and that we know a_{j-1}, that is $a\ (mod\, n^{j-1})$. We have that $a_j = a_{j-1}\ (mod\, n^{j-1})$ which implies $a_j = a_{j-1} + \gamma n^{j-1}$ for some $0 \leq \gamma < n$.

We want $a_j = a\ (mod\, n^j)$, so

$$L((1+n)^a\ mod\, n^{j+1}) = (a_j + \binom{a_j}{2}n + \ldots + \binom{a_j}{j}n^{j-1})\ mod\, n^j.$$

Let $0 < t < j$ and assume that

$$\binom{a_j}{t+1}n^t = \binom{a_{j-1}}{t+1}n^t\ (mod\, n^j).$$

Using this we have

$$L((1+n)^a\ mod\, n^{j+1}) = (a_{j-1} + \gamma n^{j-1} + \binom{a_{j-1}}{2}n + \ldots + \binom{a_{j-1}}{j}n^{j-1})\ mod\, n^j.$$

We rewrite the above and obtain

$$a_j = a_{j-1} + \gamma n^{j-1} = L((1+n)^a\ mod\, n^{j+1}) - (\binom{a_{j-1}}{2}n + \ldots + \binom{a_{j-1}}{j}n^{j-1})\ mod\, n^j.$$

Algorithm 1 could be used to calculate $a_j = a\ (mod\, n^j)$ which gives us the result $\sum m_i$.

Algorithm 1 Calculating a_{s+1}

```
a = 0;
for i = 1 to s do
    t_1 = L(a mod n^{i+1});
    t_2 = a;
    for k = 2 to i do
        a = a − 1;
        t_2 = t_2 a (mod n^i);
        t_1 = t_1 − (t_2 n^{k−1})/k! (mod n^i);
    end for
    a = t_1;
end for
```

6.2 Correct Encryption

We can use the same protocol as in Figure 2.1 with just some small modifications to prove correct encryption. We are now working in the ring $\mathbb{Z}^*_{n^{s+1}}$, hence we pick

the elements from other sets. For a ciphertext $c = (g^r, y^r(1+n)^i)$ we pick a random x_i from $\{0, 1, \ldots, T\}$. We also choose e_j and t_j for all $j \neq i \in \{0, 1, \ldots, k\}$ from the set $\{0, 1, \ldots, T\}$. The challenge e is randomly chosen from $\{0, 1\}^l$ for some l.

The protocol is still complete and special honest verifier zero-knowledge, but we are not able to prove special soundness required for it to be a Σ-protocol.

But we do have some soundness if we can find t, t', e, e' such that $gcd(e, e') = 1$, and $g^t = \alpha^e$ and $g^{t'} = \alpha^{e'}$. Then, since e and e' are relatively prime, there exists a, b such that $ae + be' = 1$, and we obtain

$$g^{at+bt'} = g^{at}g^{bt'} = \alpha^{ae}\alpha^{be'} = \alpha^{ae+be'} = \alpha.$$

We need some e_i and e_i' such that $gcd(e_i, e_i') = 1$. Now, if we rewind four times instead of two, we get $g^{t_i - t_i'} = \alpha^{e_i - e_i'}$ and $g^{t_i'' - t_i'''} = \alpha^{e_i'' - e_i'''}$. This could be used to extract the discrete logarithm, but we do not have a formal proof that $e_i - e_i'$ is relatively prime to $e_i'' - e_i'''$. From [11] we have that the probability of two random elements being relatively prime is $\frac{6}{\pi^2}$, hence it is not unreasonable to expect that they will be.

6.3 Applying Helios to the Norwegian County Election

We consider the Norwegian county election and apply Helios with Paillier encryption.

In the Norwegian county election it is possible to vote for at most one party and in addition it is possible to submit personal votes for candidates within the party you voted for.

We represent a vote as $(m_1, m_2, \ldots, m_k)$ where k is the number of parties and candidates. Hence, a ballot consists of a list of parties followed by candidates.

Let N be the total number of eligible voters and we use L^i for some $L > N$ to be a unique representation of each party/candidate for $i \in \{1, 2, \ldots, k\}$.

Like before, we encrypt each party/candidate to encode a 0 or a 1, and thus we get k ciphertexts. Each ciphertext of voter V_j is given as

$$c_{ij} = (g^{r_{ij}}, y^{r_{ij}}(1+n)^{m_{ij}}) \; mod \; n^{s+1},$$

where $m_{ij} = 0$ or $m_{ij} = 1$.

We multiply all the ballots together, and for V_j we obtain

$$c^{(j)} = \prod_{i=1}^{k} c_{ij}^{L^i} = \left(g^{\sum_{i=1}^{k} r_{ij}L^i}, y^{\sum_{i=1}^{k} r_{ij}L^i}(1+n)^{\sum_{i=1}^{k} m_{ij}L^i} \right) mod \; n^{s+1}.$$

The final result is then given as

$$\begin{aligned} C &= \prod_{j=1}^{N} c^{(j)} = \left(\prod_{j=1}^{N} \prod_{i=1}^{k} g^{r_{ij}L^i}, \prod_{j=1}^{N} \prod_{i=1}^{k} y^{r_{ij}L^i} (1+n)^{\sum_{i=1}^{k} m_{ij}L^i} \right) \\ &= \left(g^{\sum_{j=1}^{N}\sum_{i=1}^{k} r_{ij}L^i}, y^{\sum_{j=1}^{N}\sum_{i=1}^{k} r_{ij}L^i} (1+n)^{\sum_{j=1}^{N}\sum_{i=1}^{k} m_{ij}L^i} \right) mod\ n^{s+1}. \end{aligned}$$

The decryption of the final tally is

$$m = \sum_{i=1}^{k} a_i L^i \ (mod\ n^{s+1}),$$

where $a_i = \sum_{j=1}^{N} m_{ij}$, that is the total amount of votes for candidate i.

The result can be calculated with Algorithm 1 in Section 6.1.

In the original version of Helios we used an overall proof to show that the product of ciphertexts for each voter contains a value between 0 and some maximum number. In this case we have to multiply the c_{ij}'s together for each voter without the representation of L^i. Remark that we are still not able to prove that a voter voted for eligible candidates. In the county election the voters are not allowed to vote for candidates outside the party they voted for. If we apply the proof in Section 2.3.1.1 on the product of ciphertexts, we just guarantee that V did not submit too many votes, it says nothing about whether or not the candidates that V voted for belong to the party that V voted for.

So we do the following. Each single ciphertext c_i is proved to contain a 0 or a 1 with the protocol in Section 6.2. Secondly we need to check that the product of ciphertexts of the parties contains a 0 or a 1, that is that the voter voted for at most one party.

Next we need to check if the product of ciphertexts of candidates of a party contains a value larger than 0. If so, the ciphertext of the corresponding party should contain a 1 and all the other products should be 0. For simplicity we enumerate the ciphertexts for the candidates c_{il} with two indices where i tells us which party the candidates represent and l is the candidate number. Hence the representation will be

$$\begin{aligned} &c_1, c_2, \ldots, c_{k_p}, c_{1,1}, c_{1,2}, \ldots, c_{1,l_{1max}}, c_{2,1}, c_{2,2}, \\ &\ldots, c_{2,l_{2max}}, \ldots, c_{k_p,1}, c_{k_p,2}, \ldots, c_{k_p,l_{k_pmax}}. \end{aligned}$$

If a party has less than l_{max} candidates, we add several 1's to complement the party.

To check that a voter did not submit votes for candidates outside the party he voted for, we check if the next l_{imax} ciphertexts corresponding to party i is different from 0. If it is, c_i should contain a 1. That is, we prove that the product of the l_{imax} ciphertexts contains 0 or that c_i contains a 1. This can be done with a 2-OR-protocol and must be done for each party.

So, what should be published on the bulletin board is each single ciphertext $c_1, c_2, \ldots, c_k$ along with individual proofs that each ciphertext contains a 0 or a 1. We also publish the product of the ciphertexts of parties along with a proof that it contains a 0 or a 1. At last we publish the product of ciphertexts of candidates for each party, and the 2-OR-protocols as described above.

6.4 Security Analysis

In this section we will analyse the security in the new protocol.

Since we publish every ciphertext on the bulletin board along with individual proofs and the 2-OR-protocols as described in the previous section, we see that verifiability is still maintained. The principle of ballot casting assurance is also as before, a voter needs to login before he wants to cast his vote, hence everyone can verify the encryption process.

Then there is ballot secrecy left to prove. In the previous chapter we introduced $\overline{\alpha} \leftarrow \overline{g}^r$, and we were able to make a formal proof of ballot secrecy. We are not able to do this anymore. We could make the same security proof as that of Theorem 5.1, but the cheating we do in Game 3 will not work in this ring. There probably exists some similar trick which we could use in this version or we could use other protocols to prove correct encryption, but we did not have time to consider other solutions.

6.5 Numerical Example

In this section we will give a numerical example from the county election i Hordaland 2011. The reason why we look at this is because we get the restriction $\sum_{i=1}^{k} a_i L^i < n^s$, since the element $(1+n)$ has order n^s in the ring $\mathbb{Z}^*_{n^{s+1}}$ for any $s < p, q$ [5].

We have chosen some data from the last county election in Norway, held in 2011. The data are given in Table 6.1. By investigating the last election in Norway and the statistics regarding number of voters, number of candidates and parties, we found Hordaland to be one of the counties with most parties and candidates. The data are from [15]. The number of people eligible to vote in Hordaland is 368 514 [16].

Table 6.1: Statistics from the county election 2011 in Hordaland.

Number of eligible voters	368 514
Number of parties	15
Total number of different candidates	704
Maximum number of candidates within a party	63

We need $\sum_{i=1}^{k} a_i L^i < n^s$. We also need each a_i to be less than L, such that the representation $\sum_{i=1}^{k} a_i L^i$ is unique. In the most extreme case the value of a_i is 368 514, that is, every voter voted for party/candidate number i. If we also consider that Oslo had 477 933 eligible voters and take population growth into account, we should set $L = 500\,000 \approx 2^{19}$. The value of k is $704 + 15 = 719$. From this follows

$$\sum_{i=1}^{k} a_i L^i < L^{k+1} = (2^{19})^{720}.$$

This means that n^s must be $(2^{19})^{720}$. Using $s = 1$ results in $n = 2^{13680}$, a very large number, thus we should use $s \geq 2$, which tells us why we should work in $\mathbb{Z}^*_{n^{s+1}}$, $s \geq 2$. Trying different values for s we obtain the values in Table 6.2.

Table 6.2: Values of n and n^{s+1} for different s.

s values	n values	n^{s+1}
$s = 1$	$n = 2^{13680}$	$n^{s+1} = 2^{27360}$
$s = 2$	$n = 2^{6840}$	$n^{s+1} = 2^{20520}$
$s = 3$	$n = 2^{4560}$	$n^{s+1} = 2^{18240}$
$s = 4$	$n = 2^{3420}$	$n^{s+1} = 2^{17100}$
$s = 5$	$n = 2^{2736}$	$n^{s+1} = 2^{16416}$
$s = 6$	$n = 2^{2280}$	$n^{s+1} = 2^{15960}$
$s = 7$	$n = 2^{1954.5}$	$n^{s+1} = 2^{15636}$

For $s > 7$ we have that n becomes so small that we could worry about the security of n. Hence, we choose to stop at $s = 7$, and will do the further calculations with $s \in \{1, 2, \ldots, 7\}$.

6.6 Costs of the Calculations

We will now study the actual cost of doing the computations in the encryption and decryption. First we will look at the time it takes to do one exponentiation, then we will look at the cost of the encryption process and the counting.

It is the exponentiation which is expensive in the encryption process, thus we measure the time it takes to calculate one exponentiation for $s \in \{1, 2, \ldots, 7\}$. The computations are done in PARI/GP [6] on a PC with 2.67 GHz 4×6-core Xeon 24 Intel CPUs. n is found by generating two different primes with roughly the same size where their product is close to (and larger than) the n values in Table 6.2. We do 100 computations and then average the result to get the time of one exponentiation. The output is presented in Table 6.3.

Table 6.3: The cost of the encryption.

s values	Time (ms)
$s = 1$	1829
$s = 2$	1179
$s = 3$	973
$s = 4$	876
$s = 5$	824
$s = 6$	776
$s = 7$	752

As we see, for the highest n modulus, one exponentiation takes nearly 2 seconds. It is quite much considering that we have to do several exponentiations in the encryption and in the proofs.

We also want to see the total cost related to the encryption and decryption. We assume that calculating the product and the sum of two elements, and computing the hash of the commitments use negligible time. Thus we only count the exponentiations. The result is presented in Table 6.4 where k_p is the number of parties and k_l is the maximum number of candidates in each party.

Table 6.4: Costs of Calculations.

Operation	Cost
Encryption	$(16 + 10k_l)k_p + 6$
Decryption	$(17 + 9k_l)k_p + 12 + s$

Since we need to verify the proofs before we do the decryption we have included the cost of the verification which is $(17 + 9k_l)k_p + 11$.

Using Tables 6.3 and 6.4 and the numbers from Hordaland 2011 we get the following table for how long time it takes to make one encryption with proofs for $s = 1, 2, \ldots, 7$.

As we see from Table 6.5, it takes a lot of time to make one encryption with proofs. For $s = 7$, the smallest modulus, it takes 2 hours, which is pretty much. For the largest modulus it takes nearly 5 hours.

Table 6.5: Time of the Encryption Process.

s values	Time (ms)	Time (h m)
$s = 1$	17733984	$4h\ 55m$
$s = 2$	11431584	$3h\ 11m$
$s = 3$	9434208	$2h\ 37m$
$s = 4$	8493696	$2h\ 21m$
$s = 5$	7989504	$2h\ 13m$
$s = 6$	7524096	$2h\ 5m$
$s = 7$	7291392	$2h\ 2m$

Chapter 7

Conclusion

We have presented the existing voting protocol Helios, to which we have done two changes. The first change was to add an extra encryption proof, and the second was to change the whole encryption process.

We added another proof for correct encryption in addition to the proof of knowledge. The reason we did this was to make a security proof for ballot secrecy. We were not able to do this before, because we were not able to extract the witness in the non-interactive case. Then we had a problem with the decryption when we removed the secret key in the security proof.

We solved this problem with the new encryption proof by adding $\overline{\alpha} = \overline{g}^r$, where $\overline{g}$ is a generator for the group. This resulted in a complete security proof.

We kept the old proof of knowledge, to still be able to prove that each single ciphertext contains a 0 or a 1 and that the product of ciphertexts contains a value between 0 and $n - 1$.

The other change we did was to combine the already existing exponential ElGamal with a generalisation of Paillier encryption. We wanted to find an efficient voting protocol for the Norwegian county election. The new protocol is a good theoretical protocol for the county election. We have a good way to encrypt ballots with the unique representation L^i for parties and candidates $i \in \{1, 2, \ldots, k\}$ and we are able to prove the encryption correct and make a formal security proof.

Unfortunately, the voting protocol is not efficient. From Table 6.3 in Section 6.5 we see that the time it takes to make just one exponentiation with the largest modulus is nearly 2 seconds. This is quite much considering that we are going to compute three exponents in each ciphertext for each party/candidate and we have to compute some exponentiations in the proofs as well. We see the result of this in Table 6.5. It takes 2 hours to encrypt one ballot for one voter along with corresponding proofs with the smallest modulus. With the largest modulus it takes nearly 5 hours. In both cases the time needed is simply too much for the protocol to be efficient.

Bibliography

[1] Ben Adida. Helios Voting: Technical Documentation, 2012.

[2] Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. *Proceeding of the First ACM Conference on Computer and Communication Security*, pages 62–73, 1993.

[3] Josh Benaloh. Simple Verifiable Elections. *EVT'06: Electronic Voting Technology Workshop*, 2006.

[4] Ronald Cramer. Modular Design of Secure yet Practical Cryptographic Protocols. pages 19–27, 1996.

[5] Ivan B. Damgård and Mads J. Jurik. *A Generalisation, a Simplification and some Applications of Paillier's Probabilistic Public-Key System.* 2000.

[6] PARI/GP Developement. PARI/GP.

[7] Amos Fiat and Adi Shamir. *How to Prove Yourself: Practical Solutions to Identification and Signature Problems.* 1986.

[8] Rosario Gennaro. Achieving Independence Efficiently and Securely. *PODC'95: Principles of Distributed Computing Symposium*, page § 4.2, 1995.

[9] Kristian Gjøsteen. The Norwegian Internet Voting Protocol. 2013.

[10] Kristian Gjøsteen. A Latency-Free Elecion Scheme. *Topics in Cryptology-CT-RSA 2008*, pages 425–436, Springer 2008.

[11] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers, Fourth Edition.* 1975.

[12] Jens Groth Ivan B. Damgård and Gorm Salomonsen. The Theory and Implementation of an Electronic Voting System. July 31, 2002.

[13] Kasper Dupont Ivan B. Damgård and Michael Østergaard Pederesen. Unclonable Group Identification. *Advances in Cryptology-EUROCRYPT 2006*, pages 555–572, Springer 2006.

[14] Kazue Sako and Joe Kilian. *Receipt-Free Mix-Type Voting Scheme - A Practical Solution to the Implementation of a Voting Booth*. 1995.

[15] Statistisk Sentralbyrå. Kommunestyret- og fylkestingsvalget, listekandidater 2011, 1. juli 2011.

[16] Statistisk Sentralbyrå. Kommunestyret- og fylkestingsvalget, personer med stemmerett, 25. august 2011.

[17] Victor Shoup and Rosario Gennaro. Securing Threshold Cryptosystems against Chosen Ciphertext Attack. *Journal of Cryptology*, pages 15(2): 75–96, 2002.

[18] Ben Smyth and Véronique Cortier. Attacking and Fixing Helios: An Analysis of Ballot Secrecy. *CSF' 11: 24th Computer Security Foundations Symposium*, pages 297–311, 2011.