The Norwegian Internet Voting Protocol: A new Instantiation

Kristian Gjøsteen and Anders Smedstuen Lund

Department of Mathematical Sciences, NTNU
{kristag,andelund}@math.ntnu.no

Abstract

The Norwegian government ran trials of internet remote voting during the 2011 municipal elections and the 2013 parliamentary elections. From a simplified version of the voting protocol used there, the essential cryptographic operations of the voting protocol has been put together into a cryptosystem in which one can build the voting protocol on top of.

This paper proposes a new instantiation of the underlying cryptosystem, improving our confidence in the security of the cryptosystem. The new instantiation is mostly similar to a previously defined instantiation, but allows parts of the security proof to be significantly improved.

Keywords: electronic voting protocols, Decision Diffie-Hellman, zero-knowledge.

1 Introduction

During the 2011 municipal elections and the 2013 parliamentary elections in Norway, trials of internet remote voting were conducted. The cryptographic protocol used was design by the Spanish company Scytl, with contributions from Gjøsteen [4, 5].

Gjøsteen [5] abstracted the essential cryptographical operations into a cryptosystem with a set of security notions. He presented an instantiation of this cryptosystem, with corresponding security proofs. In one of the proofs a technical obstruction occurs, based on the fact that the non-interactive version of the Schnorr proof of knowledge for discrete logarithms is not a proof of knowledge. Shoup and Gennaro [6] solved a similar problem, and it turns out the same approach can be used to solve the obstruction Gjøsteen ran into.

Our contribution We describe a new instantiation of the cryptographic protocol satisfying Gjøsteen's requirements for functionality and security, but where the security proof avoids the technical obstruction encountered by Gjøsteen. The new cryptosystem uses the same encryption and transformation methods as Gjøsteen, and is based on the same group structure. The main change is that instead of using a Schnorr proof to "prove knowledge" of the decryption of a ciphertext, the techniques from Shoup and Gennaro [6] are used to achieve the same effect.

Our cryptosystem therefore has a better security proof, at the expense of a minor increase in computational cost. Our cryptosystem is essentially as practical and efficient as Gjøsteen's cryptosystem.

Overview of the paper We start by describing the group structure the cryptosystem is built on in Section 2. We also discuss certain problems related to the group structure, among them the Decision Diffie-Hellman problem.

Then we discuss certain zero knowledge proofs in Section 3. They are needed to prove the security requirements of the system, and are different versions of the equality of discrete logarithms problem in the group described in Section 2.

In Section 4 we describe the cryptosystem, along with the set of security notions, defined in [5]. We continue by giving a new instantiation of the cryptosystem, and then anlyse the security of this new instantiation proving that it satisfies the security notions stated.

Lastly we compare the computational complexity of the two instantiations in Section 5.

Note that we do not discuss the Norwegian Internet Voting protocol, since the description and analysis in [5] apply even when we replace the instantiation in that paper with our novel instantiation.

2 The Underlying Group Structure

To build the cryptosystem (and the voting protocol on top of it), we need some structure on the group we are going to use. We will briefly describe some of the structure here. Gjøsteen [5] has a more detailed description of the group structure.

2.1 The Group Structure

Let q be a prime such that $p = 2q + 1$ is a prime. The group G we are going to use is the group of quadratic residues modulo p, and we let g be a generator of the group G. G has order q.

We also define the set $O = \{1, l_1, l_2, \dots, l_L\}$, where $l_1, \dots, l_L$ are the group elements of G corresponding to the L smallest primes that are quadratic residues modulo p.

2.2 Decision Diffie-Hellman

We need to describe certain computational problems related to the group G. First we describe the Decision Diffie-Hellman problem (DDH) and two related problems. The Decision Diffie-Hellman problem can be described as follows:

Decision Diffie-Hellman. Given $(g_0, g_1) \in G \times G$ (where at least g_1 is sampled at random), decide if $(x_0, x_1) \in G \times G$ was sampled uniformly from the set $\{(g_0^s, g_1^s) \mid 0 \leq s \leq q\}$ or uniformly from $G \times G$.

It has been proven in e.g. [1, 3] that the Decision Diffie-Hellman problem is equivalent to the following problem:

***L*-DDH.** Given $(g_0, \ldots, g_L) \in G^{L+1}$ (where at least $g_1, \ldots, g_L$ is sampled at random), decide if $(x_0, \ldots, x_L) \in G^{L+1}$ was sampled uniformly from the set $\{(g_0^s, \ldots, g_L^s) \mid 0 \leq s \leq q\}$ or uniformly from G^{L+1}.

We will need to use a problem related to the Decision Diffie-Hellman problem. In this problem, one gets up to n challenges and are supposed to figure out from which of the sets described in the L-DDH problem the challenges are sampled from. We present the problem formally:

***n*-*L*-DDH.** Given $(g_0, \ldots, g_L) \in G^{L+1}$ (where at least $g_1, \ldots, g_L$ is sampled at random), and up to n tuples $(x_{i0}, \ldots, x_{iL}) \in G^{L+1}$, decide if these n tuples were sampled uniformly from the set $\{(g_0^s, \ldots, g_L^s) \mid 0 \leq s \leq q\}$ or uniformly from G^{L+1}.

2.3 Subgroup Generated by Small Primes

Now we describe a problem related to the Decision Diffie-Hellman problem, where $g_1, \ldots, g_L$ is replaced by the elements $l_1, \ldots, l_L$ discussed earlier. This gives us the following problem, which we call the Subgroup Generated by Small Primes (SGSP) problem:

Subgroup Generated by Small Primes. Let the prime p be chosen at random from an appropriate range, and let the generator g be chosen at random. Determine the group elements $l_1, \ldots, l_L$ as above. The problem is to decide if $(x_0, \ldots, x_L) \in G^{L+1}$ was sampled uniformly from the set $\{(g_0^s, l_1^s, \ldots, l_L^s) \mid 0 \leq s \leq q\}$ or uniformy from G^{L+1}.

We also define the many challenges version of the SGSP problem:

***L*-SGSP.** Let p be chosen at random from an appropriate range, and let g be a generator chosen at random. Define the group elements $l_1, \ldots, l_L$ as above. Given up to n tuples $(x_{i0}, \ldots, x_{iL}) \in G^{L+1}$, the problem is to decide if these n tuples were sampled uniformly from the set $\{(g^s, l_1^s, \ldots, l_L^s) \mid 0 \leq s \leq q\}$ or uniformly from G^{L+1}.

3 Non-interactive Zero Knowledge Proofs

When analyzing the protocol later in this paper, we will need to force the adversary to do certain computations correctly. Certain non-interactive zero knowledge (NIZK) proofs will help us achieve this. We describe them here.

3.1 Equality of Discrete Logarithms

When encrypting, we want to make sure that the voter's computer, P, computes correctly. It turns out that it will suffice to prove that two elements have the same discrete logarithm relative to distinct generators of the group G.

The proof is between a prover and a verifier. Both the prover and the verifier are given some auxillary information aux, two generators g and $\bar{g}$ and two group elements x and $\bar{x}$. The prover is given t as private input. The prover's algorithm generates a proof $\pi_{\mathcal{E}}$, and the verifer's algorithm takes the proof and the public input and produces 0 or 1.

The prover's and the verifier's algorithms are denoted by

$$\pi_{\mathcal{E}} \leftarrow \mathcal{P}^{I}_{eqdl}(aux, t; g, \bar{g}; x, \bar{x}), and$$
$$0 \text{ or } 1 \leftarrow \mathcal{V}^{I}_{eqdl}(aux; g, \bar{g}; x, \bar{x}; \pi_{\mathcal{E}}).$$

We require completness, that is, any proof generated by an honest $\mathcal{P}$ must be accepted by $\mathcal{V}$.

Instantiation We sketch a fairly standard way of making such a proof [6]. The proof takes the form of a three-way protocol between the prover and the verifier.

Step I The prover first chooses a random number $u \in \mathbb{Z}_q$, and computes $\alpha_1 = g^u$ and $\alpha_2 = \bar{g}^u$. The prover sends (α_1, α_2) to the verifier.

Step II The verifier chooses a random challenge $e \in \mathbb{Z}_q$ and sends to the prover.

Step III The prover computes $\nu = u - et \mod q$, and sends ν to the verifier.

Verification The verifier accepts if and only if

$$\alpha_1 = g^{\nu} x^{e} \quad \text{and} \quad \alpha_2 = \bar{g}^{\nu} \bar{x}^{e}.$$

Applying the Fiat-Shamir heuristic [2] we make it a non-interactive proof. We use a hash function $H_1 : \{0,1\}^* \times G^6 \to \{1, 2, \ldots, 2^{\tau}\}$ evaluated as

$$e \leftarrow H_1(aux, g, \bar{g}, x, \bar{x}, \alpha_1, \alpha_2),$$

to get a non-interactive proof. The proof is $\pi_{\mathcal{E}} = (e, \nu)$. The proof is accepted if and only if

$$e = H_1(aux, g, \bar{g}, x, \bar{x}, g^{\nu} x^{e}, \bar{g}^{\nu} \bar{x}^{e})$$

The cost of generating a proof is two full exponentiations, and the cost of verifying a proof is two full exponentiations (roughly of size $\log_2 q$) and two small exponentiations (roughly of size τ).

Security analysis We require two properties from the proofs given in this section. The proofs must be zero knowledge and it must be hard to generate valid proofs when the discrete logarithms are not equal.

First, we prove that the protocol is special honest-verifier zero knowledge. Given any challenge e, we can choose a random ν and compute

$$\alpha_1 = g^\nu x^e \text{ and } \alpha_2 = \bar{g}^\nu \bar{x}^e,$$

with x and $\bar{x}$ as above, to get α_1 and α_2 with the same distributions as in the real proof. We denote this sampling by

$$\nu \leftarrow Sim^I_{eqdl}(aux, g, \bar{g}, x, \bar{x}, e).$$

The Fiat-Shamir heuristic gives us a non-interactive zero knowledge proof from the simulation above. To generate a proof, first choose a random e, use Sim^I_{eqdl} and then reprogram the H_1 oracle appropriately.

It is now time to look at the soundness of the instantiation, that is we give a bound for the probability that the adversary has of being able to fake a proof for the logarithms being equal when the simulator follows the instantiation given in this section. The bound is given in the following theorem:

Theorem 3.1. *Take any algorithm that outputs a proof $\pi_\mathcal{E} = (e, \nu)$, a bit string aux, a integer t, and group elements $g, \bar{g}, x$ and $\bar{x}$ such that $\log_g x \neq \log_{\bar{g}} \bar{x}$. If the algorithm uses at most η queries to the random oracle H_1, then the probability that*

$$\mathcal{V}^I_{eqdl}(aux; g, \bar{g}; x, \bar{x}; \pi_\mathcal{E}) = 1$$

is at most $(\eta + 1)2^{-\tau+1}$.

To prove the theorem we need the following lemma from [6]:

Lemma 3.2. *Let G be a group of prime order q, and let g and $\bar{g}$ be generators of G. Suppose t and Δ are integers, and $\bar{g}, x, \bar{x}, \alpha_1$ and α_2 are group elements such that $x = g^t$ and $\bar{x} = \bar{g}^t \bar{g}^\Delta$.*

Then if $\Delta \neq 0$ and e is an integer choosen uniformly at random from a set with 2^τ elements, the probability that there exists a integer ν such that

$$\alpha_1 x^{-e} = g^\nu \text{ and } \alpha_2 \bar{x}^{-e} = \bar{g}^\nu$$

is at most $2^{-\tau}$.

Proof of Theorem 3.1. If the algorithm has not queried the H_1 at the relevant point, the proof verifies with probability $2^{-\tau}$.

By Lemma 3.2, every time the algorithm queries H_1 with some input for which he cannot already create a forged proof, the probability that the resulting hash value will allow an attacker to create a forged proof is at most $2^{-\tau}$.

We now have at most $(\eta + 1)$ events, each with a probability at most $2^{-\tau}$. The probability that at least one of them then happens is bounded by $(\eta + 1)2^{-\tau}$. $\square$

3.2 Two More Proofs for Equality of Discrete Logarithms

We will also need two other equality of discrete logarithms proofs. One that proves that several group elements are raised to the same power as a certain element, and one that proves that several elements has been correctly raised to distinct powers.

Both proofs are between a prover and a verifier. The prover's and the verifier's algorithms are respectively

$$\begin{aligned}\pi_{\mathcal{T}I} &\leftarrow \mathcal{P}^{II}_{eqdl}(aux, g, \gamma, s; x, w_1, w_2, \dots, w_k; \check{x}, \check{w}_1, \check{w}_2, \dots, \check{w}_k), \text{ and}\\ 0 \text{ or } 1 &\leftarrow \mathcal{V}^{II}_{eqdl}(aux, g, \gamma; x, w_1, w_2, \dots, w_k; \check{x}, \check{w}_1, \check{w}_2, \dots, \check{w}_k; \pi_{\mathcal{T}I}).\end{aligned}$$

and

$$\begin{aligned}\pi_{\mathcal{T}II} &\leftarrow \mathcal{P}^{III}_{eqdl}(aux, g, y_{21}, y_{22}, \dots, y_{2k}, a_{21}, a_{22}, \dots, a_{2k};\\ &\qquad \check{x}; \hat{w}_1, \hat{w}_2, \dots, \hat{w}_k), \text{ and}\\ 0 \text{ or } 1 &\leftarrow \mathcal{V}^{III}_{eqdl}(aux, g, y_{21}, y_{22}, \dots, y_{2k}; \check{x}; \hat{w}_1, \hat{w}_2, \dots, \hat{w}_k; \pi_{\mathcal{T}II}).\end{aligned}$$

Where the public input to both players is the input described for the verifer's algorithm, and the private input to the prover is the additional input described in the prover's algorithm.

In both cases we require completness, that is, any proof generated by an honest $\mathcal{P}$ must be accepted by $\mathcal{V}$.

Instantiation Instantiations of both proof systems, with corresponding security analysis, are given in [5].

4 The Cryptosystem

We now present our instantiation of the cryptosystem described in [5], along with the corresponding security proof, which is the main contribution of this paper.

For completeness' sake, we must also include the definition of the cryptosystem from [5] along with the security notions. Sections 4.1, 4.2 and 4.3 are thus more or less identical to material found in [5], but contain only material relevant for this paper.

4.1 Preliminaries

Let I be a set of identities, M a set of messages and $O \subseteq M$ a set of options, one of which is the null option denoted by 1_O. A ballot is a k-tuple of options. We denote options by v and a ballot $(v_1, v_2, \dots, v_k)$ by $\vec{v}$.

Let C be a set of *pre-codes*, one of which is the null pre-code denoted by 1_C. We shall have a set S of pre-code maps from M to C such that for every $s \in S$, $s(1_O) = 1_C$. We also need a set of commitments to pre-code maps, one commitment for each map.

We will extend pre-code maps to apply to k-tuples of messages $\vec{m} = (m_1, m_2, \dots, m_k)$ in the obvious way: $s(\vec{m}) = (s(m_1), \dots, s(m_k)) \in C^k$.

4.2 Definition

Our cryptosystem consists of six algorithms and one protocol:

- A *key generation algorithm* $\mathcal{K}$ that outputs a public key ek, a decryption key dk_1, a transformation key dk_2 and a pre-code decryption key dk_3.
- A *pre-code map generation algorithm* $\mathcal{S}$ that on input of a public key ek and an identity V outputs a pre-code map s and a commitment γ to that map.
- An *encryption algorithm* $\mathcal{E}$ that on input of an encryption key ek, an identity $V \in I$ and a message sequence $\vec{m} \in M^k$ outputs a ciphertext c.
- A deterministic *extraction algorithm* $\mathcal{X}$ that on input of a ciphertext c outputs a *naked ciphertext* $\bar{c}$ or the special symbol $\perp$.
- A *transformation algorithm* $\mathcal{T}$ that on input of a transformation key dk_2, an identity $V \in I$, a pre-code map s and a ciphertext c outputs a pre-code ciphertext $\check{c}$ or the special symbol $\perp$.
- A deterministic *pre-code decryption algorithm* $\mathcal{D}_R$ that on input of a pre-code decryption key dk_3, an identity $V \in I$, a pre-code map commitment γ, a ciphertext c and a pre-code ciphertext $\check{c}$ outputs a sequence of pre-codes $\vec{\rho} \in C^k$.
- A *decryption protocol* Π_{DP} between a prover and a verifier. The common input is a public key ek and a sequence of naked ciphertexts $\bar{c}_1, \dots, \bar{c}_k$. The prover's private input is a decryption key dk_1. The number of protocol rounds is independent of both public and private input. The prover and the verifier output either $\perp$ or a sequence of messages $\vec{m}_1, \dots, \vec{m}_k$.

In addition, we require one more algorithm that is only used to define security requirements.

- A *decryption algorithm* $\mathcal{D}$ that on input of a decryption key dk_1, and identity $V \in I$ and a ciphertext outputs a message sequence $\vec{m} \in M^k$ or the special symbol $\perp$.

For the cryptosystem to be useful, it must guarantee correct decryption of ciphertexts and transformed ciphertexts. This is captured in the following completeness requirements:

C1. *For any message and any identity, encryption followed by decryption should return the original message, unchanged.*

For any keys ek, dk_1 output by $\mathcal{K}$, any message $\vec{m}$ and any identity V

$$\Pr[\mathcal{D}(ek, dk_1, \mathcal{E}(ek, V, \vec{m})) = \vec{m}] = 1.$$

C2. *For any sequence of messages, encrypting, extracting and then running the decryption protocol should faithfully reproduce the messages, up to the action of the order map.*

C3. *Transformation of a ciphertext should apply the given pre-code map to the content of the ciphertext.*

For any $\vec{m} \in M^k$ and any $V \in I$, if the following actions happen:

$$(ek, dk_1, dk_2, dk_3) \leftarrow \mathcal{K};\ (s, \gamma) \leftarrow \mathcal{S}(ek, V);$$
$$c \leftarrow \mathcal{E}(ek, V, \vec{m});\ \check{c} \leftarrow \mathcal{T}(dk_2, V, s, c);$$
$$\vec{\rho} \leftarrow \mathcal{D}_R(dk_3, V, \gamma, c, \check{c}).$$

Then $\check{c} \neq \perp$ and $\vec{\rho} = s(\vec{m})$.

4.3 Security Requirements

With this cryptosystem follows a set of fairly natural security notions. For completeness we describe the relevant security notions here inn full, while the other security notions are only described briefly. All security notions are fully described in [5].

D-**Privacy** *Naked ciphertexts should not be correlatable to identities.*

For any $V \in I$ and $\vec{m} \in M^k$, if the following actions happen:

$$(ek, dk_1, dk_2, dk_3) \leftarrow \mathcal{K};\ c \leftarrow \mathcal{E}(ek, V, \vec{m});\ \bar{c} \leftarrow \mathcal{X}(c).$$

Then the distribution of $\bar{c}$ should be independent of V.

B-**Privacy** *An adversary that knows the transformation key should not be able to say anything about the content of any ciphertexts he sees.* We play the following game between a simulator and an adversary, and the probability that the adversary wins should be close to 1/2.

A simulator samples $b \leftarrow \{0, 1\}$ and computes $(ek, dk_1, dk_2, dk_3) \leftarrow \mathcal{K}$.

The adversary gets ek and dk_2, and sends the simulator a sequence of challenge messages $\vec{m}_1^0, \dots, \vec{m}_n^0$, one by one, along with identities $V_1, \dots, V_n$.

When the simulator gets $\{(\vec{m}_i^0, V_i)\}$, it, for $1 \leq i \leq n$, samples a random message $\vec{m}_i^1$ and computes $c_i \leftarrow \mathcal{E}(ek, V_i, \vec{m}_i^b)$. The ciphertexts $c_1, \dots, c_n$ are sent to the adversary.

At any time, the adversary may submit tuples $(V, c, \check{c}, s, \gamma)$ to the simulator. First, the simulator verifies that the s matches γ, and then computes $\vec{m} \leftarrow \mathcal{D}(dk_1, V, c)$ and $\vec{\rho} \leftarrow \mathcal{D}_R(dk_3, V, \gamma, c, \check{c})$. If either decryption fails, $\perp$ is returned to the adversary. If $(V, c) = (V_i, c_i)$ for some i, 1 is returned to the adversary. Otherwise $\vec{\rho}$ is returned to the adversary.

Finally, the adversary outputs $b' \in \{0, 1\}$ and wins if $b = b'$.

R-**Privacy** *An adversary that controls the pre-code decryption key and sees many transformed encryptions of valid ballots from O^k should not be able to say anything non-trivial about the content of those encryptions.* We play the following game between a simulator and an adversary, and the probability that the adversary wins should be close to 1/2.

A simulator samples $b \leftarrow \{0, 1\}$, a random permutation π_1 on O and sets π_0 to be the identity map on O. It computes the keys $(ek, dk_1, dk_2, dk_3) \leftarrow \mathcal{K}$, and sends ek and dk_3 to the adversary. The adversary chooses a challenge identity V, and sends it to the simulator. The simulator computes $(s, \gamma) \leftarrow \mathcal{S}(ek, V)$ and sends γ to the adversary.

The adversary then submits a sequence of ballots $\vec{v}_1, \dots, \vec{v}_n$ from O^k, one by one. The simulator computes $c_i \leftarrow \mathcal{E}(ek, V, \pi_b(\vec{v}_i))$, $\check{c}_i \leftarrow \mathcal{T}(dk_2, V, s, c_i)$ and sends $(c_i, \check{c}_i)$ to the adversary.

Finally, the adversary outputs $b' \in \{0, 1\}$ and wins if $b = b'$.

A**-Privacy** *An adversary that runs the verifier part of the decryption protocol should not be able to correlate ciphertexts with decryptions.*

B**-Integrity** *An adversary that knows all the key material, and chooses the per-voter key material, should not be able to create an identity, a ciphertext and a transformed ciphertext such that the transformed ciphertext decryption is inconsistent with the decryption of the ciphertext.* We play the following game between a simulator and an adversary, and the probability that the adversary wins should be close to 0.

A simulator computes $(ek, dk_1, dk_2, dk_3) \leftarrow \mathcal{K}$, and sends (ek, dk_1, dk_2, dk_3) to the adversary. The adversary then produces a tuple $(V, s, \gamma, c, \check{c})$ which he sends to the simulator. The simulator computes $\vec{m} \leftarrow \mathcal{D}(dk_1, V, c)$ and $\vec{\rho} \leftarrow \mathcal{D}_R(dk_3, V, \gamma, c, \check{c})$.

The adversary wins if $\vec{\rho} \neq \perp$ and either $\vec{m} = \perp$, or $s(\vec{m}) \neq \vec{\rho}$.

D**-Integrity** *An adversary that runs the prover's part of the protocol* Π_{DP} *should not be able to tamper with the decryption.*

4.4 Instantiation

We now present the new instantiation. It is uses ideas presented in [6], but is otherwise similar to the instantiation given in [5]. Let k be the number of options on a ballot.

- The *key generation algorithm* $\mathcal{K}$ chooses $a_{i1}, a_{i2}, \dots, a_{ik} \in_R \mathbb{Z}_q^*$, $i \in \{1, 2\}$. Then it computes, for $1 \leq j \leq k$, $a_{3j} = a_{2j} + a_{1j} \pmod{q}$. Further compute, for $1 \leq j \leq k$, $y_{1j} = g^{a_{1j}}$, $y_{2j} = g^{a_{2j}}$ and $y_{3j} = g^{a_{3j}}$. Lastly we choose a random generator $\bar{g} \in G$. The keys are $ek = (g, \bar{g}, \{y_{1i}\}_{1 \leq i \leq k}, \{y_{2i}\}_{1 \leq i \leq k}, \{y_{3i}\}_{1 \leq i \leq k})$, $dk_1 = \sum_{i=1}^{k} a_{1i}$, $dk_2 = (a_{21}, a_{22}, \dots, a_{2k})$ and $dk_3 = (a_{31}, a_{32}, \dots, a_{3k})$.
- The *pre-code map generation algorithm* $\mathcal{S}$ takes as input ek, V. It chooses $s \in_R \{1, \dots, q-1\}$, and computes $\gamma = g^s$.

  It then outputs (s, γ).

- The *encryption algorithm* $\mathcal{E}$ takes as input $ek, V, \vec{v} \in M^k$. It chooses $t \in_R \mathbb{Z}_q$ and computes $x = g^t$, $\bar{x} = \bar{g}^t$ and $w_i = y_{1i}^t v_i$, for $1 \leq i \leq k$, and then computes

$$\pi_{\mathcal{E}} = \mathcal{P}^{I}_{eqdl}(V||x||w_1||w_2||\ldots||w_k, g, t; g, \bar{g}; x, \bar{x}).$$

  It outpus the ciphertext $c = (V, x, \bar{x}, w_1, w_2, \ldots, w_k, \pi_{\mathcal{E}})$.

- The *deterministic extraction algorithm* $\mathcal{X}$ takes as input c. It verifies $\pi_{\mathcal{E}}$ and computes $\bar{w} = w_1 \cdot w_2 \cdots w_k$.

  It outputs $\bar{c} = (x, \bar{w})$.

- The *transformation algorithm* $\mathcal{T}$ takes as input dk_2, V, s, c. It verifies $\pi_{\mathcal{E}}$, computes $\check{x} = x^s$, $\hat{w}_i = \check{x}^{a_{2i}}$ and $\check{w}_i = w_i^s$, for $1 \leq i \leq k$, and generate proofs

$$\pi_{\mathcal{T}I} \leftarrow \mathcal{P}^{II}_{eqdl}(c, g, s; x, w_1, w_2, \ldots, w_k; \check{x}, \check{w}_1, \check{w}_2, \ldots, \check{w}_k),$$
$$\pi_{\mathcal{T}II} \leftarrow \mathcal{P}^{III}_{eqdl}(c, g, \hat{w}; a_{21}, a_{22}, \ldots, a_{2k}; \hat{w}_1, \hat{w}_2, \ldots, \hat{w}_k).$$

  It outputs $\check{c} = (\check{x}, \hat{w}_1, \hat{w}_2, \ldots, \hat{w}_k, \check{w}_1, \check{w}_2, \ldots, \check{w}_k, \pi_{\mathcal{T}I}, \pi_{\mathcal{T}II})$.

- The *pre-code decryption algorithm* $\mathcal{D}_R$ takes as input $dk_3, V, \gamma, c, \check{c}$. It verifies the proofs $\pi_{\mathcal{E}}$, $\pi_{\mathcal{T}I}$ and $\pi_{\mathcal{T}II}$, and, if accepted, computes $\rho_i = \check{w}_i \hat{w}_i \check{x}^{-a_{3i}} (= v_i^s)$, for $1 \leq i \leq k$.

  It outputs $\vec{\rho} = (\rho_1, \rho_2, \ldots, \rho_k)$.

- The *decryption protocol* Π_{DP} is not described in this paper because the focus of the paper is on the ballot submission part of the e-voting system.

- The *decryption algorithm* $\mathcal{D}$ takes as input dk_1, V, c. It verifies $\pi_{\mathcal{E}}$, and computes $m_i = w_i x^{-a_{1i}}$, for $1 \leq i \leq k$.

  It outputs $(m_1, m_2, \ldots, m_k)$.

  Remark: Here we must have $dk_1 = (a_{11}, a_{12}, \ldots, a_{1k})$ instead of $dk_1 = \sum_{i=1}^{k} a_{1i}$.

4.5 Security Proof

We now show that the instantiation given satisfies the requested completeness and security requirements. We will start with the completeness requirements. It is clear by inspection that the first Completeness requirement C1 holds. Since we have not described a decryption protocol Π_{DP}, we are not able to prove Completeness requirement C2.

C3 Completeness

The zero-knowledge proofs are in this case complete, and ElGamal is homomorphic. Hence the following equation proves that this requirement is fullfiled:

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

After proving the completeness requirements we now prove our instantiation fullfills the required security notions. Again, since we do not have a decryption protocol, we are unable to prove A-privacy or D-integrity.

D-Privacy

The identity V is not used in the creation of $(w_1, w_2, \dots, w_k)$ or x, hence the distribution of the ciphertext $\bar{c}$ is independent of V.

B-Privacy

We prove the following lemma giving a bound for the advantage an adversary can get against B-privacy:

Lemma 4.1. *If Decision Diffie-Hellman is (T, ϵ_{DDH})-hard and $T < 2^{\tau/2} - 1$, then any adversary against B-privacy using time at most T has advantage at most $\frac{Tn}{q} + 2^{-\tau/2+2} + k2^{-\tau} + k\epsilon_{DDH}$.*

We prove the lemma by giving a sequence of games, and for every two consecutive games bounding the probability of an adversary distinguishing between the two games. In the final game we end up in a situation where we obviously achieve B-privacy, therefore by suming up the bounds we get a bound for the advantage of an adversary against B-privacy.

Game 1 The first game is identical to the game between a simulator and an adversary used to define B-privacy. We assume the game requires time at most T and that the simulator receives at most n challenge ciphertexts.

For game i, we define the event E_i as the event that the adversary correctly guesses the bit b. The advantage of the adversary is then

$$\epsilon = |\Pr[E_1] - 1/2|.$$

Game 2 We make two changes for Game 2. The first change is that the simulator now remembers the ballot encrypted when encrypting honestly generated ballots. Secondly, we now use the Sim^I_{eqdl} (described in Section 3.1) to fake the proofs for $\log_g x = \log_{\bar{g}} \bar{x}$, and reprogram the random oracle accordingly.

The adversary can only notice these changes if the reprogramming of the random oracle fails. This happens if the oracle has been queried with the same input before the reprogramming attempt. There are at most n reprogramming attempts, and at most T random oracle queries. At each query, a random integer from $\mathbb{Z}_q$ is chosen, and hence the probabiltiy that any one reprogramming attempt fails is bounded above by $\frac{T}{q}$. Therefore we see that

$$|\Pr[E_2] - \Pr[E_1]| \leq \frac{Tn}{q}.$$

Game 3 Now, when the simulator decrypts adversarially generated ciphertexts, in addition to producing $\vec{\rho}$ and $\vec{m}$, he computes $\rho'_i = m_i^s$, for $1 \leq i \leq k$. Instead of returning $\vec{\rho}$, he returns $\vec{\rho'}$.

The only way the adversary can notice this is if $\rho_i \neq \rho'_i$ for some i, which will only happen if either $\pi_{\mathcal{T}I}$ or $\pi_{\mathcal{T}II}$ are forgeries.

The probability of this happening is bounded above by the probability that the adversary is able to forge one or both proofs for one decryption query. This is the same situation as for B-integrity, and from [5] we get that

$$|\Pr[E_3] - \Pr[E_2]| \leq 2^{-\tau/2+2}.$$

Game 4 Now we stop computing with the secret keys $a_{31}, \ldots, a_{3k}$. Nothing the adversary sees is dependent on computations where these keys are used, hence the changes are not observable. Therefore

$$\Pr[E_4] = \Pr[E_3].$$

We want to start to randomize the votes before encryption. To do this we need the following lemma:

Lemma 4.2. *Consider Game 4 modified such that when encrypting, we let either the $i-1$ first coordinates or the i first coordinates of each challenge vote $\vec{v}$ be random group elements. If there are n challenge votes $\vec{v} = (v_1, v_2, \ldots, v_k)$, the advantage of any adversary using time at most T trying to decide whether $i-1$ or i coordinates are random is at most $(T+1)2^{-\tau+1} + \epsilon_{DDH}$.*

Proof. We prove the lemma by making a sequence of games, and giving bounds for the probability an adversary has in distinguishing between each two consecutive games.

Game i This game is the same as Game 4 described before the lemma, but when encrypting each challenge vote $\vec{v}$, $v_1, \ldots, v_{i-1}$ is replaced by random group elements. Hence $w_1, \ldots, w_{i-1}$ are encryptions of random group elements for each challenge vote $\vec{v}$.

Game ii In the key generation phase, we change the generation of one group element. Now we generate $\bar{g}$ by picking a random number $r \in \mathbb{Z}_q^*$ and computing $\bar{g} = y_{1i}^r$. We observe that now $\bar{x}^{1/r} = \bar{g}^{t/r} = y_{1i}^{tr/r} = y_{1i}^t$.

Notice that y_{1i} is never the identity, and since any element except the identity in G is a generator, $\bar{g}$ has the same distribution as before. So we have

$$\Pr[E_{ii}] = \Pr[E_i].$$

Game iii Now, for adversarially generated ciphertexts we decrypt c_i by using the observation made in the previous game. That is, we compute $m_i = w_i \bar{x}^{-1/r}$. For all $1 \leq j \leq k$, $j \neq i$, we decrypt c_j as before.

This change can only be noticed if the adversary is able to fake at least one proof of $\log_g x = \log_{\bar{g}} \bar{x}$. The adversary can make at most T queries to the random oracle, so by Theorem 3.1 the probability of faking at least one such proof is $(T+1)2^{-\tau+1}$, which means that

$$|\Pr[E_{iii}] - \Pr[E_{ii}]| = (T+1)2^{-\tau+1}.$$

Game iv In this game we let y_{1i} be randomly generated from G, and we don't generate a_{1i}. Let $\vec{\zeta} = (g, \bar{g}, y_{11}, y_{12}, \ldots, y_{1k})$. After the key generation we sample a random tuple $(z_0, \bar{z}_0, z_1, \ldots, z_k)$ from the subgroup generated by $\vec{\zeta}$. When encrypting a honestly generated ballot, we sample an additional random number t' from $\mathbb{Z}_q$, and encrypt by $x = g^t z_0^{t'}$, $\bar{x} = \bar{g}^t \bar{z}_0^{t'}$ and $w_j = y_{1j}^t z_j^{t'} m_j$, $1 \le j \le k$.

This change cannot be observed by the adversary, since the distribution of the elements still are the same. Therefore we get that

$$\Pr[E_{iv}] = \Pr[E_{iii}].$$

Game v Let $\vec{\zeta}'$ be the $(k+2)$-tuple that is the same as $\vec{\zeta}$, except that the ith coordinate is replaced by a random element from G. Now, after the key generation, we sample the tuple $(z_0, \bar{z}_0, z_1, \ldots, z_k)$ from the subgroup generated by $\vec{\zeta}$ and $\vec{\zeta}'$.

The only difference between this and the previous game, is the distribution of the z_i-values. We are now able to make a distinguisher that will give an advantage on the Decision Diffie-Hellman problem with (g, y_{1i}) as base, if we have an adversary that has an advantage in distinguishing between this and the previous game. The distinguisher (which we omit) proves the following lemma:

Lemma 4.3. *If the Decision Diffie-Hellman is (T, ϵ_{DDH})-hard, then*

$$|\Pr[E_v] - \Pr[E_{iv}]| \le \epsilon_{DDH}.$$

Game vi In this game we let m_i be a random element from G, instead of the real m_i. We generate $\bar{g}$ and y_{1i} as in Game i. Also we go back to encrypting as we do in Game i.

The encryption w_i in Game v is just a random group element. Since m_i is a random element from G in this game, w_i will still be a random element from G. Therefore the change of m_i cannot be noticed. Clearly all other changes are indistinguishable for the adversary, so

$$\Pr[E_{vi}] = \Pr[E_v].$$

Analysis After Game vi we are in the situtation of Game i with coordinate i of all challenge votes randomised. Lemma 4.2 now follows from the triangle inequality. □

Game 5 We now randomize all the coordinates in every vote $\vec{v}$.

We randomize the coordinates one by one, applying Lemma 4.2 each time, always keeping already randomised coordinates still randomised. From the lemma we get that

$$|\Pr[E_5] - \Pr[E_4]| \le k((T+1)2^{-\tau+1} + \epsilon_{DDH}).$$

Analysis After Game 5 the ciphertexts no longer contains any information about the ballot, and hence the adversary cannot decide which ballot was encrypted.

The triangle inequality gives us that

$$\epsilon = |\Pr[E_1] - 1/2| \leq \frac{Tn}{q} + 2^{-\tau/2+2} + k(T+1)2^{-\tau+1} + k\epsilon_{DDH}.$$

R-Privacy

The proof that the instantiation given in this article achieves R-privacy is similar to the proof given in [5]. The only difference is in the three first games. The details of the three first games in our case are given in Appendix A.

B-Integrity

The pre-code decryption algorithm verifies that $\pi_{\mathcal{E}}$ accepts, hence it cannot be the case that c does not decrypt while $\check{c}$ decrypts.

It is a reasonable assumption to assume that the number of random oracle queries an algorithm makes is bounded by the time T the algorithm uses to execute. Now the only way the adversary can win, is if he has faked at least one of the proofs $\pi_{\mathcal{T}I}$ and $\pi_{\mathcal{T}II}$. If $T < 2^{\tau/2} - 1$, then by Proposition 1 and 2 in [5] the proofs $\pi_{\mathcal{T}I}$ and $\pi_{\mathcal{T}II}$ can be forged with probability at most $2^{-\tau/2+1}$ and $2^{-\tau/2+1}$, respectively. Hence the probability that $\rho_i \neq v_i^s$ for any i is at most $2^{-\tau/2+1} + 2^{-\tau/2+1} = 2^{-\tau/2+2}$.

5 Comparison

In this section we compare the computational complexity of this instantiation with the instantiation described by Gjøsteen in [5]. We distinguish between full exponentiations and small exponentiations, the small exponentiations are used in the equality of discrete logarithms proofs. Let N be the number of voters in the election, k the maximum number of options any one ballot can have and n_{tot} the number of votes given by all voters.

We first present the number of exponentiations for the instantiation given in Gjøsteens paper:

	Full exponentiations	*small exponentiations*
$\mathcal{E}$:	$(k+2)$ (per vote)	
$\mathcal{X}$:	N	N
$\mathcal{T}$:	$(2k+7)n_{tot}$	$(k+2)n_{tot}$
$\mathcal{D}_R$:	$(k+5)n_{tot}$	$(4k+7)n_{tot}$

Secondly, for the instantiation presented in Section 4, we have the following number of exponentiations:

	Full exponentiations	*small exponentiations*
$\mathcal{E}$:	$(k+4)$ (per vote)	
$\mathcal{X}$:	$2N$	$2N$
$\mathcal{T}$:	$(2k+8)n_{tot}$	$(k+3)n_{tot}$
$\mathcal{D}_R$:	$(k+6)n_{tot}$	$(4k+8)n_{tot}$

When discussing this we will divide into what happens during and after the election, and into what the voter's computer P does and what the central system does.

During the election we run the encryption algorithm $\mathcal{E}$ on the voter's computer. The transformation algorithm $\mathcal{T}$ and the pre-code decryption algorithm $\mathcal{D}_R$ runs on the central system, and will be considered together.

After the election we need to run the extraction algorithm $\mathcal{X}$ to get the naked ciphertexts that are sent for decryption.

The computational complexity after the election increases with a factor of 2. This small increase in complexity after the election should not be of any concern, especially since this will not be noticed by the voters using the system. Furthermore, since the ballot box has essentially verified these proofs as part of the transformation algorithm, there's little need to verify them again as part of the extraction algorithm. Which means that this increase in complexity only affects the auditor.

We now look at the change in computational complexity during the election. This is also the most important phase, since it is during this phase the voters will use and notice how well the system works. We see that for the encryption algorithm, we have a increase by two exponentiations per vote given. Since each vote is given on a distinct computer, this is the interesting number for the encryption algorithm. This increase is not very significant, and should not be very noticable for a voter.

The most crucial part of the system when looking at the computational complexity is the combined complexity of the transformation algorithm $\mathcal{T}$ and the pre-code decryption algorithm $\mathcal{D}_R$ that does the task of receiving the votes from the computer of the different voters and transforming them into pre-codes that again can be turned into receipt codes.

From the above tables one sees that the increase in complexity is for each of the algorithms by $2n_{tot}$ full exponentiations and $2n_{tot}$ small exponentiations, which gives a total increase of $4n_{tot}$ full exponentiations and $4n_{tot}$ small exponentiations for both of them combined. It is important to take in the factor of n_{tot} since many votes from different voters (on different computers) can be submitted simultanously. However, the increase is minor compared to the total workload.

6 Conclusion

We have described the cryptosystem underlying the Norwegian Internet voting protocol, and given a new instantiation for it together with a security analysis of the instantiation. The new instantiation solves the technical problem with the proof of knowledge for the encryption exponent encountered in the instantiation given in [5].

After describing the instantiation and its security analysis we also analysed the computational complexity of the new instantiation. The computational complexity increases with this new instantiation, but also has increased security, as mentioned above. The system in our opinion will still be practical and quite effective with the new instantiation.

References

[1] Ivan Damgård, Kasper Dupont, and Michael Østergaard Pedersen. Unclonable group identification. In *Advances in Cryptology-EUROCRYPT 2006*, pages 555–572. Springer, 2006.

[2] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, pages 186–194, 1986.

[3] Kristian Gjøsteen. A latency-free election scheme. In *Topics in Cryptology - CT-RSA 2008, The Cryptographers' Track at the RSA Conference 2008, San Francisco, CA, USA, April 8-11, 2008. Proceedings*, pages 425–436, 2008.

[4] Kristian Gjøsteen. Analysis of an internet voting protocol. Cryptology ePrint Archive, Report 2010/380, 2010. `http://eprint.iacr.org/`.

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

[6] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *J. Cryptology*, 15(2):75–96, 2002.

A R-privacy

We prove that the same bound as Gjøsteen gave in [5] for the advantage of an adversary against R-privacy also is achieved with our instantiation. The following theorem was stated in the article by Gjøsteen:

Lemma A.1. *If the SGSP problem and the DDH problem is, respectively (T, ϵ_{SGSP})-hard and (T, ϵ_{DDH})-hard, then any adversary against R-privacy using at most N voter identities and time at most T has advantage at most*

$$\epsilon \leq \frac{3n_{tot}T}{q} + L\epsilon_{SGSP} + Nq^{-L} + k\epsilon_{DDH},$$

To prove the lemma we do the following sequence of indistinguishable games:

Game 1 In the first game, everything is as in the game for R-privacy. We assume the game requires time at most T and the adversary submits at most n_{tot} ballots.

The advantage of the adversary is given as

$$\epsilon = |\Pr[E_1] - 1/2|.$$

Game 2 In this game, every proof that we have to generate is faked. We do this by using the simulators given in Section 3 and then reprograming the oracles appropriately.

The adversary cannot notice this change unless a reprograming attempt fails. That happens if the oracle has already be queried at the point that the simulator tries to reprogram. It is reasonable to assume that the number of queries to the random oracle is bounded by the time T, and each reprograming attempt involves a group element chosen uniformly at random. Therefore the probability that a single reprograming attempt fails is bounded by T/q. Since there are at most $3n_{tot}$ reprogramming attempts, we have that

$$|\Pr[E_2] - \Pr[E_1]| = \frac{3n_{tot}T}{q}.$$

Game 3 In the key generation phase, we generate a pick a random number u, and generate $\bar{g}$ as $\bar{g} = g^u$. Also we from now one always compute $\bar{x}$ as $\bar{x} = x^u$ $(= \bar{g}^t)$.

Since g is a generator, $\bar{g}$ will have the same distribution as earlier. Also, $\bar{x}$ still is computed correctly, hence these changes are indistinguishable and

$$\Pr[E_3] = \Pr[E_2]$$

We are now in the same position as before Game 3 in the proof for R-privacy in [5]. The same set of games as Gjøsteen applies in Game 3 to Game 8 can be applied here, and thus we achieve the same bound for R-privacy as Gjøsteen, as proposed.