The Norwegian Internet Voting Protocol

Kristian Gjøsteen*

August 1, 2013

Abstract

The Norwegian government ran a trial of internet remote voting during the 2011 local government elections, and will run another trial during the 2013 parliamentary elections. A new cryptographic voting protocol will be used, where so-called return codes allow voters to verify that their ballots will be counted as cast.

This paper discusses this cryptographic protocol, and in particular the ballot submission phase.

The security of the protocol relies on a novel hardness assumption similar to Decision Diffie-Hellman. While DDH is a claim that a random subgroup of a non-cyclic group is indistinguishable from the whole group, our assumption is related to the indistinguishability of certain special subgroups. We discuss this question in some detail.

Keywords: electronic voting protocols, Decision Diffie-Hellman.

1 Introduction

The Norwegian government ran a trial of internet remote voting during the 2011 local government elections. During the advance voting period, voters in 10 municipalities were allowed to vote from home using their own computers. This form of voting made up a large majority of advance voting. The Norwegian goverment will run a second trial of remote voting during the 2013 parliamentary elections.

Internet voting, and electronic voting in general, faces a long list of security challenges. For Norway, the two most significant security problems with internet voting will be compromised voter computers and coercion.

Coercion will be dealt with by allowing voters to revote electronically. Revoting cancels previously submitted ballots. Also, the voter may vote once on paper, in which case every submitted electronic ballot is canceled, even those submitted after the paper ballot submission. In theory, almost everyone should therefore have sufficient tools to avoid coercion.

*kristian.gjosteen@math.ntnu.no, Department of Mathematical Sciences, Norwegian University of Science and Technology

This leaves compromised computers as the main remaining threat. Since a significant fraction of home computers are compromised, the voting system must allow voters to detect ballot tampering without relying on computers. This is complicated by the fact that voters are unable to do even the simplest cryptographic processing without computer assistance.

Norwegian municipal elections are somewhat complicated. The voter chooses a party list, he is allowed to give personal votes to candidates on the list, and he is allowed to amend the list by adding a certain number of candidates from other party lists. Parliamentary elections are also somewhat complicated. The voter again chooses a party list, but he is now allowed to reorder candidates or strike them out.

Essentially, a ballot consists of a short, variable-length sequence of options (at most about a hundred options) chosen from a small set of possible options (at most a few thousand). Note that the entire sequence is required to properly interpret and count the ballot. For municipal elections, order within the sequence does not matter, but for parliamentary elections order does matter.

We note in Norway paper ballots submitted in an election are considered sensitive and access is restricted. One reason for this is that because of the complex ballots, many distinct ballots will have essentially the same effect on the election result. Therefore, it is possible to mark ballots, which means that if the counted ballots were public, anyone could reliably buy votes.

Related work We can roughly divide the literature into protocols suitable for voting booths [7, 8, 25, 26], and protocols suitable for remote internet voting [2, 9, 18, 20, 23], although protocols often share certain building blocks. One difference is that protocols for voting booths should be both coercion-resistant and voter verifiable, while realistic attack models (the attacker may know more than the voter knows) for remote internet voting probably make it nearly impossible to achieve both voter verifiability and coercion-resistance.

For internet voting protocols, we can again roughly divide the literature into two main strands distinguished by the counting method. One is based on homomorphic tallying. Ballots are encrypted using a homomorphic cryptosystem, the product of all the ciphertexts is decrypted (usually using some form of threshold decryption) to reveal the sum of the ballots. For simple elections, this can be quite efficient, but for Norwegian elections this becomes unwieldy.

The other strand has its origins in mix nets [5]. Encrypted ballots are sent through a mix net. The mix net ensures that the mix net output cannot be correlated with the mix net input. There are many types of mixes, based on nested encryption [5] or reencryption, verifiable shuffles [15, 23] or probabilistic verification [3, 18], etc. These can be quite efficient, even for the Norwegian elections.

Much of the literature ignores the fact that a voter simply will not do any computations. Instead, the voter delegates computations to a computer. Unfortunately, a voter's computer can be compromised, and once compromised may modify the ballot before submission.

One approach to defend against compromised computers is so-called preencrypted ballots and return codes [6, 4], where the voter well in advance of the election receives a table with candidate names, identification numbers and return codes. The voter inputs a candidate identification number to vote and receives a response. The voter can verify that his vote was correctly received by checking the response against the printed return codes. In Norway, preencrypted ballots will be too complicated, but return codes can still be used.

Note that unless such systems are carefully designed, privacy will be lost. Clearly, general multiparty computation techniques can be used to divide the processing among several computing nodes (presumably used by [6]). In practice, few independent data centres are available that have sufficient quality and reputation to be used in elections. This means that general multiparty computation techniques are not so useful.

One approach for securely generating the return codes is to use a proxy oblivious transfer scheme [16, 17]. A ballot box has a database of return codes and the voter's computer obliviously transfers the correct one to a messenger, who then sends the return code to the voter. The main advantage of this approach is that very few computing nodes are required. Unfortunately, this particular solution is probably too computationally expensive to be used for Norwegian elections.

Another useful tool is the ability for out-of-band communication with voters [21]. This allows us to give the voter information directly, information that his computer should not know and not be able to tamper with. The scheme in [16, 17] sends return codes to the voter out-of-band. This helps ensure that a voter is notified whenever a vote is recorded, preventing a compromised computer from undetectably submitting ballots on the voter's behalf.

Our contribution The cryptographic protocol to be used in Norway is designed by Scytl, a Spanish electronic voting company, with contributions by the present author. It is mostly a fairly standard internet voting system based on ElGamal encryption of ballots and a mix-net before decryption.

The system works roughly as follows (see Figure 1). The *voter* V gives his ballot to a *computer* P, which encrypts the ballot and submits it to a *ballot box* B. The ballot box and a *return code generator* R cooperate to compute a sequence of return codes for the submitted ballot. These codes are sent by SMS to the voter's mobile phone F. The voter verifies the return codes against a list of precomputed option–return code pairs printed on his voting card.

Once the ballot box closes, the submitted ciphertexts are decrypted by a *decryptor* D. An *auditor* A supervises the entire process.

The main contribution of the current author to the protocol, and the focus of this paper, is a novel method for computing the return codes efficiently. We use the fact that exponentiation is in some sense a pseudo-random function [10, 12], and since ElGamal is homomorphic, exponentiation can be efficiently done "inside" the ciphertext.

The mechanics of the Norwegian electoral system means that we must en-

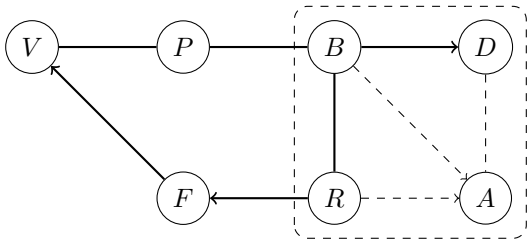

Figure 1: Overview of the protocol players and communication channels.

crypt each option separately in order to generate return codes for each option separately. If every ballot consisted of up to 100 ElGamal ciphertexts, mixing would be prohibitively expensive. Therefore, Scytl uses a clever encoding of options [22, 24] to allow the protocol to compress ciphertexts before mixing and then recover the complete ballot after decryption. However, this encoding forces us to do a more careful security analysis of the return code computation.

The main advantage of our contribution is that generating return codes is cheap, amounting to just a few exponentations. At the same time, mixing is reasonably fast. Since the protocol requires very few players to achieve reasonable security, we have a protocol that is deployable in practice.

The protocol in this paper is an improved version of the protocol used in the 2011 trials [13]. The main difference is that the protocol in this paper uses multi-ElGamal and more efficient NIZK proofs to get a significant performance improvement. The protocol analysis in this paper is also a significant improvement on the analysis done on the previous protocol. The protocol that will be used in the 2013 trials is a minor modification of the protocol described in this paper.

Overview of the paper The cryptographic protocol is essentially based on ElGamal encryptions. Section 2 describes the group structure used for ElGamal and the special properties we require of it to be able to compress many ciphertexts into one, and still recover complete ballots from the decryption. It then defines and discusses a conjecturally hard problem on this group structure, a problem that is similar to Decision Diffie-Hellman and required for the security of the system.

Section 3 defines and analyses certain NIZK proofs required by the cryptosystem. These are used to prove that various computations are correct. We also need a proof of knowledge. Modeling this proof and its security is non-trivial, and we spend some time on it in this section and in Appendix A.

Section 4 defines a cryptosystem with corresponding security notions, an instantiation of this cryptosystem, and relates the security of this instantiation to the conjecturally hard problem discussed in Section 2. This cryptosystem encapsulates the essential cryptographic operations in the voting protocol.

Section 5 then shows how we build the cryptographic voting protocol on top of the cryptosystem defined in Section 4, and how the security of the voting

protocol essentially follows from the security of the cryptosystem, as well as the assumed properties of various infrastructures.

2 The Underlying Group Structure

Underlying the entire voting protocol is a group structure with certain very specific properties, needed to be able to compress ElGamal ciphertext. The basic idea is that from the product of not too many small primes computed modulo a large prime, the small primes can still be recovered efficiently by trial division. These small primes then lead to a problem similar to Decision Diffie-Hellman.

2.1 The Group Structure

Let q be a prime such that $p = 2q + 1$ is also a prime. Then the quadratic residues of the finite field with p elements is a finite cyclic group G of prime order q. Let g be a generator.

Let $\ell_1, \ell_2, \ldots, \ell_L$ be the group elements corresponding to the L smallest primes that are quadratic residues modulo p. Let $O = \{1, \ell_1, \ell_2, \ldots, \ell_L\}$.

2.2 Factoring

Factoring products of small primes is efficient. Suppose that all of these primes are smaller than $\sqrt[k]{p}$ for some integer $0 < k$. If we select k elements from O and multiply them, then we can efficiently recover the k elements from the product, up to order. If we use the obvious ordering on the group elements, we get a map from the set of all such products to O^k, and this map can be extended to a map $\phi : G \to G^k$ by taking any other group element x to the k-tuple $(1, \ldots, 1, x)$.

If we care about the ordering of the tuples, we must use a different approach. A tuple $(v_1, \ldots, v_k)$ is mapped to the product

$$\prod_{i=1}^{k} v_i^i.$$

Again, we can efficiently recover the prime powers in the product to see which options were included, and in what order. Note that this approach significantly reduces the length k of the tuples that can be encoded.

2.3 A Subgroup Problem

We are interested in a problem related to the prime p and the elements $\ell_1, \ell_2, \ldots, \ell_L$. We begin our discussion with the usual Decision Diffie-Hellman (DDH) problem, which can be formulated 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 $\{(g_0^s, g_1^s) \mid 0 \leq s < q\}$ or uniformly from $G \times G$.*

It is well-known (e.g. [10, 12]) that this is equivalent to the following problem:

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

Remark. An adversary for the former problem trivially becomes an adversary for the latter problem.

Given $(g_0, g_1), (x_0, x_1) \in G \times G$, we can create L tuples

$$(y_i, z_i) = (g_0^{r_i} x_0^{t_i}, g_1^{r_i} x_1^{t_i}), i = 1, 2, \ldots, L,$$

for random r_i, t_i. If $g_1 = g_0^u$ and $(x_0, x_1) = (g_0^s, g_1^s)$, then $z_i = y_i^u$. If $(x_0, x_1) = (g_0^s, g_1^v)$, $v \not\equiv s \pmod{q}$, then z_i and y_i are uniformly random and independent.

It follows that an adversary for the latter problem becomes an adversary for the former problem. We remark on this fact because in contrast to the obvious hybrid argument, this argument does not reduce the adversary's advantage by $1/L$.

However, suppose the subgroup is generated by small primes instead of random group elements. We get 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 $\ell_1, \ell_2, \ldots, \ell_L$ as above. The problem is to decide if $(x_0, x_1, \ldots, x_L) \in G^{L+1}$ was sampled uniformly from the set $\{(g^s, \ell_1^s, \ldots, \ell_L^s) \mid 0 \leq s < q\}$ or uniformly from G^{L+1}.*

While this problem is very similar to Decision Diffie-Hellman, and indeed cannot be hard unless Decision Diffie-Hellman is hard, it seems unlikely that its hardness follows from hardness of Decision Diffie-Hellman.

For a prime p of practical interest, a reasonably large L will make it feasible to find a single non-trivial relation among the small primes, which is enough to decide where $(x_0, \ldots, x_L)$ was sampled from. However, when L is small, as it will be for our application, this may not be true.

It is generally believed that the best way to solve the Decision Diffie-Hellman is to compute one of the corresponding discrete logarithms.

It is known [19] that solving the static Diffie-Hellman problem with a fairly large number of oracle queries is easier than solving the discrete logarithm problem. We can consider the challenge tuple $(x_0, \ldots, x_L)$ as the answers to certain oracle queries, so any adversary against the SGSP problem could in some sense be considered an adversary against static Diffie-Hellman. Therefore, our problem should not be easier than the static Diffie-Hellman problem.

For fairly large L, a static Diffie-Hellman solver could be applied to decide the SGSP problem. This would be faster than the fastest known solver for the Decision Diffie-Hellman problem in the same group. However, for our application, L will always be small, hence a static Diffie-Hellman solver can not be

directly applied. A hybrid approach could perhaps be deployed, but for small L such an approach should not be significantly faster than simply computing a discrete logarithm.

Remark. Note that it will probably be possible to choose the prime p *together* with a relation among the small primes. Given such a relation, the decision problem above will be easy, since the relation will hold for prime powers as well. It is therefore important for our purposes that the prime p is chosen verifiably at random. There are straight-forward ways to do this.

2.4 Further Analysis

While the SGSP problem discussed above is sufficient for our purposes, it is not necessary. A weaker, sufficient condition would be if, given a permutation of a subset of $\{\ell_1^s, \ldots, \ell_L^s\}$ and g^s for some random s, it was hard to deduce any information about which primes were involved and what the permutation was.

We study the case when there are only two elements, say ℓ_0 and ℓ_1, and the subset contains one of them. Let A be an algorithm that takes as input five group elements and outputs 0 or 1. Define

$$\begin{aligned}
\pi_{00} &= \Pr[A(\ell_0, \ell_1, g, g^s, \ell_0^s) = 0], \\
\pi_{11} &= \Pr[A(\ell_0, \ell_1, g, g^s, \ell_1^s) = 1], \text{ and} \\
\pi_{i,\mathrm{rnd}} &= \Pr[A(\ell_0, \ell_1, g, g^s, g^t) = i],\ i \in \{0, 1\},
\end{aligned}$$

where s and t are sampled uniformly at random from $\{0, 1, \ldots, q-1\}$. Note that $\pi_{0,\mathrm{rnd}} = 1 - \pi_{1,\mathrm{rnd}}$, since the input distribution to A is identical for both probabilities.

We may define the advantage of A as $|\pi_{00} + \pi_{11} - 1|$. Observe that if $|\pi_{00} - \pi_{0,\mathrm{rnd}}|$ or $|\pi_{11} - \pi_{1,\mathrm{rnd}}|$ are large, we have a trivial solver for Decision Diffie-Hellman with the generator fixed to either ℓ_0 or ℓ_1.

We may assume that $\pi_{00} + \pi_{11} - 1 = 2\epsilon > 0$. Then either $\pi_{00} \geq 1/2 + \epsilon$ or $\pi_{11} \geq 1/2 + \epsilon$, so assume the former. Furthermore, let $\pi_{00} - \pi_{0,\mathrm{rnd}} = \mu$. If $|\mu| \geq \epsilon$, we have an adversary against Decision Diffie-Hellman with the generator fixed to ℓ_0, so assume $|\mu| < \epsilon$. Then

$$\pi_{11} - \pi_{1,\mathrm{rnd}} = 1 + 2\epsilon - \pi_{00} - (1 - \pi_{0,\mathrm{rnd}}) = 2\epsilon - \mu \geq \epsilon,$$

which means that we must have an adversary with advantage at least ϵ against Decision Diffie-Hellman with the generator fixed to either ℓ_0 or ℓ_1.

The same arguments applies to an algorithm that can recognize one out of multiple elements. It must lead to a successful adversary against Decision Diffie-Hellman with the generator fixed to one of the elements.

Unfortunately, the above argument breaks down if the algorithm is allowed to see multiple elements raised to the same power, that is, if given $\{\ell_i^s \mid i \in I\}$ for some small index set I, the algorithm can decide what I is.

2.5 Further variants

Consider a group G^{L+1} and a generator $(g_0, \ldots, g_L)$ for a subgroup of G^{L+1}.

First, suppose we choose L further elements uniformly and independently at random from G^{L+1}. Note that the probability that these $L+1$ elements do not generate the entire group G^{L+1} is

$$1 - \prod_{i=1}^{L}(1 - 1/q^{L+1-i}) \approx \frac{1}{q}.$$

We can safely ignore this probability.

Suppose we have an algorithm that decides if L elements from G^{L+1} have been sampled uniformly at random from the entire group, or from the subgroup generated by $(g_0, \ldots, g_L)$.

A standard hybrid argument shows that such an algorithm can be converted into one that decides if a single element in G^{L+1} has been sampled uniformly at random from the entire group, or from the subgroup. This means that, up to a reduction of advantage by a factor $1/L$, we are free to assume L sampled elements.

This is important, since if we have $L+1$ elements chosen uniformly at random from G^{L+1}, we can (except with probability $1/q$, which we can safely ignore) span the entire group with a linear combination of our $L+1$ elements.

We get the following alternative variation of the Decision Diffie-Hellman problem.

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

If DDH is (T, ϵ_{DDH})-hard, then this alternative problem is $(T, \epsilon_{DDH} L)$-hard.

Remark. We shall assume that n is larger than L, in which case this bound is better than the bound $(T, \epsilon_{DDH} n)$-hard that we get from standard hybrid arguments.

If we replace the randomly chosen $g_1, \ldots, g_L$ by $\ell_1, \ldots, \ell_L$, we get a many-challenges version of the Subgroup Generated by Small Primes problem, and if the SGSP problem is (T, ϵ_{SGSP})-hard, then the many-challenges problem is $(T, \epsilon_{SGSP} L)$-hard.

3 Non-interactive Zero Knowledge Proofs

In our analysis, we need to force the adversary to prove that he knows the content of a ciphertext and that certain exponentiations have been done correctly. For this, we use non-interactive zero knowledge proofs. The instantiations of these proofs are fairly standard. We include certain proofs taken from [1] for completeness sake.

On (prove, aux, g, x, t) from honest U:

1. Verify that $x = g^t$.
2. $\pi_{\text{pok}} \leftarrow \mathcal{P}_{pok}(aux; g; x; t)$.
3. Send (proof, π_{pok}) to U.

On (verify, $aux, g, x, \pi_{\text{pok}}$) from honest U:

1. If $\mathcal{V}_{pok}(aux; g; x; \pi_{\text{pok}}) = 1$, send (verified) to U.
2. Otherwise, send (reject) to U.

Figure 2: Protocol for proof of knowledge of one discrete logarithm. Note that verifying the correctness of the given discrete logarithms in the proof generation step is superfluous in the complete protocol.

3.1 Proof of knowledge

Once we have produced a group element by exponentiation, we will need to be able to prove that we know the corresponding discrete logarithm, that is, we will need a proof of knowledge. Furthermore, this proof must be tied to certain auxillary information.

The public input is some auxillary information aux, one generator g for the group G and one group element x. The prover's private input is t such that $x = g^t$. The prover's algorithm generates a proof π_{pok}, and the verifier's algorithm takes the proof and the public input and either accepts or rejects.

We denote the proof generation and verification algorithms by

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

The corresponding protocol is given in Figure 2. As usual, we require completeness, in that any proof created by $\mathcal{P}_{pok}$ must be accepted by $\mathcal{V}_{pok}$.

Instantiation We sketch one example of such a proof of knowledge based on batch verification. The description takes the form of a three-move protocol between a prover and a verifier, and then applies the Fiat-Shamir heuristic to get a suitable non-interactive system.

The prover chooses u at random, computes $\alpha = g^u$ and sends u to the verifier.

The verifier chooses a random challenge β and sends it to the prover.

The prover computes

$$\nu \leftarrow r - t\beta \bmod q.$$

and sends ν to the verifier.

The verifier accepts if and only if

$$\alpha \stackrel{?}{=} g^{\nu} x^{\beta}.$$

The Fiat-Shamir heuristic with a hash function $H : \{0,1\}^* \times G^3 \to \mathbb{Z}$ evaluated as

$$\beta \leftarrow H(aux, g, x, \alpha)$$

gives us a non-interactive zero knowledge proof $\pi_{\text{pok}} = (\beta, \nu)$, verified by checking that

$$\beta \stackrel{?}{=} H(aux, g, x, g^{\nu} x^{\beta}).$$

The cost of generating a proof is one full exponentiation. The cost of verifying a proof is one full exponentiation and one short exponentiation.

Security Analysis It is clear that this protocol is special honest verifier zero knowledge, since for any challenge β, we can choose a random ν and get a uniformly random α from the verification equation. This provides us with a simulated proof with the same distribution as the real proof. We denote this sampling by

$$(\alpha, \nu) \leftarrow Sim_{pok}(aux, g, x, \beta).$$

Under the Fiat-Shamir heuristic, special honest verifier zero knowledge gives us non-interactive zero knowledge. To generate a proof, we first choose a random challenge β, use Sim_{pok} and then reprogram the random oracle to return the random challenge when queried with α.

The proof that the interactive protocol is a proof of knowledge is done by rewinding. Once the adversary has produced two correct responses for two challenges to the same commitment, an easy computation recovers the discrete logarithm.

Rewinding is more difficult after we apply the Fiat-Shamir heuristic, and no sound argument is known for why the non-interactive version really is a proof of knowledge.

However, in the generic group model (see Appendix A), it is possible to prove that the non-interactive Schnorr proofs are proofs of knowledge. In the real world constructing a valid proof without knowing the requisite discrete logarithms definitely seems to be hard. It therefore seems reasonable to assume that anyone who can reliably create valid proofs can also produce the requisite discrete logarithms, even if we cannot explicitly construct an extractor.

We claim that the ideal functionality described in Figure 3 is a reasonable interpretation of the security afforded by the Schnorr proofs. Note first that the functionality never generates a proper proof, but instead uses Sim_{pok} and informs the ideal adversary about the generated proof (allowing a simulator to properly simulate the random oracle). This means that proofs are zero knowledge. Second, whenever an honest player attempts to verify a proof that wasn't generated by the functionality, the functionality informs the ideal adversary and accepts only if the ideal adversary responds with the requisite discrete logarithms. This means that proofs are proofs of knowledge.

We want to phrase this claim in the usual language of universal composability, but this is surprisingly tricky. In the traditional formulation, an environment

On (prove, aux, g, x, t) from honest U:

1. Verify that $x = g^t$.
2. Choose random challenge β, and compute

$$(\alpha, \nu) \leftarrow Sim_{pok}(aux, g, x, \beta).$$

3. Let $\pi_{\text{pok}} = (\beta, \nu)$. Store $(aux, g, x, \pi_{\text{pok}})$, send (proof, π_{pok}) to U and hand over (proved, $\pi_{\text{pok}}, aux, g, x, \beta$) to $\mathcal{A}$.

On (verify, $aux, g, x, \pi_{\text{pok}}$) from honest U:

1. If $(aux, g, x, \pi_{\text{pok}})$ is stored, send (verified) to U.
2. Otherwise, hand over (verify, $aux, g, x, \pi_{\text{pok}}$) to $\mathcal{A}$, and wait for (verify, t) or (reject).
3. In the former case, verify that $x = g^t$, store $(aux, g, x, \pi_{\text{pok}})$ and send (verified) to U.
4. In the latter case, send (reject) to U.

Figure 3: Ideal functionality for proof of knowledge of discrete logarithms, parameterized on the group G and the simulator Sim_{pok}.

and an adversary interacts with the protocol. The natural idea would be to replace the adversary by some simulator. However, the boundary between the environment and adversary is not well-defined. Nothing prevents the environment from forging the proof and then passing it to the adversary. Indeed the adversary may even be a so-called dummy adversary.

We use the alternative UC formulation given in [14], where there is no explicit adversary. Replacing the environment does not actually make sense, but adding some bookkeeping that extracts the discrete logarithm from the environment could make sense.

Many settings (and in particular, our setting) where an adversary tries to forge a Schnorr proof will correspond to a two-part environment, where one part of the environment actually knows the discrete logarithm, while another part is trying to forge a proof of knowledge. In this situation, it is of course trivial to add bookkeeping code to the environment, but such a modified environment is clearly useless.

We use the vague phrase "morally equivalent" environment to mean an environment with additional bookkeeping code that does not resort to such trivial solutions. The bookkeeping code must recover the discrete logarithm by looking just at the part of the environment trying to forge the proof, not the part of the environment that should know the discrete logarithm.

We make the following, rather vague assumption about the security of the protocol given in Figure 2. We have no proof that the assumption is sound.

Schnorr proof assumption. *Any environment interacting with the protocol described Figure 2 (instantiated as above with a random oracle), can be replaced by a "morally equivalent" environment interacting with the functionality in Fig-*

ure 3 and some ideal simulator $\mathcal{A}$ (that among other things simulates the random oracle and extracts the appropriate discrete logarithms from the modified environment).

If the original interaction requires time at most T, then the new interaction requires time at most χT for some small χ.

We shall also assume that any environment's advantage ϵ_{pok} in distinguishing the two cases is small.

Remark. The time requirements are motivated both in the generic model proof (Appendix A) and the fact that no attacks against Schnorr signatures better than computing discrete logarithms are known.

Remark. Fischlin [11] gives similar non-interactive proofs with online extractors. Based on these, it should be easy to realize the functionality, which means that the analysis in Section 4.4 works unchanged. However, Fischlin's proofs are significantly more costly than Schnorr proofs.

Another, less expensive, alternative is the techniques Gennaro and Shoup [28] used to build public key encryption schemes with distributed decryption. While these will probably be more costly than Schnorr proofs, they will definitely be cheaper than Fischlin's proofs. However, adopting these techniques will probably require rewriting the analysis in Section 4.4.

3.2 Proof of correct computation I

We need to prove that we have raised a number of group elements to the same power.

The public input is some auxillary information aux, one generator g, a commitment γ and $2k$ group elements $x_1, x_2, \ldots, x_k, \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k$. The prover's private input is an integer s such that $\gamma = g^s$ and $\bar{x}_i = x_i^s$, $i = 1, 2, \ldots, k$. The prover's algorithm generates a proof π_{eqdl}, and the verifier's algorithm takes the proof and the public input and either accepts or rejects.

We denote the proof generation and verification algorithms by

$$\pi_{\text{eqdl}} \leftarrow \mathcal{P}_{eqdl}(aux, g, s; x_1, x_2, \ldots, x_k; \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k), \text{ and}$$
$$0 \text{ or } 1 \leftarrow \mathcal{V}_{eqdl}(aux, g, \gamma; x_1, x_2, \ldots, x_k; \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k; \pi_{\text{eqdl}}).$$

As usual, we require completeness, in that any proof created by $\mathcal{P}_{eqdl}$ must be accepted by $\mathcal{V}_{eqdl}$.

Instantiation We sketch one example of such a proof based on batch verification [1]. The description takes the form of a four-move protocol between a prover and a verifier, and then applies the Fiat-Shamir heuristic to get a suitable non-interactive system.

The verifier chooses random $\beta_1, \ldots, \beta_k$ and sends them to the prover.

The prover computes $x = \prod_{i=1}^{k} x_i^{\beta_i}$, chooses u at random, computes $\alpha_1 = g^u$, $\alpha_2 = x^u$ and sends (α_1, α_2) to the verifier.

The verifier chooses random β and sends it to the prover.

The prover computes $\nu \leftarrow u - s\beta \bmod q$, and sends ν to the verifier.
The verifier computes

$$x = \prod_{i=1}^{k} x_i^{\beta_i} \quad \text{and} \quad \bar{x} = \prod_{i=1}^{k} \bar{x}_i^{\beta_i},$$

and accepts if and only if

$$\alpha_1 \stackrel{?}{=} g^{\nu}\gamma^{\beta} \quad \text{and} \quad \alpha_2 \stackrel{?}{=} x^{\nu}\bar{x}^{\beta}.$$

The Fiat-Shamir heuristic with hash functions $H_1 : \{0,1\}^* \times G^{2k+2} \to \mathbb{Z}^k$ and $H_2 : \{0,1\}^* \times G^{2k+4} \to \mathbb{Z}^k$ evaluated as

$$\begin{aligned}(\beta_1, \beta_2, \ldots, \beta_k) &\leftarrow H_1(aux, g, \gamma, x_1, x_2, \ldots, x_k, \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k), \text{ and} \\ \beta &\leftarrow H_2(aux, g, \gamma, x_1, x_2, \ldots, x_k, \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k, \alpha_1, \alpha_2),\end{aligned}$$

gives us a non-interactive proof. The proof is $\pi_{\text{eqdl}} = (\beta, \nu)$, and it is accepted if and only if

$$\beta \stackrel{?}{=} H_2(aux, g, \gamma, x_1, x_2, \ldots, x_k, \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k, g^{\nu}\gamma^{\beta}, x^{\nu}\bar{x}^{\beta}).$$

The cost of generating a proof is two full exponentiations and k short exponentiations. The cost of verifying a proof is two full exponentiations and $2k+2$ short exponentiations.

Security Analysis We shall require two properties. First, the proofs must be zero knowledge, in the sense that there must exist a simulator that generates appropriate proofs without knowledge of the secret s. Furthermore, it must be hard to generate valid proofs when the discrete logarithms are not equal. The latter requirement can be formalized in a game where the adversary presents the public input, the claimed discrete logarithm and a proof. The adversary wins if the discrete logarithm is incorrect for at least one value.

It is clear that the protocol is special honest verifier zero knowledge, since for any challenge $\beta_1, \ldots, \beta_k$ and β, we can choose a random ν and get properly distributed α_1, α_2 using

$$\alpha_1 = g^{\nu}\gamma^{\beta} \text{ and } \alpha_2 = x^{\nu}\bar{x}^{\beta}$$

with x and $\bar{x}$ as above. This provides us with a simulated proof with the same distribution as the real proof. We denote this sampling by

$$\nu \leftarrow Sim_{eqdl}(aux, g, \gamma, x_1, \ldots, x_k, \bar{x}_1, \ldots, \bar{x}_k, \beta_1, \ldots, \beta_k, \beta).$$

The Fiat-Shamir heuristic gives us a non-interactive zero knowledge proof. To generate a proof, we first choose a random β, query the H_1 hash oracle to get $\beta_1, \ldots, \beta_k$, use Sim_{eqdl}, and then reprogram the H_2 oracle appropriately.

These non-interactive proofs are also sound in the random oracle model, in the sense that unless the discrete logarithms are all equal, a very large number of hash queries is required to produce a proof that incorrectly verifies.

Proposition 1. *For any algorithm that makes at most $\eta < 2^{\tau/2} - 1$ queries to the random oracles H_1 and H_2 and outputs a proof $\pi_{\text{eqdl}} = (\beta, \nu)$, a bit string aux, an integer s, and group elements $g, \gamma, x_1, \ldots, x_k, \bar{x}_1, \ldots, \bar{x}_k$ such that $x_i^s \neq \bar{x}_i$ for some i, then*

$$\mathcal{V}_{eqdl}(aux, g, \gamma; x_1, \ldots, x_k; \bar{x}_1, \ldots, \bar{x}_k; \pi_{\text{eqdl}}) = 1$$

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

We need three minor results. The first shows that when sampling each coordinate from any sufficiently large subset of a finite field, the resulting vector is unlikely to be confined to any proper subspace. The second shows that random linear combinations are unlikely to satisfy a discrete logarithm relation unless the elements combined originally satisfy the relation. The third proves that the usual equality of discrete logarithms proof is sound in our sense.

Lemma 1. *Let V be any proper subspace of $\mathbb{F}_q^k$, and let S be a subset of $\mathbb{F}_q$ with 2^τ elements. Sample $\beta_1, \ldots, \beta_k$ independently and uniformly at random from S. Then the probability that the vector $(\beta_1, \beta_2, \ldots, \beta_k)$ lies inside V is at most $2^{-\tau}$.*

Proof. Let pr_i be the projection onto the first i coordinates, $i = 0, 1, 2, \ldots, k$. For some i, the image of V under both pr_{i-1} and pr_i has dimension $i - 1$. This means any choice of $\beta_1, \ldots, \beta_{i-1}$ corresponds to a vector in V, but the ith coordinate in any such vector is fully determined by the first $i - 1$ coordinates.

Since the values $\beta_1, \ldots, \beta_i$ are sampled independently, the probability that β_i equals the value determined by the first $i - 1$ values is at most $2^{-\tau}$. □

Lemma 2. *Let G be a group of prime order q, and let g be a generator. Let S be a subset of $\{0, 1, \ldots, q - 1\}$ with 2^τ elements. Suppose $s, \Delta_1, \ldots, \Delta_k$ are integers such that $s \not\equiv 0 \pmod{q}$ and $\Delta_j = 0$ for at least one j. Finally, let $x_1, \ldots, x_k, \bar{x}_1, \ldots, \bar{x}_k$ group elements such that $\bar{x}_i = x_i^s g^{\Delta_i}$, $i = 1, \ldots, k$.*

If $\Delta_i \not\equiv 0 \pmod{q}$ for any i, and $\beta_1, \ldots, \beta_k$ are integers chosen independently and uniformly at random from a set with 2^τ elements, then

$$\prod_{i=1}^{k} \bar{x}_i^{\beta_i} = \left(\prod_{i=0}^{k} x_i^{\beta_i} \right)^s \tag{1}$$

holds with probability at most $2^{-\tau}$.

Proof. The equation $\sum_{i=1}^k \Delta_i \beta_i \equiv 0 \pmod{q}$ defines a proper subspace of $\mathbb{F}_q^k$.

If (1) holds, then $\prod_{i=1}^k g^{\Delta_i \beta_i} = 1$, or $\sum_{i=1}^k \Delta_i \beta_i \equiv 0 \pmod{q}$. Therefore (1) holds only if $(\beta_1, \ldots, \beta_k)$ considered as an $\mathbb{F}_q$-vector falls inside a proper subspace of $\mathbb{F}_q^k$. The claim then follows by Lemma 1. □

Lemma 3. *Let G be a group of prime order q, and let g be a generator. Suppose s and Δ are integers and $\gamma, x, \bar{x}, \alpha_1, \alpha_2$ are group elements such that $\gamma = g^s$ and $\bar{x} = x^{s+\Delta}$.*

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

$$\alpha_1 \gamma^\beta = g^\nu \text{ and } \alpha_2 \bar{x}^\beta = x^\nu$$

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

Proof. Let $\alpha_1 = g^u$ and $\alpha_2 = x^{u+\delta}$. The requirements on ν then translate into

$$u + \beta s \equiv \nu \pmod{q} \qquad \text{and} \qquad u + \delta + \beta s + \beta\Delta \equiv \nu \pmod{q}.$$

We see that no integer can satisfy these equations unless

$$\delta + \beta\Delta \equiv 0 \pmod{q}.$$

Since both δ and Δ are fixed before β is chosen, the probability that this equation holds is at most $2^{-\tau}$. □

Proof of Proposition 1. If the algorithm has not already queried both H_1 and H_2 at the relevant points, the proof verifies correctly with probability $2^{-\tau}$.

By Lemma 2, every time the algorithm queries H_1, the probablity that the resulting hash value will allow an attacker to create a forgery is at most $2^{-\tau}$.

By Lemma 3, every time the algorithm queries H_2 for some input for which he cannot already create a forged proof, the probability that the result hash value will allow an attacker to create a forgery is at most $2^{-\tau}$.

We now have at most $\eta + 1$ events, each with probability at most $2^{-\tau}$, and $\eta + 1 < 2^{\tau/2}$. We can bound the probability that at least one of them happen. Let $\delta = 2^{-\tau}$. Then

$$\begin{aligned}
1 - (1-\delta)^{\eta+1} &= 1 - \sum_{i=0}^{\eta+1} \binom{\eta+1}{i} (-1)^i \delta^i = \sum_{i=1}^{\eta+1} \binom{\eta+1}{i} (-1)^i \delta^i \\
&\leq (\eta+1)\delta + \sum_{i=2}^{\eta+1} \binom{\eta+1}{i} \delta^i \leq (\eta+1)\delta + \sum_{i=2}^{\eta+1} (\eta+1)^i \delta^i \\
&= (\eta+1)\delta + \sum_{i=2}^{\eta+1} ((\eta+1)\delta)^i \leq (\eta+1)\delta + \sum_{i=2}^{\eta+1} (\sqrt{\delta})^i \\
&\leq (\eta+1)\delta + \sum_{i=2}^{\eta+1} \delta \leq (\eta+1)\delta + (\eta-1)\delta,
\end{aligned}$$

which completes the proof. □

Note that a better proof and a sharper result is probably possible, but the result is sufficient for our needs.

3.3 Proof of correct computation II

We shall also need to prove that a single group element has been raised to correct, distinct powers.

The public input is some auxillary information aux, one generator g, a set of commitments $y_1, \ldots, y_k$, the base $\bar{x}$ and the and the powers $\hat{w}_1, \ldots, \hat{w}_k$. The prover's private input are integers $a_1, \ldots, a_k$. The prover's algorithm generates a proof π'_{eqdl}, and the verifier's algorithm takes the proof and the public input and either accepts or rejects.

We denote the proof generation and verification algorithms by

$$\pi'_{\text{eqdl}} \leftarrow \mathcal{P}'_{eqdl}(aux, g, \bar{x}; a_1, \ldots, a_k; \hat{w}_1, \ldots, \hat{w}_k) \text{ and}$$
$$0 \text{ or } 1 \leftarrow \mathcal{V}'_{eqdl}(aux, g, \bar{x}; y_1, \ldots, y_k; \hat{w}_1, \ldots, \hat{w}_k; \pi'_{\text{eqdl}}).$$

As usual, we require completeness, in that any proof created by $\mathcal{P}'_{eqdl}$ must be accepted by $\mathcal{V}'_{eqdl}$.

Instantiation We sketch one example of such a proof based on batch verification [1]. The description takes the form of a four-move protocol between a prover and a verifier, and then applies the Fiat-Shamir heuristic to get a suitable non-interactive system.

The verifier chooses random $\beta_1, \ldots, \beta_k$ and sends them to the prover.

The prover computes $y = \prod_{i=1}^{k} y_i^{\beta_i}$, chooses u at random, computes $\alpha_1 = g^u$, $\alpha_2 = y^u$, and sends (α_1, α_2) to the verifier.

The verifier chooses random β and sends it to the prover.

The prover computes $\nu \leftarrow u - \beta \sum_{i=1}^{k} \beta_i a_i \bmod q$ and sends ν to the verifier.

The verifier computes

$$y = \prod_{i=1}^{k} y_i^{\beta_i} \qquad \text{and} \qquad \hat{w} = \prod_{i=1}^{k} \hat{w}_i^{\beta_i},$$

and accepts the proof if and only if

$$\alpha_1 \stackrel{?}{=} g^{\nu} y^{\beta} \quad \text{and} \quad \alpha_2 \stackrel{?}{=} \bar{x}^{\nu} \hat{w}^{\beta}.$$

The Fiat-Shamir heuristic with hash functions $H_1 : \{0,1\}^* \times G^{2k+2} \to \mathbb{Z}^k$ and $H_2 : \{0,1\}^* \times G^{2k+4} \to \mathbb{Z}$ evaluated as

$$(\beta_1, \ldots, \beta_k) \leftarrow H_1(aux, g, \bar{x}, y_1, \ldots, y_k, \hat{w}_1, \ldots, \hat{w}_k), \text{ and}$$
$$\beta \leftarrow H_2(aux, g, \bar{x}, y_1, \ldots, y_k, \hat{w}_1, \ldots, \hat{w}_k, \alpha_1, \alpha_2)$$

gives us a non-interactive proof. The proof is $\pi'_{\text{eqdl}} = (\beta, \nu)$, and it is accepted if and only if

$$\beta \stackrel{?}{=} H_2(aux, g, \bar{x}, y_1, \ldots, y_k, \hat{w}_1, \ldots, \hat{w}_k, g^{\nu} y^{\beta}, \bar{x}^{\nu} \hat{w}^{\beta}).$$

The cost of generating a proof is two full exponentiations and k short exponentiations. The cost of verifying a proof is two full exponentiations and $2k + 2$ short exponentiations.

Security Analysis The security properties we require are as for the previous proof.

It is clear that the protocol is special honest verifier zero knowledge, since for any challenge $\beta_1, \ldots, \beta_k$ and β, we can choose a random ν and get properly distributed α_1, α_2 using

$$\alpha_1 = g^{\nu} y^{\beta} \text{ and } \alpha_2 = \bar{x}^{\nu} \hat{w}^{\beta}$$

with y and $\hat{w}$ as above. This provides us with a simulated proof with the same distribution as the real proof. We denote this sampling by

$$\nu \leftarrow Sim'_{eqdl}(aux, g, \bar{x}, y_1, \ldots, y_k, \hat{w}_1, \ldots, \hat{w}_k, \beta_1, \ldots, \beta_k, \beta).$$

The Fiat-Shamir heuristic gives us a non-interactive zero knowledge proof. To generate a proof, we first choose a random β, query the H_1 hash oracle to get $\beta_1, \ldots, \beta_k$, use Sim'_{eqdl}, and then reprogram the H_2 oracle appropriately.

The soundness result is essentially the same as the one stated in Proposition 1.

Proposition 2. *For any algorithm that makes at most $\eta < 2^{\tau/2} - 1$ queries to the random oracles H_1 and H_2 and outputs a proof $\pi_{\text{eqdl}} = (\beta, \nu)$, a bit string aux, integer $a_1, \ldots, a_k$, and group elements $g, y_1, \ldots, y_k, \bar{x}, \hat{w}_1, \ldots, \hat{w}_k$ such that $y_i = g^{a_i}$, $i = 1, 2, \ldots, k$, but $\bar{x}_i^{a_i} \neq \hat{w}_i$ for some i, then*

$$\mathcal{V}'_{eqdl}(aux, g, \gamma; x_1, \ldots, x_k; \bar{x}_1, \ldots, \bar{x}_k; \pi_{\text{eqdl}}) = 1$$

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

The proof of the above proposition is identical to the proof of Proposition 1, except that the reference to Lemma 2 is replaced by a reference to the following result.

Lemma 4. *Let G be a group of prime order q, and let g be a generator. Suppose $a_1, \ldots, a_k, \Delta_1, \ldots, \Delta_k$ are integers and $\bar{x}, y_1, \ldots, y_k, \hat{w}_1, \ldots, \hat{w}_k$ group elements such that $y_i = g^{a_i}$ and $\hat{w}_i = \bar{x}^{a_i} g^{\Delta_i}$, $i = 1, \ldots, k$.*

If $\Delta_i \not\equiv 0 \pmod{q}$ for any i, and $\beta_1, \ldots, \beta_k$ are integers chosen independently and uniformly at random from a set with 2^{τ} elements, then the probability that an integer a exists such that

$$\prod_{i=0}^{k} y_i^{\beta_i} = g^a \text{ and } \prod_{i=1}^{k} \hat{w}_i^{\beta_i} = \bar{x}^a \tag{2}$$

holds with probability at most $2^{-\tau}$.

Proof. Suppose the left half of (2) holds. Then

$$\prod_{i=1}^{k} g^{\Delta_i \beta_i} = 1$$

or $\sum_{i=1}^{k} \Delta_i \beta_i \equiv 0 \pmod{q}$. Therefore (2) only holds if $(\beta_1, \ldots, \beta_k)$ considered as an $\mathbb{F}_q$-vector falls inside a previously defined, proper subspace of $\mathbb{F}_q^k$. The claim then follows by Lemma 1. □

4 The Cryptosystem

In order to simplify the analysis of the Norwegian internet voting protocol, we isolate the most essential cryptographic operations into a cryptosystem that can be considered in isolation. We can then later use the security properties of the cryptosystem to reason about the voting protocol's security.

We briefly summarize the relevant cryptographic operations.

Before an election can be run, keys must be generated and the per-voter option–return code correspondence must be set up.

During ballot submission, the voter V's computer P encrypts the voters' ballot into a ciphertext that is tied to the voter's identity. The ballot box B transforms this ciphertext into a new ciphertext that contains pre-codes, a half-way step between options and return codes. The return code generator R decrypts this ciphertext in order to get the pre-codes, which it will turn into human-readable return codes.

During counting, the ballot box extracts "naked" ciphertexts that cannot be tied to individual voters. The decryptor D decrypts the naked ciphertexts and convinces the auditor A that the decryptions are correct.

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 $(v_1, v_2, \ldots, v_k)$ 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 extend pre-code maps to k-tuples of messages $\vec{m} = (m_1, m_2, \ldots, m_k)$ in the obvious way: $s(\vec{m}) = (s(m_1), \ldots, s(m_k)) \in C^k$.

We also require a total order $\prec$ on the set of options, and a canonical ordering map on ballots $\omega : O^k \to O^k$ such that for any ballot $\vec{v}$, if $\omega(\vec{v}) = (v_1, v_2, \ldots, v_k)$, then for any $1 \leq i \leq j \leq k$, $v_i \prec v_j$. This map is extended in some way to a map $\omega : M^k \to M^k$.

For elections where option order is relevant, we let ω be the identity map.

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 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 an identity V and a ciphertext c outputs a *naked ciphertext* $\tilde{c}$ 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 $\tilde{c}_1, \ldots, \tilde{c}_n$. 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 $\vec{m}_1, \ldots, \vec{m}_n$.

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 $\perp$.

Such a cryptosystem cannot be useful unless it guarantees correct decryption of ciphertexts and transformed ciphertexts. We capture this with 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.*

For any message and identity sequences $\vec{m}_1, \vec{m}_2, \ldots, \vec{m}_n$ and $V_1, V_2, \ldots, V_n$, if the following actions happen:

> $(ek, dk_1, dk_2, dk_3) \leftarrow \mathcal{K}$; for i from 1 to n: $c_i \leftarrow \mathcal{E}(ek, V_i, \vec{m}_i)$, $\tilde{c}_i \leftarrow \mathcal{X}(V_i, c_i)$; the protocol Π_{DP} is run with ek and $(\tilde{c}_1, \ldots, \tilde{c}_n)$ as public input and dk_1 as the prover's private input.

Then the prover and verifier in the protocol both output the same sequence of messages, and that sequence is a permutation of the sequence $\omega(\vec{m}_1), \ldots, \omega(\vec{m}_n)$.

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

We define a set of fairly natural notions of security for the cryptosystem, relating to privacy and integrity.

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:

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

Then the distribution of $\tilde{c}$ 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 a sequence of challenge messages $\vec{m}_1^0, \vec{m}_2^0, \ldots, \vec{m}_n^0$ to the simulator, one by one, along with identities $V_1, V_2, \ldots, V_n$.

When the simulator gets $(\vec{m}_i^0, V_i)$, $1 \leq i \leq n$, it samples a random message $\vec{m}_i^1$, computes $c_i \leftarrow \mathcal{E}(ek, V_i, \vec{m}_i^b)$ and sends c_i 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, 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 $c \neq c_i$ for any i, if $\vec{m} \neq \perp \neq \vec{\rho}$, $(\vec{m}, \vec{\rho})$ is returned to the adversary. If $c = c_i$ for some i, then $\vec{\rho}$ is returned to the adversary. Otherwise, $\perp$ 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 $(ek, dk_1, dk_2, dk_3) \leftarrow \mathcal{K}$. The adversary gets ek and dk_3, and chooses a challenge identity V. The simulator computes $(s, \gamma) \leftarrow \mathcal{S}(ek, V)$ and sends γ to the adversary.

The adversary then submits a sequence of ballots $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n$ 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.* 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, then chooses two sequences of identities $V_1, \ldots, V_{n'}$, $V'_1, \ldots, V'_{n''}$ and corresponding messages $\vec{m}_1, \ldots, \vec{m}_{n'}$, $\vec{m}^{(0)}_1, \ldots, \vec{m}^{(0)}_{n''}$, for some $n', n'' < n$.

The simulator sets π_0 to be the identity map on $\{1, 2, \ldots, n'\}$, and samples a random permutation π_1 on $\{1, 2, \ldots, n'\}$ and a sequence of random messages $\vec{m}^{(1)}_1, \ldots, \vec{m}^{(1)}_{n''}$. Then the simulator computes $c'_i \leftarrow \mathcal{E}(ek, V'_i, \vec{m}^{(b)}_i)$ for $i = 1, 2, \ldots, n''$ and $c_i \leftarrow \mathcal{E}(ek, V_i, \vec{m}_{\pi_b(i)})$, $\tilde{c}_i \leftarrow \mathcal{X}(V_i, c_i)$ for $i = 1, 2, \ldots, n'$, sends $c'_1, \ldots, c'_{n''}, c_1, \ldots, c_{n'}$ to the adversary and runs the prover part of the protocol Π_{DP} with appropriate input against the adversary's verifier.

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

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 $(ek, dk_1, dk_2, dk_3) \leftarrow \mathcal{K}$. The adversary gets (ek, dk_1, dk_2, dk_3), then produces a tuple $(V, s, \gamma, c, \check{c})$. 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.* 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 $(ek, dk_1, dk_2, dk_3) \leftarrow \mathcal{K}$. The adversary gets ek and dk_1, then chooses a sequence of identities $V_1, \ldots, V_n$ and messages $\vec{m}_1, \ldots, \vec{m}_n$.

The simulator computes $c_i \leftarrow \mathcal{E}(ek, V_i, \vec{m}_i)$, $\tilde{c}_i \leftarrow \mathcal{X}(V_i, c_i)$, $i = 1, 2, \ldots, n$, sends $\tilde{c}_1, \ldots, \tilde{c}_L$ to the adversary and runs the verifier part of the protocol Π_{DP} with appropriate input against the adversary's prover.

The adversary wins if the verifier run ouputs a sequence of messages that is not a permutation of $\omega(\vec{m}_1), \ldots, \omega(\vec{m}_n)$.

Remark. The cryptosystem is a convenient abstraction of one part of the voting protocol. Section 5 shows that security of the voting protocol follows from the above security properties and other security measures in the protocol.

A cryptosystem $(\mathcal{K}, \mathcal{S}, \mathcal{E}, \mathcal{X}, \mathcal{T}, \mathcal{D}_R, \Pi_{\mathrm{DP}})$ is $(T, N, n, L, k, \epsilon)$-secure if it satisfies D-Privacy, and any adversary against the above privacy and integrity notions using time at most T and seeing at most n ballots (where relevant), each with k options chosen from among $L+1$ options, has advantage at most ϵ.

4.4 Instantiation

We shall now describe our instantiation of the above cryptosystem. It will be based on the group structure described in Section 2.

The set of group elements of G will be both the message space M and the set of pre-codes C. We interpret O as the set of options, and 1 as the null option and null code.

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

When option order is irrelevant, we define the map ω as

$$\omega(\vec{m}) = \phi(m_1 m_2 \cdots m_k).$$

Otherwise, ω is simply the identity map.

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

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

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

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

- The *encryption algorithm* $\mathcal{E}(ek, V, \vec{v})$ samples a random number t uniformly at random from $\{0, 1, 2, \ldots, q-1\}$, computes $x = g^t$ and $w_i = y_{1,i}^t v_i$ for $i = 1, 2, \ldots, k$, and generates a proof of knowledge $\pi_{\mathrm{pok}} \leftarrow \mathcal{P}_{pok}(V||x||w_1|| \ldots ||w_k; g; x; t)$. The ciphertext is $c = (x, w_1, w_2, \ldots, w_k, \pi_{\mathrm{pok}})$.

- The *extraction algorithm* $\mathcal{X}(V, c)$ where $c = (x_1, w_1, \ldots, w_k, \pi_{\text{pok}})$, first verifies the proof π_{pok}, computes

$$\tilde{w} = \begin{cases} w_1 w_2 \cdots w_k & \text{if order is irrelevant,} \\ \prod_{i=1}^{k} w_i^i & \text{otherwise,} \end{cases}$$

  and outputs the naked ciphertext $\tilde{c} = (x, \tilde{w})$.

- The *transformation algorithm* $\mathcal{T}(dk_2, V, s, c)$, $c = (x, w_1, \ldots, w_k, \pi_{\text{pok}})$, verifies the proof π_{pok}, computes $\bar{x} = x^s$, $\bar{w}_i = w_i^s$ and $\hat{w}_i = \bar{x}^{a_{2,i}}$ for $i = 1, 2, \ldots, k$, generates proofs

$$\bar{\pi} \leftarrow \mathcal{P}_{eqdl}(c, g, s; x, w_1, \ldots, w_k; \bar{x}, \bar{w}_1, \ldots, \bar{w}_k),$$
$$\hat{\pi} \leftarrow \mathcal{P}'_{eqdl}(c, g, \bar{x}; a_{2,1}, \ldots, a_{2,k}; \hat{w}_1, \ldots, \hat{w}_k)$$

  and outputs the pre-code ciphertext $\check{c} = (\bar{x}, \bar{w}_1, \ldots, \bar{w}_k, \hat{w}_1, \ldots, \hat{w}_k, \bar{\pi}, \hat{\pi})$.

- The *pre-code decryption algorithm* $\mathcal{D}_R(dk_3, V, \gamma, c, \check{c})$, with $c = (x, w_1, \ldots, w_k, \pi_{\text{pok}})$ and $\check{c} = (\bar{x}, \bar{w}_1, \ldots, \bar{w}_k, \hat{w}_1, \ldots, \hat{w}_k, \bar{\pi}, \hat{\pi})$, verifies π_{pok}, $\bar{\pi}$ and $\hat{\pi}$, computes $\rho_i = \bar{w}_i \hat{w}_i \bar{x}^{-a_{3,i}}$ for $i = 1, 2, \ldots, k$ and outputs the precodes $\vec{\rho} = (\rho_1, \ldots, \rho_k)$.

- Since this paper's focus is on the ballot submission protocol, we do not specify a *decryption protocol*.

  However, we note that there is a straight-forward one-move protocol. The prover first creates a shuffle of the naked ciphertexts along with a proof of correct shuffle [15, 23], then decrypts the shuffled naked ciphertexts and creates a proof of correct decryption (possibly using the proof in Section 3.2). It then applies ϕ to every message and outputs the resulting message list. The verifier role verifies the two proofs. If they verify, it outputs the message list, otherwise it outputs $\perp$.

The *decryption algorithm* first verifies the proof of knowledge π_{pok}, then computes $m_i = w_i x^{-a_{1,i}}$, $i = 1, 2, \ldots, k$. Note that, for this to work, the decryption key must be $(a_{1,1}, \ldots, a_{1,k})$, not $\sum_i a_{1,i} \bmod q$.

The decryption algorithm clearly satisfies completeness requirement C1.

The completeness requirement C2 is satisfied, because the zero knowledge proofs are complete, ElGamal encryptions are homomorphic and the map ϕ recovers a proper representation of the ballot from the product.

The completeness requirement C3 is again satisfied because the zero knowledge proofs are complete and ElGamal is homomorphic. For an encryption $(x, w_i) = (g^t, y_{1,i}^t m)$ and a pre-code map s, we get that $\bar{x} = g^{ts}$ and

$$\rho_i = \bar{w}_i \hat{w}_i \bar{x}^{-a_{3,i}} = w_i^s x^{sa_{2,i}} x^{-sa_{3,i}} = g^{ta_{1,i}s} m^s g^{tsa_{2,i}} g^{-tsa_{3,i}} = m^s$$

because $a_{1,i} + a_{2,i} \equiv a_{3,i} \pmod{q}$.

We summarize the security claims about the cryptosystem in the following statement.

Theorem 1. *Suppose there are $L+1$ options, each ballot contains at most k options, there are at most n ballots, and that proof challenges are τ bits long. Suppose further that the Schnorr proof assumption holds, that the SGSP problem (and therefore Decision Diffie-Hellman) is $(\chi T, \epsilon)$-hard, and that $\chi T < 2^{\tau/2} - 1$. Then the cryptosystem is $(T, N, n, L, k, \epsilon')$-secure, where*

$$\epsilon' \leq (k^2 + L)\epsilon + \epsilon_{pok} + 2^{-\tau/2+2} + \frac{3nT}{q} + \frac{N}{q^L}.$$

4.4.1 Security: D-Privacy

Since the voter's identity is only used to create the proof of knowledge π_{pok}, and this part is removed before creating the naked ciphertext, the naked ciphertext is independent of the identity.

4.4.2 Security: B-Integrity

Since the pre-code decryption algorithm verifies the π_{pok} part of c, it cannot be the case that c does not decrypt while $\check{c}$ decrypts.

We may assume that the number of random oracle queries an algorithm makes is bounded by the time T the algorithm executes. Therefore, if T is less than $2^{\tau/2} - 1$, Propositions 1 and 2 tells us that if $\bar{\pi}$ and $\hat{\pi}$ are both accepted, then $\rho_i = m_i^s$ except with probability at most $2^{-\tau/2+2}$.

4.4.3 Security: B-Privacy

We do this proof in the usual manner, and construct a sequence of indistinguishable games resulting in a proof of the following result:

Proposition 3. *If the Schnorr proof assumption holds, Decision Diffie-Hellman is $(\chi T, \epsilon_{DDH})$-hard and $\chi T < 2^{\tau/2} - 1$, then any adversary against B-privacy using time at most T has advantage at most $k^2\epsilon_{DDH} + \epsilon_{pok} + 2^{-\tau/2+2}$, where k is the maximum number of options to encrypt.*

Game 1 corresponds to the game defining B-privacy, and the event E_1 is used to measure the adversary's advantage. Our object is to bound the distance between $\Pr[E_1]$ and $1/2$.

Games 2–5 begin the process by changing the original game such that secret keys are no longer used. This means that we can change the key generation process in Game 6 such that the secret keys no longer exist. Also, we change the encryption process so that it no longer samples a random exponent and computes powers of the encryption keys, but instead samples random DDH tuples by computing linear combinations of a basis of DDH tuples.

In the final game, we change our basis of DDH tuples to a basis of random tuples, which means that we encrypt using random tuples, not DDH tuples. At this point no information about b leaks to the adversary.

An upper-bound on the adversary's advantage follows from this sequence of games, from which the claim follows.

Game 1 We begin with the usual game between a simulator and an adversary, requiring time at most T, receiving at most n challenge ciphertext and making at most N_d queries to the decryption oracle.

If E_1 is the event that the adversary correctly guesses the bit b, the adversary's advantage is

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

Game 2 The original game can be considered as an environment interacting with the proof-of-knowledge-generating protocol given in Figure 2, which by assumption (see Section 3.1) can be replaced with a "morally equivalent" environment. In this case, "morally equivalent" means that the simulator is unchanged and that the modified adversary works as before, except that it also reveals certain discrete logarithms, namely those elements for which the simulator verifies a proof of knowledge.

Furthermore, if the original system required time T, our security assumption states that the modified system requires time χT and any environment has distinguishing advantage at most ϵ_{pok}. Specifically, this means that

$$|\Pr[E_1] - \Pr[E_2]| \leq \epsilon_{pok}.$$

Game 3 In this game, for every honestly generated ciphertext, we remember the ballot encrypted. It is clear that

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

Game 4 In this game, whenever the simulator decrypts a pre-code ciphertext tied to an adversarially generated ciphertext $(x, w_1, \ldots, w_k, \pi_{\text{pok}})$ to pre-codes $\rho_1, \ldots, \rho_k$, it uses the logarithm t of x (revealed by the adversary upon verification of π_{pok}) to decrypt the ciphertext as

$$m_i = w_i y_{1,i}^{-t},$$

and then computes $\rho'_i = m_i^s$.

For any pre-code ciphertext that is tied to an honestly generated ciphertext $(x, w_1, \ldots, w_k, \pi_{\text{pok}})$, the simulator recalls the encrypted ballot $(m_1, \ldots, m_k)$ and computes $\rho'_i = m_i^s$.

The simulator returns the values $\rho'_1, \ldots, \rho'_k$ instead of $\rho_1, \ldots, \rho_k$.

Unless either $\bar{\pi}$ or $\hat{\pi}$ are forgeries, this change is unobservable.It follows by Propositions 1 and 2 that when the time bound χT is less than $2^{\tau/2} - 1$, the probability of forgeriesis bounded by $2^{-\tau/2+2}$. It follows that this game is $2^{-\tau/2+2}$-indistinguishable from the previous game. Specifically, this means that

$$|\Pr[E_3] - \Pr[E_4]| \leq 2^{-\tau/2+2}.$$

Game 5 In this game, we stop computing with the secret keys $a_1, a_{3,1}, \ldots, a_{3,k}$. Since no observable behaviour depended on these computations, this change cannot be observed and

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

Game 6 During key generation, we sample the elements $y_{1,1}, \ldots, y_{1,k}$ at random. The keys $a_{2,1}, \ldots, a_{2,k}$ are generated as usual, but we never generate the keys $a_{1,1}, \ldots, a_{1,k}$ and $a_{3,1}, \ldots, a_{3,k}$.

After key generation, we sample k tuples $(z_{10}, z_{11}, \ldots, z_{1k}), \ldots, (z_{k0}, z_{k1}, \ldots, z_{kk})$ uniformly from the subgroup generated by $(g, y_{1,1}, \ldots, y_{1,k})$.

When we encrypt a message $\vec{v}$, we choose $k+1$ random numbers $t_0, t_1, \ldots, t_k$ and compute the ciphertext as

$$x = g^{t_0} \prod_{j=1}^{k} z_{j0}^{t_j}, \quad \text{and} \quad w_i = y_{1,i}^{t_0} v_i \prod_{j=1}^{k} z_{ji}^{t_j}, \ i = 1, 2, \ldots, k.$$

It is clear that the changes in key generation and encryption are unobservable. Therefore,

$$\Pr[E_5] = \Pr[E_6].$$

Game 7 We sample the k tuples $(z_{10}, z_{11}, \ldots, z_{1k}), \ldots, (z_{k0}, z_{k1}, \ldots, z_{kk})$ uniformly from G^{k+1} after key generation.

First, we see that the only difference between the two games is the distribution of the z_{ji} values. The discussion in Sections 2.3 and 2.5 applies, which means that it is easy to create a Decision Diffie-Hellman distinguisher from an environment that can distinguish these games, using standard results. The following lemma can then be proven.

Lemma 5. *Suppose Decision Diffie-Hellman is $(\chi T, \epsilon_{DDH})$-hard. Then*

$$|\Pr[E_7] - \Pr[E_6]| \leq k^2 \epsilon_{DDH}.$$

Analysis It is clear that in Game 7, it is impossible for the adversary to decide if the requested messages were encrypted or not. In other words,

$$\Pr[E_7] = 1/2.$$

A simple application of the triangle inequality tells us that

$$\epsilon = |\Pr[E_1] - 1/2| \leq \epsilon_{pok} + 2^{-\tau/2+2} + k^2 \epsilon_{DDH},$$

which concludes the proof of Proposition 3.

4.4.4 Security: R-Privacy

We shall now prove a bound for the advantage of an adversary against R-privacy.

Proposition 4. *If the SGSP problem is (T, ϵ_{SGSP})-hard and the DDH problem is (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*

$$\epsilon \leq \frac{3 n_{tot} T}{q} + L \epsilon_{SGSP} + N q^{-L} + k^2 \epsilon_{DDH}.$$

We do this proof in the usual manner, and construct a sequence of indistinguishable games resulting in a proof of the claim. Game 1 corresponds to the game defining R-privacy, and the event E_1 is used to measure the adversary's advantage. Our goal is to bound the distance between $\Pr[E_1]$ and $1/2$.

Game 2 replaces real proofs of knowledge and proofs of correct computation by simulated versions. Game 3–4 isolates the use of the per-identity pre-code generation map to the start of the game.

In Game 5 we use random pre-codes instead of computing pre-codes, and the distinguishing advantage is bounded by the advantage against the SGSP problem.

Games 6–8 replaces the encryptions of the ballots by encryptions of random group elements. The distinguishing advantage is essentially bounded by the advantage against the DDH problem. In the final game, it is trivial to verify that the adversary has no advantage.

Game 1 We begin with the R-privacy game between a simulator and an adversary. The game requires time at most T and the adversary submits at most n_{tot} ballots.

If E_1 is the event that the adversary correctly guesses the bit b, the adversary's advantage is

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

Game 2 In this game, whenever we need to generate a proof, we fake the proofs using the simulators given in Section 3 and reprogram the random oracle appropriately.

This is only noticeable if the random oracle reprogramming fails, because the oracle has already been queried there before. There are at most $3n_{tot}$ reprogramming attempts, and since the number of random oracle queries is bounded by T, and each query involves a group element chosen uniformly at random, the probability that any single reprogramming attempt fails is bounded by T/q.

We can now bound the probability of the exceptional event, and we see that

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

Note that the encryption process is described by the following equations:

$$\begin{array}{lll} x = g^t & w_i = y_{1,i}^t v_i & \\ \check{x} = x^{s_V} & \bar{w}_i = w_i^{s_V} & \hat{w}_i = \check{x}^{a_{2,i}} \end{array}$$

where $1 \leq i \leq k$.

Game 3 At the start of this game, we construct for each voter V a table of options and corresponding pre-codes $(v, \rho_{V,v})$, where $\rho_{V,v} = v^{s_V}$.

Now we encrypt using the following equations:

$$\begin{array}{lll} x = g^t & w_i = x^{a_{1,i}} v_i & \\ \check{x} = \gamma_V^t & \bar{w}_i = \check{x}^{a_{1,i}} \rho_{V,v_i} & \hat{w}_i = \check{x}^{a_{2,i}} \end{array}$$

Note that we exploit the fact that key generation is done by the simulator. Therefore, we know all the secret keys.

This change to the encryption process cannot be noticed, since

$$\gamma_V^t = (g^{s_V})^t = (g^t)^{s_V} = x^{s_V}.$$

Therefore,

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

Game 4 We change the encryption process again. The simulator chooses additional random numbers $t'_1, \ldots, t'_k$ and does:

$$\begin{array}{lll} x = g^t \prod v_i^{t'_i} & w_i = x^{a_{1,i}} v_i & \\ \check{x} = \gamma_V^t \prod \rho_{V,v_i}^{t'_i} & \bar{w}_i = \check{x}^{a_{1,i}} \rho_{V,v_i} & \hat{w}_i = \check{x}^{a_{2,i}} \end{array}$$

Since the group is cyclic of prime order, for every i there exists an integer u such that $v_i = g^u$ and $\rho_{V,v_i} = v_i^{s_V} = (g^{s_V})^u = \gamma_V^u$. The joint distribution of x and $\check{x}$ is therefore identical to the one in the previous game, and

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

Game 5 At the start of the game, when we construct the per-voter table of options and corresponding pre-codes, the simulator chooses for each option a random pre-code $\rho_{V,v}$ instead of computing $\rho_{V,v} = v^{s_V}$.

We can now construct a distinguisher for the L-SGSP problem. The distinguisher gets L challenge tuples and constructs the per-voter tables as random linear combinations of these tuples. It then runs the remaining (common) game.

If the distinguisher gets elements that all lie inside the subgroup, the random linear combinations will all lie inside the subgroup, and the tables will be generated according to the exact same distribution as in Game 4. Otherwise, the random linear combinations will result in random tuples, and the tables will be generated according to the exact same distribution as in Game 5. The distinguisher therefore proves the following lemma:

Lemma 6. *If the SGSP problem is (T, ϵ_{SGSP})-hard, then*

$$|\Pr[E_5] - \Pr[E_4]| \leq L\epsilon_{SGSP}.$$

Game 6 Once more, we change the encryption process as follows (t' is another random number):

$$\begin{array}{lll} x = g^t & w_i = x^{a_{1,i}} v_i & \\ \check{x} = \gamma_V^{t'} & \bar{w}_i = \check{x}^{a_{1,i}} \rho_{V,v} & \hat{w}_i = \check{x}^{a_{2,i}} \end{array}$$

Unless the randomly chosen tuple $(\gamma_V, \rho_{V,v_1}, \ldots, \rho_{V,v_L})$ lies inside the subgroup, the distribution of $\check{x}$ in Game 5 will be independent of the distribution of x, just like in this game.

The probability that one such tuple lies inside the subgroup is q^{-L}. The probability that at least one out of N such tuples lies inside the subgroup is upperbounded by Nq^{-L}.

We now have a bound on an exceptional event, from which it follows that

$$|\Pr[E_6] - \Pr[E_5]| \leq Nq^{-L}.$$

We note that this term is extremely small, but we include it for completeness sake.

Game 7 We change the encryption process as follows:

$$\begin{array}{lll} x = g^t & w_i = y_{1,i}^t v_i & \\ \check{x} = g^{t'} & \bar{w}_i = y_{1,i}^{t'} \rho_{V,v_i} & \hat{w}_i = y_{2,i}^{t'} \end{array}$$

It is clear that the resulting distributions are identical to the previous game's distribution, and therefore that

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

At this point, the pre-code ciphertext is an independent encryption of a random per-option value.

Game 8 In the final game, we again change the encryption process as follows (u_i is a random group element):

$$\begin{array}{lll} x = g^t & w_i = y_{1,i}^t u_i v_i & \\ \check{x} = g^{t'} & \bar{w}_i = y_{1,i}^{t'} \rho_{V,v_i} & \hat{w}_i = y_{2,i}^{t'} \end{array}$$

As for Game 4 and Game 5, we can now create a k-DDH distinguisher. This distinguisher (which we omit) proves the following lemma:

Lemma 7. *If the DDH problem is (T, ϵ_{DDH})-hard, then*

$$|\Pr[E_8] - \Pr[E_7]| \leq k^2 \epsilon_{DDH}.$$

Analysis In the final game, since the ciphertexts are independent of the exact options used to create them, the adversary cannot tell if we encrypt v_i or a permutation of v_i. Therefore,

$$\Pr[E_8] = 1/2.$$

A simple application of the triangle inequality tells us that

$$\epsilon = |\Pr[E_1] - 1/2| \leq \frac{3n_{tot}T}{q} + L\epsilon_{SGSP} + Nq^{-L} + k^2\epsilon_{DDH},$$

which concludes the proof of Proposition 4.

4.4.5 Security: D-Integrity and A-Privacy

Since this paper's focus is on ballot submission, the shuffle and decryption proofs have not been specified. It is therefore impossible to properly analyse the security.

However, the standard security notions for verifiable decryption and verifiable shuffles are sufficient to ensure verifiable shuffled decryption, which is what our security requirements amount to.

Such proofs exist and are reasonably efficient.

5 The Voting Protocol

The voting protocol is built on top of the cryptosystem from Section 4, together with several other tools.

The players in the voting protocol are the voters, the voters' computers, the voters' telephones, several electoral board players and several infrastructure players.

5.1 Functional and Security Requirements

The ideal functionality described in Figure 4 defines both the functional requirements for our internet voting protocol, as well as the security requirements.

Conceptually, there are three phases to the election: setup, ballot submission, and counting. We assume that every player knows when the transition from the setup phase to the ballot submission phase happens. Therefore, we are justified in not describing the housekeeping that deals with this transition. However, there may be players that do not know the exact time of the transition from ballot submission to counting. Therefore, we describe a bit of housekeeping for that transition.

The setup phase is used by the electoral board and the infrastructure players to generate keys. The adversary is free to block key generation, but in that case, the functionality never enters the ballot submission phase.

During ballot submission, voters submit ballots. The adversary may to a certain degree interfere with ballot submission, or attempt to forge ballots.

Finally, the functionality enters counting phase when the ballot box is instructed to close. The adversary may to a certain degree interfere with counting.

The adversary's ability to interfere undetectably with ballots is determined by the relation $\sim$ on the ballot set O^k.

The amount of information that leaks directly to an adversary during ballot submission is described by the $\mathit{leak}(\cdot,\cdot,\cdot,\cdot)$ function defined as follows:

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

Note that Λ is a permutation on O which is extended to O^k in the obvious manner.

We see that a corrupt computer will learn the voter's confidential ballot, and a corrupt return code generator will learn a permutation of the voter's chosen options.

The following claims about the security of the ideal functionality are all easy to verify:

S1 If the auditor accepts the result, then:

- At most one ballot per voter will be counted.
- The number of ballots counted will not be higher than the number of voters who submitted a ballot, attempted to submit a ballot or complained about a forgery.

S2 If the voter accepts his ballot as cast as intended, and does not later revote or complain about a forgery, the ballot is counted as intended up to equivalence under $\sim$.

S3 If the voter's computer and the return code generator are both honest, and the voter does not complain about forgeries, the content of the voter's ballot remains private.

S4 If the return code generator is corrupt, the adversary learns the number of null options in each ballot submission, and if a voter submits multiple ballots, learns where these ballots differ.

When we prove that the protocol realizes the ideal functionality, it then follows that the above claims also hold for the protocol. We shall see that these claims (and the leakage function) are optimal for the protocol.

The leakage caused by corrupt computers seems unavoidable in practice. To protect against a corrupt computer, the voter must somehow input an encrypted ballot, which is too difficult for the complex Norwegian ballots. For other elections, this could be practically feasible.

Likewise, the leakage caused by a corrupt return code generator is again hard to avoid, since the voter must be able to interpret the return codes. Multiple

Setup phase

On (keygen) from U:

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

Ballot submission phase

On (use, P) from V:

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

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

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

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

On $(\mathsf{replay}, \mathit{seq})$ from $\mathcal{A}$:

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

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

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

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

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

Figure 4a: Ideal functionality for the internet voting protocol, parameterized by a $\mathit{leak}(\cdot, \cdot, \cdot, \cdot)$ function.

Ballot submission phase (cont.)

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

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

Counting phase

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

1. Select the next sequence number *seq*.
2. Stop if V is marked as voting. Mark V as voting.
3. Hand over $(\mathsf{voting}, \mathit{seq}, V, P, \mathit{leak}(\Lambda_V, V, P, \vec{m}))$ to $\mathcal{A}$. Wait for (store, V, b_1), $(\mathsf{notify}, V, b_2)$ and $(\mathsf{finish}, V, b_3)$ from $\mathcal{A}$. Between messages, hand over (done) to $\mathcal{A}$.
4. Send (reject) to V.
5. Mark V as not voting.

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

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

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

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

Figure 4b: Ideal functionality for internet voting, parameterized by a $\mathit{leak}(\cdot, \cdot, \cdot, \cdot)$ function.

On (send, X, m) from Y:

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

Figure 5: Secure channel functionality.

sets of return codes would provide some measure of security, but the added complexity is not commensurate with the added security.

While the leakage seems acceptable (comparable to surveillance attacks against ordinary paper elections), active attacks are much more worrying. If the voter has ever used a corrupt computer, the functionality allows an adversary to submit forged ballots on the voter's behalf. The forged ballot is unproblematic, since the voter will report the forgery eventually and take corrective action. But the adversary may use this ability to learn information about confidential ballots.

The following attack works. The attacker notices that the voter attempts to submit a ballot, guesses what ballot the voter is about to submit, and immediately submits the guess as a forgery. Careful interaction with the functionality ensures that the voter accepts the ballot if and only if the attacker's guess matches the voter's ballot. The subsequent behaviour of the voter leaks information about the ballot. Eventually, the voter will complain about a forgery, but this does not prevent a loss of confidentiality.

5.2 Assumptions about the Environment

The players in the protocol can be distinguished into three groups: the voters and their computers and phones, the infrastructure players and the electoral board players.

There is a set of voters $\mathcal{V}$, a set of computers $\mathcal{P}$ and a set of phones $\mathcal{F}$. Several voters may share one computer, and one voter may use multiple computers. Phones, however, are personal and we denote the phone belonging to voter V by F_V.

We assume that there are confidential, identified and authenticated channels between the infrastructure players, between voters and computers, between voters and their own phones, and a one-way channel from the return code generator R to the voters' phones. This assumption is captured by the ideal functionalities in Figure 5 and 6.

For the channel from the return code generator to the voter's phone, messages are confidential, and the adversary cannot interfere with their integrity. They are also delivered in the same order as they were sent. The only thing under adversarial control is the timing of the delivery. For the channels modeled by the functionality in Figure 5, the adversary cannot even interfere with the timing of the delivery.

Remark. In the interests of readability, we ignore the secure channel functional-

On (send, F_V, m) from R:

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

On $(\mathsf{deliver}, F_V)$ from $\mathcal{A}$:

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

Figure 6: Phone channel functionality

ity in the description of the voting protocol. When U wants to send a message to U', U will a message to U', not send (send, U', m) to the secure channel functionality. Likewise, U will receive a message from U', not a (recv, U', m) message from the secure channel functionality.

We shall also assume the existence of a pre-existing PKI where voters can delegate their identity to one or more computers, after which the computer can on behalf of the voter digitally sign and establish authenticated connections to the ballot box. We shall assume that every infrastructure player can verify such signatures. This assumption is captured by the ideal functionality described in Figure 7.

Remark. In the interests of readability, we ignore most of the technicalities involving the PKI functionality in the description of the voting protocol. We shall say that P establishes a connection to B, and then sends messages to and receives messages from B, ignoring the trivial technicalities involving session identifiers and sending messages to the functionality. Likewise, we shall say that P signs a message on behalf of V, instead of sending a signing message and waiting for the signature reply.

Finally, we need a hash function H for various purposes. We shall require that this function is collision resistant.

The return code generator will have a signing key, and the ballot box and every computer has the corresponding verification key. The assumption that this cryptographic infrastructure is pre-existing is captured by the ideal functionality described in Figure 8.

Remark. As usual, in the interests of readability, we say that R signs something to get a signature, instead of sending a message to the functionality and then waiting for the reply. Likewise, we say that the other players verify R's signature on some message.

5.3 Setup Phase

Our focus in this paper is not on the setup phase. We shall quite simply assume that all keys are generated by a trusted dealer (except for the return code generator's signing key, where only key distribution is needed), as described in Figure 9.

We note that election key generation can happen a long time before the election, so key generation does not have to be very quick. The key generation

On (use, P) from V:

1. Hand (using, V, P), and wait for (using, V, P). Record (use, V, P). Send (using) to V.

On $(\mathsf{establish}, B, V)$ from P:

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

On (send, sid, m) from U:

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

On $(\mathsf{deliver}, sid, b)$ from $\mathcal{A}$:

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

On (sign, V, m) from P:

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

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

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

Figure 7: PKI functionality, parameterized over a signature scheme $(\mathcal{K}_s, \mathcal{S}_s, \mathcal{V}_s)$.

On (sign, m) from R:

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

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

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

Figure 8: Return code generator signature functionality, parameterized over a signature scheme $(\mathcal{K}_s, \mathcal{S}_s, \mathcal{V}_s)$.

On (keygen) from U:

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

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

1. If $b = 1$, send $(\mathsf{keys}, \mathit{material})$ to U, where *material* is as detailed in Table 1. Otherwise, send (reject) to U.
2. If $(\mathsf{keygen}, U, 1)$ has been received for every infrastructure player and every electoral board player, then for each voter V, send $(\mathsf{keys}, \mathit{material})$ to V, where *material* is as detailed in Table 1.

Figure 9: Trusted dealer for the setup phase of the protocol.

algorithm can therefore easily be replaced by a suitable multiparty protocol.

One possible protocol for generating the per-voter key material is the following three-party protocol. Player 1 samples the precode derivation map s (and its commitment γ) and a random permutation $m_1, \ldots, m_L$ of the elements of O. It then sends to Player 2 the random permutation, and to Player 3 the commitment to s and the sequence $s(m_1), \ldots, s(m_L)$. Player 3 chooses from C_H a random unique value for each element in the sequence, and sends this list to Player 2.

Player 1 now knows s, Player 3 knows γ and a random injective function D from the image of s to C_H, and Player 2 can reconstruct the set $\{(m, r) \mid r = D(s(m))\}$.

Table 1: Distribution of key material to the players.

Player	Key material
V	The set $\{(m, D_V(s_V(m))) \mid m \in O \setminus \{1_O\}\}$.
P	The public key ek.
B	The transformation key dk_2 and the set $\{(V, s_V)\}$.
R	The pre-code decryption key dk_3 and the set $\{(V, \gamma_V, D_V)\}$.
D	The decryption key dk_1.
A	The public key ek.

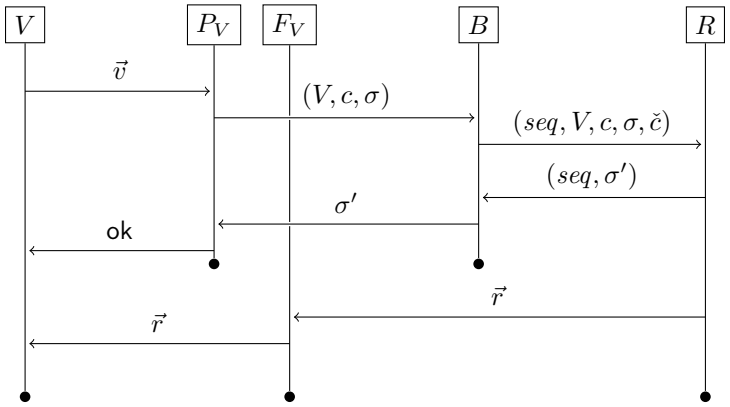

Figure 10: Overview of messages sent during ballot submission.

5.4 Protocol

The voting protocol is described in Figure 11 (pp. 40–44.)

As discussed above, the protocol execution goes through three phases: setup, ballot submission and counting. During the setup phase, only the electoral board players and the infrastructure players are active.

During ballot submission, only two of the four infrastructure players are active, along with the voters and their computers and phones. The diagram in Figure 10 gives an overview of the ballot submission process.

- The voter gives his ballot to a computer.
- The computer encrypts the ballot and signs it on the voter's behalf, then submits it to the ballot box.
- The ballot box transforms the ciphertext and sends everything to the return code generator.
- The return code generator creates the return codes and sends them to the voter's phone. It also signs a hash of the ballot and sends this signature to the ballot box.

- The ballot box stores the encrypted ballot and passes the return code generator's signature on to the voter's computer.
- The computer verifies the signature and tells the voter that the ballot was accepted.
- When the voter's phone receives the return codes, it passes them on to the voter.
- The voter accepts the ballot as cast only if the computer accepted the ballot as cast, and the return codes are correct.

Ballot submission ends when the ballot box is told to close. During the counting phase, only the infrastructure players are active.

The ballot box refuses to accept new ballot submissions, but waits until ongoing ballot submissions are done. Then it informs the return code generator that the ballot box has closed, sends every recorded ballot to the auditor, extracts naked ciphertexts from the ballots that should be counted, and sends the naked ciphertexts to the decryptor.

The return code generator sends its records to the auditor.

The decryptor tells the auditor which naked ciphertexts it receives, then runs the decryption protocol with the auditor as verifier.

The auditor verifies that the ballot box and the return code generator agree on which ballots were submitted, and that it agrees with the decryptor about which naked ciphertexts should be counted. Then it runs the decryption protocol with the decryptor as prover.

5.5 Security Analysis

It is well-known that no ideal functionality for public key encryption can be realized under adaptive corruption. Since our cryptosystem is essentially public key encryption, we must restrict ourselves to static corruption.

Our protocol can only guarantee security if at most one infrastructure player is corrupt. Likewise, if the return code generator and some voter's computer are both corrupt, security cannot be guaranteed. Furthermore, assuming that a corrupt voter will have to use an honest computer is unrealistic. We therefore arrive at the following four classes of static corruption:

- the ballot box, a subset of the voters and a subset of the computers are corrupt;
- the return code generator is corrupt;
- the decryptor is corrupt; and
- the auditor is corrupt.

Our goal is to prove the following theorem.

Setup phase

On (keys, T) from key generation functionality:

1. Remember the set T.

Ballot submission phase

On input (use, P):

1. Send (use, P) to the PKI functionality, wait for (using) in return, and record (use, V, P).

On input $(\mathsf{vote}, P, \vec{m})$:

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

On $(\mathsf{codes}, \ldots)$ from F_V:

1. Output $(\mathsf{forgery})$.

Figure 11a: The voting protocol: The voter.

On m from R:

1. Send m to V.

Figure 11b: The voting protocol: The phone.

Setup phase

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

1. Remember ek.

Ballot submission phase

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

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

Figure 11c: The voting protocol: The computer.

Setup phase

On input (keygen):

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

Ballot submission phase

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

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

On input (close):

1. Enter counting phase. Wait until every ballot submission currently in progress has finished.
2. Send (close) to R.
3. Let L_1 be the list of all ballot records. For each voter, select the ballot record with maximal sequence number, resulting in the records $(\text{ballot}, seq_1, V_1, c_1, \sigma_1), \ldots, (\text{ballot}, seq_n, V_n, c_n, \sigma_n)$.
4. Compute $\tilde{c}_i = \mathcal{X}(V_i, c_i)$, $i = 1, 2, \ldots, n$.
5. Let L_2 be the sorted list of naked ciphertexts $\tilde{c}_1, \ldots, \tilde{c}_n$.
6. Send (decrypt, L_2) to D and (content, L_1) to A.

Counting phase

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

1. Ignore.

Figure 11d: The voting protocol: The ballot box

Setup phase

On input (keygen):

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

Ballot submission phase

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

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

On (close) from B:

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

Counting phase

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

1. Ignore.

Figure 11e: The voting protocol: The return code generator.

Setup phase

On input (keygen):

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

Counting phase

On (decrypt, L_2) from B:

1. Send (hash, $H(L_2)$) to A. Wait for (accept) from A.
2. Run the prover role of the decryption protocol Π_{DP} with dk_1 as private input and L_2 as public input, and with A playing the verifier role.
3. If the decryption protocol output is $\perp$, output (reject).
4. If the decryption protocol output is $\vec{m}_1, \ldots, \vec{m}_n$, then output (result, $sort(\vec{m}_1, \ldots, \vec{m}_n)$).

Figure 11f: The voting protocol: The decryptor.

Theorem 2. *Consider an election with N voters, L options and ballot length k. Suppose the cryptosystem is $(T, N, n_{tot}, L, k, \epsilon)$-secure and the hash function H is (T, ϵ)-collision resistant. Let R_2 be a resource bound for interaction with the internet voting protocol that limits the total number of messages to ν, the total number of submitted ballots to n_{tot} and the time used to T.*

There exists a resource bound R_1 that is the same as R_2, though with a slightly smaller time bound, such that under static corruption restricted as described above, the internet voting protocol given in Figure 11 interacting with the ideal functionalities given in Figures 5–9 $(R_1, \delta, R_2, \delta + 3\epsilon, \delta + 3\epsilon)$-realizes the internet voting functionality given in Figure 4.

This theorem follows from Propositions 5, 6, 7 and 8.

We must specify an ideal simulator and prove that the composition of the functionality and the simulator is indistinguishable from the real protocol interacting with its functionalities. Under static corruption, we can consider the each case separately.

In all cases, our strategy is the same. We begin with a game where the environment interacts with the protocol and its functionalities. Then we make several modifications to this game. Finally, in the end, we show how an ideal simulator can be constructed based on the final game.

5.5.1 The Ballot Box

Proposition 5. *Under the conditions of Theorem 2, there exists a resource bound R_1 such that when the ballot box, a subset of the voters and a subset of the computers are corrupt, the internet voting protocol given in Figure 11 interacting*

Setup phase

On input (keygen):

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

Counting phase

On (content, L_1) from B:

1. Wait for (hashes, L_3) from R and (hash, h) from D.
2. Verify that for each ballot (ballot, seq, V, c, σ) in L_1, σ is V's signature on c.
3. Verify that for each ballot (ballot, seq, V, c, σ) in L_1, there is a corresponding record (ballot, V, seq, h') in L_3 with $h' = H(V, c)$, and *vice versa*.
4. For each voter, select the ballot record with maximal sequence number from L_1, resulting in the records (ballot, $seq_1, V_1, c_1, \sigma_1$), ..., (ballot, $seq_n, V_n, c_n, \sigma_n$).
5. Compute $\tilde{c}_i = \mathcal{X}(V_i, c_i)$, $i = 1, 2, \ldots, n$.
6. Verify that h equals $H(sort(\tilde{c}_1, \ldots, \tilde{c}_n))$.
7. Send (accept) to D.
8. Run the verifier role of the decryption protocol Π_{DP} with ek and L_2 as public input, and with D playing the prover role.
9. If the decryption protocol output is $\perp$, output (reject).
10. If the decryption protocol output is $\vec{m}_1, \ldots, \vec{m}_n$, then output (result, $sort(\vec{m}_1, \ldots, \vec{m}_n)$).

Figure 11g: The voting protocol: The auditor.

with the ideal functionalities given in Figures 5–9 $(R_1, \delta, R_2, \delta+3\epsilon, \delta+3\epsilon)$-realizes the internet voting functionality given in Figure 4.

Let T_i be the time bound for R_i.

The proof proceeds as follows: We begin with a game where an environment interacts with the voting protocol and its underlying functionalities. Then we make a sequence of modifications to this game, where each modification gives a new game that is indistinguishable from the preceeding game.

Game 1 is the initial game. Game 2 is a technical step where we gather all the various honest machines into one big machine, and make various technical preparations for subsequent games.

In Game 3, our simulated return code generator computes the pre-codes from the ballot decryptions using the per-voter pre-code map, instead of from the transformed ciphertexts. This does not change observable behaviour by B-integrity.

Game 4 introduces bookkeeping to keep track of the decryption of the ciphertexts generated or seen by the honest players, in preparation for a later game. Game 5 compares two lists of ciphertexts instead of comparing the hash of the lists. Since the hash function is collision resistant, this is not observable.

Game 6 decrypts the ciphertexts directly (using $\mathcal{D}$) instead of running the decryption protocol. Game 7 no longer decrypts values, instead relying on the bookkeeping introduced in Game 4. These changes are unobservable due to completeness.

In Game 8, honestly generated ballots are encryptions of random ballots instead of the real ballots. Since the decryptions of the honestly generated ciphertexts are never used, this is unobservable by B-privacy.

Game 9 removes the reliance on the actual return codes. By the properties of the various functions used, this does not change observable behaviour.

When we have suitably modified the initial game, we construct an ideal simulator that runs a copy of the final game. The input to the protocol machines is simulated, based on messages from the ideal functionality. Certain events in the game are translated into messages to the ideal functionality.

It will be straight-forward to verify that our machine in the final game is indistinguishable from the ideal functionality interacting with the ideal simulator and the environment. Transitivity of indistinguishability then proves that the voting protocol realizes the functionality.

Game 1 The initial game consists of some environment interacting with the protocol and its underlying functionalities.

We assume that this game is (R_1, δ)-bounded, where R_1 is a resource bound with time bound T_1 and at most ν messages sent by the environment. Note that ν is very small compared to T_1.

Game 2 We gather all the honest players and the functionalities into one machine.

We also insert certain delays into much of the processing in this game. The delays allow us to later modify processing without changing time usage. This simplifies the technicalities involved.

Given the time bound T_1 for Game 1, this game will have a time bound $T_2 = T_1 + poly(\nu)$. As will be seen through the rest of the proof, the constants involved are very small, so $T_2 \approx T_1$.

With R_2 the same as R_1 except that the time bound is replaced by T_2, we have that Game 2 is (R_2, δ)-bound. The following result is immediate.

Lemma 8. *Game 2 is $(R_1, \delta, R_2, \delta, \delta)$-indistinguishable from Game 1.*

Game 3 Step 5 in the return code generator is modified as follows:

5. Compute $\vec{\rho}_0 \leftarrow \mathcal{D}_R(dk_3, V, \gamma_V, c, \check{c})$ and $\vec{m} \leftarrow \mathcal{D}(dk_1, V, c)$, verify that $\vec{m} \neq \perp \neq \vec{\rho}_0$ and that $m_i \in O$ for $i = 1, 2, \ldots, k$. Compute $\vec{\rho} = (s_V(m_1), \ldots, s_V(m_k))$.

Lemma 9. *If the cryptosystem is $(T_2, N, n_{tot}, L, k, \epsilon)$-secure, with $n_{tot} < \nu$, then Game 3 is $(R_2, \delta, R_2, \delta + \epsilon, \epsilon)$-indistinguishable from Game 2.*

Proof. It is clear that the two games behave differently only when the system as a whole has constructed a ciphertext c and transformed ciphertext $\check{c}$ that decrypt inconsistently, either because c does not decrypt correctly, but $\check{c}$ does, or the decryption of $\check{c}$ is inconsistent with the decryption of c.

This is exactly the requirement for winning the B-integrity game, and we see that it is easy to convert the system in Game 3 into an adversary against B-integrity.

We therefore have an exceptional event and a probability bound on that event, and the claim follows. □

Game 4 We add some bookkeeping: When an honest computer generates a ciphertext, we record who generated it and the corresponding message. When Step 5 of the return code generator successfully decrypts a message, we record the message and the corresponding voter identity.

Game 5 Modify the decryptor and the auditor so that instead of verifying that they agree on a hash of the naked ciphertexts, they verify that they agree on the naked ciphertexts. The following result is immediate.

Lemma 10. *If the hash H is (T_2, ϵ)-secure, then Game 5 is $(R_2, \delta + \epsilon, R_2, \delta + 2\epsilon, \epsilon)$-indistinguishable from Game 4.*

Proof. Finding a collision in the hash is an exceptional event. If the hash is secure, we have an appropriate probability bound, and the claim follows. □

Game 6 We modify both the decryptor and the auditor so that instead of running the decryption protocol on naked ciphertexts, they run the decryption algorithm on the corresponding ciphertexts as divulged by the corrupt ballot box, then applies the ω before sorting the result.

The completeness requirements on the cryptosystem ensures that this game behaves exactly as the previous game.

Game 7 Modify Step 5 so that instead of decrypting ciphertexts, we look up the ciphertext record and use that result instead.

Note that every ciphertext that is decrypted is either honestly generated (in which case we have a record), or seen and decrypted by the return code generator (in which case we have a record). Therefore, this game behaves exactly as the previous game.

Game 8 Modify Step 3 of the computer description to encrypt a random message, not the voter's ballot.

3. Choose random $\vec{m}_0$ and compute $c \leftarrow \mathcal{E}(ek, V, \vec{m}_0)$.

Lemma 11. *If the cryptosystem is $(T_2, N, n_{tot}, L, k, \epsilon)$-secure with $n_{tot} < \nu$, then Game 8 is $(R_2, \delta + 2\epsilon, R_2, \delta + 3\epsilon, \epsilon)$-indistinguishable from Game 7.*

Proof. We want to transform the two games into adversaries for the B-privacy game. To do that, we have to create an adversary that when interacting with the privacy game's simulator behaves either as Game 7 or Game 8, depending on the simulator's choice for b.

The decryption of transformed ciphertexts and ciphertexts generated by the adversary can be replaced by queries to the decryption oracle. There is no other use of private keys, so key generation can be simulated using the key material the adversary receives.

Furthermore, the encryptions done by honest computers can be replaced by encryption queries. If the simulator's b is 0, the encryption oracle encrypts as in Game 7, otherwise it encrypts as in Game 8. Any change in behaviour will yield an advantage against B-privacy, thereby upperbounding the distinguishability. □

Game 9 We modify the voter and the return code generator as follows:

- The voter keeps track of the sequence in which ballots are submitted to honest computers. This corresponds to the sequence in which honestly generated ciphertexts are created, which we also keep track of.
- When the return code generator responds to an honestly generated ciphertext, it replies with its sequence number, not with the return codes. When the return code generator responds to a adversary-generated ciphertext for an honest voter, it replies with the ballot itself and not the return code.
- When the honest voter receives a message containing a sequence number, it looks up the corresponding ballot and compares it with the submitted ballot. When the honest voter receives a message containing a ballot, it compares it with the submitted ballot.

The return code generator rejects a ciphertext unless the decryption is in O^k. Since the function $D_V \circ s : O \to C_H$ is injective, comparing ballots before or after applying $D_V \circ s$ makes no difference.

The Ideal Simulator The simulator runs a copy of Game 9. The outputs from the various players are discarded, unless otherwise specified.

On (keygen, U), hand over (keygen) to U. On output (accept) from U, hand over $(\mathsf{keygen}, U, 1)$ to the functionality. On output (reject) from U, hand over $(\mathsf{keygen}, U, 0)$ to the functionality.

On (using, P, V), hand over (use, P) to V. On (using) from V, hand over (using, P, V) to the functionality.

On $(\mathsf{voting}, \mathit{seq}, V, P, \perp)$ or $(\mathsf{voting}, \mathit{seq}, V, P, \vec{m})$, the simulator starts a voting session by giving V as input $(\mathsf{vote}, P, (1_O, \ldots, 1_O))$ or $(\mathsf{vote}, P, \vec{m})$.

When the return code generator accepts a ciphertext c for the voter V, there are four cases:

- If the voter is corrupt, the ideal simulator hands over $(\mathsf{forge}, V, P, \vec{m})$ to the functionality, for some corrupt computer P.
- If the ciphertext was not created by an honest computer, a corrupt computer P, which V has used, must have asked the PKI functionality to sign the ciphertext. The ideal simulator then hands over $(\mathsf{forge}, V, P, \vec{m})$ to the functionality.
- If the ciphertext was created by an honest computer in a session where $(\mathsf{store}, V, \cdot)$ has not yet been sent, hand over $(\mathsf{store}, V, 1)$ to the functionality.
- If the ciphertext was created by an honest computer in a session with sequence number seq where $(\mathsf{store}, V, 0)$ was sent, hand over $(\mathsf{replay}, \mathit{seq})$ to the functionality.

When a voting session for voter V where $(\mathsf{notify}, V, 1)$ has not yet been sent receives a message from the voter's phone, hand over $(\mathsf{notify}, V, 1)$ to the functionality.

If the voter V is about to output $(\mathsf{forgery})$, hand over $(\mathsf{notify}, V, 1)$ to the functionality.

When a voting session for voter V using P receives (reject) from P, the simulator hands over $(\mathsf{finish}, V, 0)$ to the functionality. If they have not already been sent for this session, it also hands over $(\mathsf{store}, V, 0)$ and $(\mathsf{notify}, V, 0)$.

When a voting session for voter V using P receives (accept) from P, the simulator hands over $(\mathsf{finish}, V, 1)$ to the functionality.

When the return code generator receives (close) from the corrupt B, the simulator sends (close) to the functionality. The functionality immediately replies with the final ballots. When the auditor/decryptor decides to output (reject), the simulator hands over $(\mathsf{finish}, A/D, 0)$. When the auditor/decryptor decides to output $(\mathsf{result}, \ldots)$, the simulator hands over $(\mathsf{finish}, A/D, 1)$.

We note the following about Game 9:

- During the ballot submission phase, ballots submitted through honest computers are only used for the return code verification. In the simulated Game 9, this check is done with incorrect ballots. The ideal simulator ignores the outcome of this incorrect check. Instead, the functionality performs the correct check with the correct ballots.

- When we consider the ballots of a single voter, they are received by the return code generator with strictly increasing sequence numbers. Every time the return code generator accepts a ballot, the functionality is instructed to record that ballot as the voter's choice.

  Since the ballots with maximal sequence number are selected for counting, this will correspond to the ballots stored by the functionality.

  Furthermore, ballots are added to the functionality's notification queue in the same order as messages are inserted into the phone channel queue.

It is now fairly easy to verify that the ideal functionality interacting with this ideal simulator has the same behaviour as our machine in Game 9, after which Proposition 5 follows.

5.5.2 The Return Code Generator

Proposition 6. *Under the conditions of Theorem 2, there exists a resource bound R_1 such that when the return code generator is corrupt, then the internet voting protocol given in Figure 11 interacting with the ideal functionalities given in Figures 5–9 $(R_1, \delta, R_2, \delta + \epsilon, \delta + \epsilon)$-realizes the internet voting functionality given in Figure 4.*

Let T_i be the time bound for R_i.

The proof proceeds as follows: We begin with a game where an environment interacts with the voting protocol and its underlying functionalities. Then we make a sequence of modifications to this game, where each modification gives a new game that is indistinguishable from the preceeding game.

Game 1 is the initial game. Game 2 is a technical step where we gather all the various honest machines into one big machine, and make various technical preparations for subsequent games.

Game 3 does not run the decryption protocol, but instead records ballots as ciphertexts are created, and computes the result from these records. Since every ciphertext is honestly generated, completeness ensures that this does not change observable behaviour.

Game 4 chooses randomly permutes ballot options before encryption and before return code verification. This change is unobservable by R-privacy.

When we have suitably modified the initial game, we construct an ideal simulator that runs a copy of the final game. The input to the protocol machines is simulated, based on messages from the ideal functionality. Certain events in the game are translated into messages to the ideal functionality.

It will be straight-forward to verify that our machine in the final game is indistinguishable from the ideal functionality interacting with the ideal simulator

and the environment. Transitivity of indistinguishability then proves that the voting protocol realizes the functionality.

Game 1 We begin with a game where an environment and a corrupt return code generator interacts with the other honest players and the ideal functionalities used by the protocol.

We assume that this game is (R_1, δ)-bounded, where R_1 is a resource bound with time bound T_1 and at most ν sent by the environment. Note that ν is very small compared to T_1.

Game 2 We gather the honest players into a single machine.

We also insert certain delays into much of the processing in this game. The delays allow us to later modify processing without changing time usage. This simplifies the technicalities involved.

Given the time bound T_1 for Game 1, this game will have a time bound $T_2 = T_1 + poly(\nu)$. As will be seen through the rest of the proof, the constants involved are very small, so $T_2 \approx T_1$. With R_2 the same as R_1 except that the time bound is replaced by T_2, we have that Game 2 is (R_2, δ)-bound. The following result is immediate.

Lemma 12. *Game 2 is $(R_1, \delta, R_2, \delta, \delta)$-indistinguishable from Game 1.*

Game 3 In this game, we record all the plaintexts and ciphertexts as they are created. We do not run the decryption protocol, but instead compute the election result from the recorded plaintexts and the actual ciphertexts submitted for counting.

Because of completeness of the cryptosystem, this change cannot be observed by any environment.

Note that we no longer use the decryption key in this game.

Game 4 In this game, we choose for each voter a random permutation on O and apply it before we encrypt the ballot, and again before we check the return codes.

It is clear that we can use this game and the previous game to construct an adversary against R-privacy for the cryptosystem. This proves the following lemma.

Lemma 13. *If the cryptosystem is $(T_2, N, n_{tot}, L, k, \epsilon)$-secure with $n_{tot} < \nu$, this game is $(R_2, \delta, R_2, \delta + \epsilon, \epsilon)$-indistinguishable from the previous game.*

Ideal Simulator Sketch Constructing an ideal simulator is easy. Our simulator runs a simulation identical to Game 4. Whenever a permuted ballot leaks from the ideal functionality, we start the voter with the permuted ballot as input. The adversary now has three clearly defined ways of interfering with the ballot submission:

- He can delay or prevent delivery of messages through the PKI functionality.
- He can reject the ballot by refusing to sign the hash of the ciphertext, or producing an incorrect signature.
- He can send the voter incorrect or no return codes on ballot submission, and can send return codes at a time of his choosing.

It is easy to observe that all of these actions are easily detectable in Game 4, and the ideal simulator can send messages to the ideal functionality reproducing the correct effect of the return code generator's actions.

Proposition 6 is therefore proved.

5.5.3 The Decryptor

Proposition 7. *Under the conditions of Theorem 2, there exists a resource bound R_1 such that when the decryptor is corrupt, then the internet voting protocol given in Figure 11 interacting with the ideal functionalities given in Figures 5–9 $(R_1, \delta, R_2, \delta + \epsilon, \delta + \epsilon)$-realizes the internet voting functionality given in Figure 4.*

Let T_i be the time bound for R_i.

The proof proceeds as follows: We begin with a game where an environment interacts with the voting protocol and its underlying functionalities. Then we make a sequence of modifications to this game, where each modification gives a new game that is indistinguishable from the preceeding game.

Game 1 is the initial game. Game 2 is a technical step where we gather all the various honest machines into one big machine, and make various technical preparations for subsequent games.

In Game 3, random ballots are used during ballot submission, but before counting, the real ballots are encrypted yet again. The properties of the PKI functionality and D-privacy ensure that there is no observable change of behaviour.

In Game 4, we stop if the decrypted ballots do not match the ballots used to create the ciphertexts. By D-privacy, this change of behaviour is unobservable.

When we have suitably modified the initial game, we construct an ideal simulator that runs a copy of the final game. The input to the protocol machines is simulated, based on messages from the ideal functionality. Certain events in the game are translated into messages to the ideal functionality.

It will be straight-forward to verify that our machine in the final game is indistinguishable from the ideal functionality interacting with the ideal simulator and the environment. Transitivity of indistinguishability then proves that the voting protocol realizes the functionality.

Game 1 We begin with a game where an environment and a corrupt decryptor interacts with the other honest players and the ideal functionalities used by the protocol.

We assume that this game is (R_1, δ)-bounded, where R_1 is a resource bound with time bound T_1 and at most ν sent by the environment. Note that ν is very small compared to T_1.

Game 2 We gather the honest players into a single machine.

We also insert certain delays into much of the processing in this game. The delays allow us to later modify processing without changing time usage. This simplifies the technicalities involved.

Given the time bound T_1 for Game 1, this game will have a time bound $T_2 = T_1 + poly(\nu)$. As will be seen through the rest of the proof, the constants involved are very small, so $T_2 \approx T_1$. With R_2 the same as R_1 except that the time bound is replaced by T_2, we have that Game 2 is (R_2, δ)-bound. The following result is immediate.

Lemma 14. *Game 2 is $(R_1, \delta, R_2, \delta, \delta)$-indistinguishable from Game 1.*

Game 3 In this game, we run ballot submission with random ballots. When the ballot box closes, we "rerun" ballot submission with just the ballots that should be submitted, in random order. During this "rerun", we use random identities. These ciphertexts are then used for the remainder of the protocol run.

Because the PKI functionality only leaks the message lengths, and the cryptosystem has a fixed-length encoding, the adversary cannot see that ballot submission was run with random ballots. Likewise, he cannot see that ballot submission was rerun. Furthermore, the sorted order of naked ciphertexts is independent of the ballot submission order.

It follows that this game is indistinguishable from the previous game.

Game 4 In this game, if the auditor accepts, but the decrypted ballots are different from the ballots used for the "rerun", the auditor rejects anyway.

It is straight-forward to use D-integrity to bound the probability of the exceptional event. This proves the following lemma:

Lemma 15. *If the cryptosystem is $(T_2, N, n_{tot}, L, k, \epsilon)$-secure, then Game 4 is $(R_2, \delta, R_2, \delta + \epsilon, \epsilon)$-indistinguishable from Game 3.*

Ideal Simulator Sketch The ideal simulator runs a copy of Game 4, except that it does not know the real ballots upon ballot submission. However, when it wants to do the "rerun", it has received the correct ballots from the ideal functionality.

Observe that the adversary can only interfere with ballot submission by delaying or rejecting messages through the PKI functionality. The ideal simulator can mimic this effect by delaying its messages to the ideal functionality, or by sending rejection messages.

It is easy to check that the ideal functionality interacting with this ideal simulator is indistinguishable from Game 4, which means that the Proposition 7 has been proven.

5.5.4 The Auditor

Proposition 8. *Under the conditions of Theorem 2, there exists a resource bound R_1 such that when the auditor is corrupt, then the internet voting protocol given in Figure 11 interacting with the ideal functionalities given in Figures 5–9 $(R_1, \delta, R_2, \delta+\epsilon, \delta+\epsilon)$-realizes the internet voting functionality given in Figure 4.*

Let T_i be the time bound for R_i.

The proof proceeds as follows: We begin with a game where an environment interacts with the voting protocol and its underlying functionalities. Then we make a sequence of modifications to this game, where each modification gives a new game that is indistinguishable from the preceeding game.

Game 1 is the initial game. Game 2 is a technical step where we gather all the various honest machines into one big machine, and make various technical preparations for subsequent games.

In Game 3, random ballots are used during ballot submission, but before counting, the real ballots are encrypted yet again. The properties of the PKI functionality ensures that there is no observable change of behaviour.

In Game 4, when the ballots are encrypted again, ballots that should not be counted are replaced by random ballots, and the other ballots are encrypted in random order. By A-privacy, this change of behaviour is unobservable.

When we have suitably modified the initial game, we construct an ideal simulator that runs a copy of the final game. The input to the protocol machines is simulated, based on messages from the ideal functionality. Certain events in the game are translated into messages to the ideal functionality.

It will be straight-forward to verify that our machine in the final game is indistinguishable from the ideal functionality interacting with the ideal simulator and the environment. Transitivity of indistinguishability then proves that the voting protocol realizes the functionality.

Game 1 The initial game consists of some environment interacting with the protocol and its underlying functionalities.

We assume that this game is (R_1, δ)-bounded, where R_1 is a resource bound with a time bound T_1 and at most ν messages sent by the environment. Note that ν is very small compared to T_1.

Game 2 We gather all the honest players and the functionalities into one machine.

We also insert certain delays into much of the processing in this game. The delays allow us to later modify processing without changing time usage. This simplifies the technicalities involved.

Given the time bound T_1 for Game 1, this game will have a time bound $T_2 = T_1 + O(\nu)$. As will be seen through the rest of the proof, the constants involved are very small, so $T_2 \approx T_1$.

With R_2 the same as R_1 except that the time bound is replaced by T_2, we have that Game 2 is (R_2, δ)-bound. The following result is immediate.

Lemma 16. *Game 2 is $(R_1, \delta, R_2, \delta, \delta)$-indistinguishable from Game 1.*

Game 3 In this game, we run the ballot submission phase twice. First once, allowing the adversary to interact. Then once more, but this time we do not allow the adversary to interact, and we only consider those ballots that were stored in the ballot box during the first run.

Because the PKI functionality only leaks the length of messages, and ciphertexts have a fixed-length encoding, the adversary cannot detect directly that the ciphertexts transmitted during the first submission phase (that the adversary can interfere with), are distinct from those it later receives.

It follows that this game is indistinguishable from the previous game.

Game 4 In this game, we use random ballots for the first run of the ballot submission phase. In the second run, we replace the ballots that we know should not be counted by random ballots. The ballots that should be counted are used in random order.

If the environment distinguishes this game from the previous game, we can easily construct an adversary against A-privacy, which proves the following lemma.

Lemma 17. *If the cryptosystem is $(T_2, N, n_{tot}, L, k, \epsilon)$-secure, with $n_{tot} < \nu$, then Game 3 is $(R_2, \delta, R_2, \delta + \epsilon, \epsilon)$-indistinguishable from Game 3.*

Ideal Simulator Sketch The ideal simulator runs a copy of Game 3, except that it does not know the real ballots upon ballot submission. However, before the second submission phase (where the adversary cannot interfere), it knows which ciphertexts will be decrypted, and it has received the correct ballots from the ideal functionality.

Observe that the adversary can only interfere with ballot submission by delaying or rejecting messages through the PKI functionality. The ideal simulator can mimic this effect by delaying its messages to the ideal functionality, or by sending rejection messages.

We can now observe that the ideal functionality interacting with this ideal simulator is indistinguishable from Game 3 which proves Proposition 8.

6 Concrete Security

We shall consider the exact security of the voting protocol. The election parameters are based on a full-scale Norwegian election, even though the current

trials involve less than a tenth of all voters. The security parameters correspond to those being used in the current trials.

Suppose an election involving $N = 2^{21}$ voters each submitting ballots with up to $k = 2^7$ options chosen from a set of $L = 2^{13}$ options. Suppose the voters submit not more than $n_{tot} = 2^{30}$ ballots in total. The most interesting numbers are k and L, since their effect will dominate the others.

The cryptosystem will be run with $p \approx 2^{2048}$, and the NIZK proofs both use $\tau = 256$.

We shall assume that the best attack on the hash function is a brute force collision search, which means that any adversary we know against the hash function using time at most 2^{90} has success probability at most 2^{-70}. That is, the hash function is $(2^{90}, 2^{-70})$-secure.

The usual analysis says that computing discrete logarithms in our field requires about 2^{112} work. Some discrete logarithm algorithms can trade one bit of work for two bits of success probability (halving the effort reduces success probability to one fourth). The number field sieve does not work like that. When the number of relations is significantly smaller than the number of elements in the factor bases, the rank of the resulting linear system will with high probability be too small, and the algorithm will fail.

If we accept that computing discrete logarithms using the number field sieve is the best approach to solving the DDH and the SGSP problems, it seems reasonable to assume that the SGSP problem is $(2^{90}, 2^{-50})$-hard.

Finally, we shall assume that the constant involved in the Schnorr proof assumption is $\chi = 2^{10}$, and that any adversary using time at most 2^{80} is $\epsilon_{pok} = 2^{-60}$.

The secure communication and the phone functionality represent assumptions about the capabilities of an intruder, and as such no concrete security assumptions are natural.

Since $2^{90} < 2^{128} - 1$, Theorem 1 then says that the cryptosystem is $(2^{80}, 2^{21}, 2^{30}, 2^{13}, 2^7, \epsilon')$-secure, with

$$\begin{aligned}
\epsilon' &\leq (k^2 + L)\epsilon + \epsilon_{pok} + 2^{-\tau/2+2} + \frac{3n_{tot}T}{q} + \frac{N}{q^L} \\
&= (2^{14} + 2^{13})2^{-50} + 2^{-60} + 2^{-126} + \frac{3 \cdot 2^{110}}{2^{2047}} + \frac{2^{21}}{2^{2047 \cdot 2^{13}}} \\
&\approx 3 \cdot 2^{-37}.
\end{aligned}$$

Since any adversary that breaks the security properties claimed in Section 5.1 will result in a distinguisher, Theorem 2 can be interpreted as saying that any adversary limited to the above parameters and time 2^{80} will not break the above security properties except with probability $3 \cdot 2^{-35}$.

Remark. Analysis of the PKI functionality is out of scope for this paper. Also, in practice, any PKI functionality will rely on TLS, which despite recent progress is showing considerable resistance to analysis.

From a practical point of view, it seems easier today to compromise a voter's computer or the PKI protocols built on top of TLS, rather than TLS. To a

certain extent, any attacks on the PKI protocol will be subsumed by our rather generous assumptions on computer compromise built into the PKI protocol.

7 Conclusion

We have defined a novel public key encryption scheme with certain new properties. This scheme encapsulates the essential cryptographic operations required for an internet voting protocol that we also define. We show that the security of the internet voting protocol follows from the security of the encryption scheme and the properties of certain infrastructure.

The analysis of the encryption scheme requires a novel cryptographic problem related to the Decision Diffie-Hellman problem, which we have defined and discussed.

The security of the encryption scheme also relies on the Schnorr proof of knowledge, but in order to use this proof of knowledge for we have to model it. We use rather unconventional (and for some, unconvincing) techniques, but we argue, using among other things evidence from the generic group model, that these techniques are reasonable in practice. We also discuss briefly two alternatives to the Schnorr proof of knowledge.

An earlier paper [13] similar to this one was the basis for the internet voting system used in an internet voing trial during the 2011 municipal elections in Norway. This paper describes a significantly more efficient protocol that will be used in a second internet voting trial during the 2013 parliamentary elections. Compared to the previous paper, the protocol analysis is significantly improved.

Acknowledgements

The author would like to thank Kjell Jørgen Hole, the E-VALG 2011 people, the people at Scytl, Helger Lipmaa, Filip van Laenen, David Wagner, Mariana Raykova, Berry Schoenmakers and Anders S. Lund, as well as many others, for useful discussions and feedback.

References

[1] Riza Aditya, Kun Peng, Colin Boyd, Ed Dawson, and Byoungcheon Lee. Batch verification for equality of discrete logarithms and threshold decryptions. In Markus Jakobsson, Moti Yung, and Jianying Zhou, editors, *ACNS*, volume 3089 of *Lecture Notes in Computer Science*, pages 494–508. Springer, 2004.

[2] Josh D. Cohen [Benaloh] and Michael J. Fischer. A robust and verifiable cryptographically secure election scheme (extended abstract). In *Proceedings of 26th Symposium on Foundations of Computer Science*, pages 372–382. IEEE, 1985.

[3] Dan Boneh and Philippe Golle. Almost entirely correct mixing with applications to voting. In Vijayalakshmi Atluri, editor, *ACM Conference on Computer and Communications Security*, pages 68–77. ACM, 2002.

[4] e-voting security study. CESG, United Kingdom, July 2002. Issue 1.2.

[5] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[6] David Chaum. Surevote. `http://www.surevote.com`, 2000.

[7] David Chaum. Secret-ballot receipts: True voter-verifiable elections. *IEEE Security & Privacy*, 2(1):38–47, 2004.

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

[9] Ronald Cramer, Matthew K. Franklin, Berry Schoenmakers, and Moti Yung. Multi-autority secret-ballot elections with linear work. In Ueli M. Maurer, editor, *EUROCRYPT*, volume 1070 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 1996.

[10] Ivan Damgård, Kasper Dupont, and Michael Østergaard Pedersen. Unclonable group identification. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 555–572. Springer, 2006.

[11] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 152–168. Springer, 2005.

[12] Kristian Gjøsteen. A latency-free election scheme. In Tal Malkin, editor, *CT-RSA*, volume 4964 of *Lecture Notes in Computer Science*, pages 425–436. Springer, 2008.

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

[14] Kristian Gjøsteen, George Petrides, and Asgeir Steine. A novel framework for protocol analysis. In Xavier Boyen and Xiaofeng Chen, editors, *ProvSec*, volume 6980 of *Lecture Notes in Computer Science*, pages 340–347. Springer, 2011.

[15] Jens Groth. A verifiable secret shuffle of homomorphic encryptions. In Yvo Desmedt, editor, *Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 145–160. Springer, 2003.

[16] Sven Heiberg, Helger Lipmaa, and Filip van Laenen. On achieving e-vote integrity in the presence of malicious trojans. Submission to the Norwegian e-Vote 2011 tender (see also `http://eprint.iacr.org/2010/195`), August 2009.

[17] Sven Heiberg, Helger Lipmaa, and Filip van Laenen. On e-vote integrity in the case of malicious voter computers. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *ESORICS*, volume 6345 of *Lecture Notes in Computer Science*, pages 373–388. Springer, 2010.

[18] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In Dan Boneh, editor, *USENIX Security Symposium*, pages 339–353. USENIX, 2002.

[19] Antoine Joux, Reynald Lercier, David Naccache, and Emmanuel Thomé. Oracle-assisted Static Diffie-Hellman is easier than discrete logarithms. In Matthew G. Parker, editor, *IMA Int. Conf.*, volume 5921 of *Lecture Notes in Computer Science*, pages 351–367. Springer, 2009.

[20] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. Cryptology ePrint Archive, Report 2002/165, 2002. `http://eprint.iacr.org/`.

[21] Miroslaw Kutylowski and Filip Zagórski. Verifiable internet voting solving secure platform problem. In Atsuko Miyaji, Hiroaki Kikuchi, and Kai Rannenberg, editors, *IWSEC*, volume 4752 of *Lecture Notes in Computer Science*, pages 199–213. Springer, 2007.

[22] David Naccache and Jacques Stern. A new public-key cryptosystem. In *EUROCRYPT*, pages 27–36, 1997.

[23] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *ACM Conference on Computer and Communications Security*, pages 116–125, 2001.

[24] Kun Peng. A hybrid e-voting scheme. In Feng Bao, Hui Li, and Guilin Wang, editors, *ISPEC*, volume 5451 of *Lecture Notes in Computer Science*, pages 195–206. Springer, 2009.

[25] P. Y. A. Ryan and T. Peacock. Prêt à voter: a systems perspective. Technical Report CS-TR No 929, School of Computing Science, Newcastle University, September 2005.

[26] Kazue Sako and Joe Kilian. Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *EUROCRYPT*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403. Springer, 1995.

[27] Claus-Peter Schnorr and Markus Jakobsson. Security of signed elgamal encryption. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 73–89. Springer, 2000.

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

A On the Security of Schnorr Proofs of Knowledge

We have claimed that the Schnorr proof of knowledge protocol given in Figure 2 "morally" realizes the functionality in Figure 3 with a fairly tight time cost. We substantiate that claim by proving this result for our application in the generic model. That is, we prove that there exists a "morally equivalent" environment and an ideal simulator that is indistinguishable from the original environment interacting with the protocol.

This argument was first given by Schnorr and Jakobsson [27]. We redo the argument for our cryptosystem and B-privacy.

In the generic group model, players have access to a generic group oracle (in addition to any other oracles, such as a random hash oracle). The generic group oracle has a random bijection $\sigma : G \to S$, where S is a set of group element representations. The oracle responds to queries $\perp$, $s \in S$ or $s_1, s_2 \in S$ with $(\sigma(1), \sigma(g))$, $\sigma(\sigma^{-1}(s))$ and $\sigma(\sigma^{-1}(s_1)\sigma^{-1}(s_2))$, where g is some generator.

Game 1 We begin with the usual B-privacy game between a simulator and an adversary in the generic group model, where at most n_{tot} honest ciphertexts are generated. Note that the game's simulator can be considered as a composition of the Schnorr proof of knowledge protocol and some machine M. The environment therefore consists of M and the adversary.

Game 2 The first modification we make to the game is to simulate the proof of knowledge generated by M using Sim_{pok}. This change is unobservable.

Game 3 The next modification is to compute the encryptions by choosing x to be a random element from S, and computing $w_i = x^{a_{1,i}} v_i$, $i = 1, 2, \ldots, k$. This change is unobservable.

Game 4 Next, we add bookkeeping for a second representation of the group elements as tuples from $\mathbb{Z}_q^{n_{tot}+3}$. The identity is represented by the all-zeros tuple $(0, \ldots, 0)$ and the generator is represented by $(1, 0, \ldots, 0, 1)$. The element x_i used for the ith encryption is represented by a tuple $(0, \ldots, 0, 1, 0, \ldots, 0, d_{x_i})$, where the 1 is in the $i+1$th position and d_x randomly chosen. For every new element $s \in S$ seen by the oracle, the bookkeeping records a tuple (a, b, c, d) as follows:

- If s is input, it chooses c uniformly at random and records $(0, \ldots, 0, c, c)$.
- If s is the inverse of an element with record $(a, b_1, \ldots, b_{n_{tot}}, c, d)$, it records $(-a, -b_1, \ldots, -b_{n_{tot}}, -c, -d)$.
- If s is the sum of elements with records $(a, b_1, \ldots, b_{n_{tot}}, c, d)$ and $(a', b'_1, \ldots, b'_{n_{tot}}, c', d')$, it records $(a + a', b_1 + b'_1, \ldots, b_{n_{tot}} + b'_{n_{tot}}, c + c', d + d')$.

Two tuples are considered equivalent if they have equal fourth coordinates.

We note that any tuple $(a, b_1, \ldots, b_{n_{tot}}, c, d)$ recorded will satisfy $a + \sum_{i=1}^{n_{tot}} b_i d_{x_i} + c = d$.

Game 5 Next, before every group operation, we first compute the corresponding operation on the second representation. We stop the game if we ever find distinct tuples that are equivalent. It can be shown in the usual manner that if the number of group oracle queries is small, collisions are unlikely.

Analysis Suppose the protocol accepts a maliciously generated proof of knowledge (β, ν) for some auxillary information aux and element x. We know that

$$\beta = H(aux, \sigma(g), x, \sigma(g)^\nu x^\beta).$$

Since H is a random function, it is extremely unlikely that this equation holds unless H has been queried at $(aux, \sigma(g), x, \alpha)$ for some α before β was known, and

$$\alpha = g^\nu x^\beta.$$

Suppose also that x is represented by $(a, b_1, \ldots, b_{n_{tot}}, c, d)$ and α is represented by $(a', b'_1, \ldots, b_{n_{tot}}, c', d')$. We then know that

$$(\nu, 0, 0, \nu) = (a' - a\beta, b'_1 - b_1\beta, \ldots, b'_{n_{tot}} - b_{n_{tot}}\beta, c' - c\beta, d' - d\beta).$$

Since both b_i and b'_i are chosen before the hash function is queried and β is determined, we know that except with negligible probability, $b'_i = b_i = 0$, $i = 1, 2, \ldots, n_{tot}$ and $c = c' = 0$. It follows that x is represented by $(a, 0, \ldots, 0, a)$.

From this it follows that if we add code to the adversary for keeping track of the a-values for each group element produced (that is, we do not add the full bookkeeping introduced in Game 4), then for any accepted proof of knowledge for x, the discrete logarithm of x can easily be extracted from the bookkeeping. In this way we can create a new environment that interacts with the functionality and an ideal simulator that simulates the random hash function.

Since the bookkeeping is only added to the adversary, and M is left unchanged, this is clearly a "morally equivalent" environment.

The bookkeeping requires storing one element of $\mathbb{Z}_q$ for each group element stored. Therefore, if the total number of group elements seen is n, then the total memory usage is $O(n)$. The cost of storing and retrieving information is $O(\log n)$. If the number of group operations n is polynomial in the size of the group order q, then $O(\log n) = O(\log \log q)$, though the constant terms

need not be comparable. Also, for every group operation, the bookkeeping code must do one addition/negation in $\mathbb{Z}_q$, which costs $O(\log q)$. If we count the cost of a group operation as polynomial in $\log q$, a plausible estimate is that the bookkeeping time cost is at most equal to the original time cost.

We conclude that in the generic model, the Schnorr proof assumption holds for our specific environment with $\chi \leq 2$.