# *Chapter 5*

# E-Voting in Norway

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

## CONTENTS



## 5.1 Introduction

In 2008 the Norwegian parliament authorized trials of electronic voting in Norway. This lead to trials of electronic voting from home during the 2011 local elections and 2013 parliamentary elections.

The author participated in these trials from 2009 as a member of the steering group for the 2011 trials and as a consulting cryptographer for both trials.

The following is an account of the two trials from a cryptographer's point of view. The account touches on more than cryptography, since the author was involved in much more than just the cryptography.

To understand certain choices made during these trials, it is necessary to understand the Norwegian electoral system and what Norwegians consider important about elections. These topics are discussed in Sections 5.2 and 5.3.

The Norwegian government wanted to buy a suitable electronic voting system. This process is discussed in Section 5.4. Eventually, a vendor was chosen, but we were not entirely happy with the vendor's electronic voting system. Section 5.5 discusses why we were unhappy and what remedy we chose.

Eventually, the electronic voting system was deployed. The results are discussed in Section 5.6.

## 5.2 Elections in Norway

Norway has four different elections: municipal, county and parliamentary, as well as elections to Sametinget, a representative body for the Sami.[1] While the elections share certain common features, there are important differences.

Before an election, each participating political party nominates a *list* of candidates. The voter chooses one of these party lists to submit as his ballot. Before submission, the voter may modify the ballot.

- In municipal elections, the voter may modify the list by marking specific candidates (so-called personal votes) or writing in candidates from other party lists. (Note that arbitrary write-ins are not allowed.)
- In county elections, the voter may modify the list by marking specific candidates (so-called personal votes), but may not write in candidates from other party lists.
- In parliamentary elections and elections to Sametinget, voters may modify the list by reordering or deleting candidates.

For all the elections, the political parties are essentially awarded seats in proportion to the number of ballots they receive. (For parliamentary elections, multiple representatives are elected for each district.) The elections differ in how list candidates are ranked after the election.

For municipal elections, candidates are ranked by the number of times they are marked by their party's voters or written in by other parties' voters. Ballots with

[1] The Sami are Norway's indigenous people.

write-in candidates are split between the party the list belongs to and the parties of the written-in candidates. (Note that while write-in candidates typically influence which candidates are selected, they may also change the number of seats awarded to each party.)

For county elections, the political parties are awarded seats in proportion to the number of ballots they receive. Candidates are ranked by the number of times they are marked by their party's voters. (Note that there are no write-ins in county elections.)

For parliamentary elections, ballot modifications to a candidate's order (or striking) will be discarded unless more than half of all voters have made the same change for that candidate. Ballot modifications never change the candidate ordering for parliamentary elections.

## The Electoral Roll

The Norwegian electoral roll is derived from a national registry run by the Norwegian tax administration some three months before the election.

After the election, the list of who voted is considered confidential, but election researchers may be given access to it.

## The Ballot

Logically, a Norwegian ballot consists of a sequence of values. For each election, the first of these values describes the party (possibly no party, as implied by a blank ballot).

- For municipal elections, the remaining values describe candidates and are chosen from a large (up to a few thousand) set of possible values. Their order does not matter.
- For county elections, the remaining values describe candidates nominated by the party and are chosen from a small (less than one hundred) set of possible values. Their order does not matter.
- For elections to parliament and Sametinget, the remaining values describe reordering or striking, and come from a small (less than sixty) set of possible values. Their order matters.

The entire sequence of values is required to count the ballot correctly. That is, none of these elections is equivalent to a series of (simpler) independent races.

## Ballot Casting

Voters may vote in advance for roughly one month before election day. About 30% of all voters voted in advance in the 2013 parliamentary elections.

Advance voting inside Norway is usually done in polling stations (typically in the city hall or a local public library), although temporary polling stations may be organized, e.g., at universities or nursing homes. Mobile voting booths are available for those with special needs.

Voters abroad may only vote in advance at Norwegian embassies or consulates, or they may send their ballots by mail. They may not vote on election day.

On election day, voters vote at regular polling stations.

While almost all voters use official paper ballots, almost any piece of paper may serve as a ballot. In this case, the voter expresses his intended ballot simply by writing it on the piece of paper. This is mostly relevant for advance voting abroad, where voters may not have thought to acquire official paper ballots before deciding to vote.

Voters with special needs (e.g., blind voters) need help to prepare their ballot. (The paper ballots are stored in containers marked with braille writing, so that blind voters can choose the correct paper list. But blind voters cannot reliably modify their ballot.) At any time, during advance voting or on election day, voters may ask for assistance from poll workers in preparing their ballot.

### Counting

After the polling stations close, the paper ballots are counted, mostly by machines. Preliminary results usually arrive within a few hours, and accurate results are ready by next morning.

A long time ago, there were strict formal requirements for ballots. This caused many ballots to be discarded, and voters were reluctant to modify their ballots, since any mistake might invalidate the ballot. Today, the principle is that as long as the intention of the ballot is sufficiently clear, the ballot should be counted as intended.

The counted ballots are stored until after the next election. The ballots can be made available to election researchers, but they are otherwise considered confidential and the general public will not be allowed to inspect the ballots.

### Benefits of Electronic Voting

Using electronic voting machines in polling stations would have the following benefits for Norwegian elections:

- Counting time for a fully electronic election could be significantly reduced.
- Voters with special needs could modify their ballots unassisted.

For any election with a significant fraction of paper votes, counting time would not be significantly reduced. Since Norway's ballots are reasonably easy to count using optical scanners, there is little to be gained in accuracy.

Using electronic voting from home would have the following benefits for Norwegian elections:

- General access would be improved. While most people in Norway live reasonably close to a polling station, many people do not.
- Voters with special needs could modify their ballots unassisted.
- Voters abroad could vote more easily.

## 5.3 Requirements

Any voting scheme is subject to a number of requirements, often shaped by long tradition, only some of which are relevant for security.

### No Influence on Outcome

The voting scheme should as far as possible not influence the election outcome. For instance, if it is hard to modify a ballot, fewer voters will modify their ballot, which will change the outcome of Norwegian elections.

Another example of functionality that would be suspect in an electronic voting system is allowing the voter to search for candidate names in order to write their name on the ballot. This could increase the number of write-in candidates relative to the existing paper voting scheme, which could change the outcome.

### Correctness

Correctness is perhaps the most important requirement for Norwegian elections. The final count should correctly reflect the ballots cast, and moreover, the intention behind the ballots cast. There is no requirement that official ballots should be used. A blank ballot (or indeed any piece of paper) with a legible party name written on it will be counted as a vote for that party.

### Secrecy

Secrecy is important in Norwegian elections, and this is reflected in voting procedures. Secrecy of the ballot is in a sense mandatory. You are allowed to tell anyone what you voted, obviously, but you should not be able to prove that you are telling the truth. Modern technology makes this difficult to enforce, but the intention is clearly encoded in rules and regulations.

To prepare the ballot in a polling station on election day, the voter must enter an enclosure alone. There, the voter will select and possibly modify the ballot. The voter

folds the paper ballot to hide its contents before leaving the enclosure and placing the ballot in the ballot box.

One common mistake is to fold the ballot the wrong way so that its contents are visible. Polling station attendants will send any voter making this mistake back into the enclosure with instructions to come out with a correctly folded ballot.

### Coercion Resistance

Coercion prevents the free exercise of the right to vote, which is essential for democracy. Any system that makes coercion easier will be suspect in Norway.

Without secrecy, preventing coercion is impossible. This is one more reason for caring about secrecy.

Since the Norwegian ballot is so complicated, so-called Italian attacks would be possible. The number of possible ballot modifications is so large, that a random choice of modifications is likely unique. Random modification choices will tend to cancel out. Keeping the counted ballots secret is a countermeasure against this kind of attack.

While poll workers do observe straightforward attempts at coercion during ordinary paper voting, they also observe less obvious coercion. Often this takes the form of trying to help other voters, and there is no explicit coercion. If challenged, the coerced voter may not even agree that he was coerced, even though he would have voted otherwise without the coercion.

This has important implications for mechanisms that provide resistance to coercion. For instance, the mechanism must not require that the voter actively tries to prevent coercion (à la kill codes) or recover from coercion (cancelling a coerced ballot submission).

### Verifiability

Norwegians expect the government to make mistakes. But we do not expect the government to deliberately try to cheat us. This level of trust means that there is no demand for verifiability in Norway. This does not mean that verifiability is undesirable, only that any verifiability must be achieved without compromising more important properties.

## 5.4 Buying a Voting System

The municipalities are responsible for running elections in Norway. By 2008, municipalities had acquired many different election administration and paper ballot scanning systems, of widely differing quality and capability. The central government wanted to improve on this rather inefficient state of affairs, and decided to have an

election administration system developed which it could later offer to municipalities free of charge.

The Norwegian parliament had already decided to try electronic voting. The government felt that the best approach was limited trials of remote internet voting.

The two projects, election administration and internet voting, were combined into one acquisition process. Since it later turned out that no single company could assemble a good bid for both parts of the project, it seems reasonable to ask if this was a wise decision.

## The Acquisition Process

The acquisition process was organized as a "competitive dialogue," a somewhat obscure process where over a number of rounds the participating vendors would get to influence the final specification. (Eventually, many of the proposals were not submitted by individual companies, but by consortiums. The word "vendor" in the following will also refer to these consortiums.)

One curious feature of the process was that every tender would be completely transparent, from the proposed solution down to contract details such as financials. There were two slightly different reasons for this transparency.

Politically the spirit of the open source movement was important, where openness was important in and of itself. This meant among other things that the final source code license was an item of discussion.

The project organization considered transparency a vital strategy for ensuring both security and belief in security. One of the main goals of an election system is to convince the losers that they really lost. This implies that security is a necessary property for an election system, but it is not sufficient. We also need a widespread belief in the security.

Some vendors did use the competitive dialogue process to object to this transparency, even claiming that transparency was bad for security. Such arguments were rejected and the requirement was retained.

The main unsolved security problem identified by the Norwegian government prior to the competitive dialogue was that the voter's computer could be compromised. Any proposal must give the voter tools to preserve ballot integrity.

So-called return codes à la Chaum's SureVote [141] and the British CESG study [34] were a plausible solution to the integrity problem that was known to the Norwegian government, but alternative solutions were still sought. In the event, no convincing alternative was proposed.

While preserving confidentiality against a compromised computer was obviously desirable, this was not a requirement since it would most likely be very difficult without compromising usability. However, any proposal that included even a partial solution would have a significant advantage over other proposals. In the event, no

plausible solutions to preserving confidentiality against a compromised computer were proposed during the competitive dialogue.

One property much desired of any solution was understandability. Cryptography, especially the kind of cryptography usually employed in common academic voting protocols, is very hard to understand for non-experts. The Norwegian government believed that it would be easier for the public to understand and accept simpler systems. In the event, no non-cryptographic solution was proposed.

## The Proposals

The fourth round of the competitive dialogue was supposed to see preliminary submissions with sufficient detail for independent security analysis. Five proposals were submitted. One vendor submitted a detailed cryptographic proposal for part of the voting system along with sensible and correct analysis. One submission contained a partial description that allowed some analysis to be done. Several submissions were sorely lacking in details, preventing any realistic analysis.

The Norwegian government requested the assistance of the security and cryptography groups at the Norwegian universities, as well as from other groups.

Where analysis was possible, multiple issues were found, some of them serious. Where no or partial analysis was possible, some possible issues were identified, but largely these submissions were ignored by the external analysis teams. All issues identified were reported to the vendors.

The fifth and final round required complete proposals. Three complete proposals were submitted. Every proposal used return codes to protect the integrity of the ballot from a compromised voter's computer, though they used very different underlying cryptographic solutions.

The same groups that participated in analysis after the fourth round now did one more round of analysis. After minor clarifications and corrections, it was clear that two of the proposals could plausibly be fixed to provide secure solutions. One proposal could not be fixed.

One plausible proposal was significantly cheaper than the other two proposals, which made the job of choosing the winning proposal easy. (At this point it should be repeated that the government was buying both an internet voting system and an election administration system. Internet voting was a fairly small part of the total contract.)

## The Winning Bid

The winning bid was submitted by a consortium consisting of the Norwegian company Ergo, which had previously delivered voting systems to Norwegian municipalities, and the Spanish company Scytl, an electronic voting specialist.

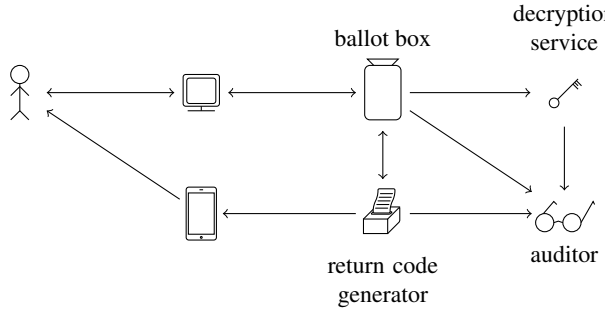


**Figure 5.1: The voter uses his computer and his phone to communicate with an infrastructure consisting of four players. The decryption service and the auditor are only active during counting.**

Scytl's proposed solution (similar to a later published version [54]) was based on Scytl's existing products. A brief overview of the different players and how they communicate is given in Figure 5.1.

The *voter* instructs the *voter's computer* which ballot to submit. The computer encrypts the ballot, signs it on behalf of the voter and then submits it to a *ballot box*.

A voter is allowed to submit multiple electronic ballots and a single paper ballot. If the voter submitted a paper ballot, that ballot is counted, otherwise the final electronic ballot will be counted.

When counting begins, any electronic ballot that should not be counted is removed from the ballot box. The remaining encrypted ballots are given to a *decryption service* where they are shuffled and rerandomized using a Scytl-developed protocol, somewhat similar to randomized partial checking. Finally, the shuffled ballots are verifiably decrypted.

To implement return codes, the voter's computer adds a special "encryption" of the ballot which is sent to the ballot box. The ballot box does some extra work on the special "ciphertext" to produce a new ciphertext that is sent to a *return code generator*. The return code generator decrypts the new ciphertext with a special key and applies a pseudorandom function to the result to derive the return code. The return code is sent to the voter's mobile phone.

As originally described, the voter had to enter a special secret number (printed on his poll card) into the voting client to get return codes, and this step was optional. Strictly speaking, this was neither usable nor secure, but the solution could obviously be modified to be both usable and secure.

## 5.5 Cryptographic Protocol

While we believed that Scytl's proposed cryptographic protocol could be made sufficiently secure, we were not entirely happy with it. In particular, we had found numerous minor problems with it during analysis, and we were not entirely sure that we had found every flaw.

Some time after it was clear that Ergo and Scytl had submitted the winning tender, the author realized that it was possible to improve upon Scytl's proposed protocol, and that it would be possible to prove positive statements about the security of the modified protocol. The modified protocol was also significantly more efficient.

In order to understand why Scytl's protocol was modified and what the modifications were, we must first look at Scytl's proposal.

### *5.5.1 Scytl's Proposal*

We can divide the execution of Scytl's proposed voting protocol into three phases: setup, ballot submission and counting.

In a system based on return codes, the main challenge is for the voting infrastructure to deduce the correct return codes to send the voter, without looking at the voter's ballot.

Scytl's solution is to construct the random-looking return code function $rc$ in such a way that the voter's computer can create encryptions of the ballot and the return code that can be related by reasonably efficient non-interactive zero knowledge (NIZK) proofs such that it is hard to create ciphertexts with inconsistent ballot and return code.

Scytl [54] describes a variant of the proposed protocol. We briefly sketch a slightly modified version of this variant's ballot submission protocol, which is sufficiently similar to Scytl's proposal for our purposes. Recall that in the Norwegian elections, a ballot consists of a party list and a sequence of zero or more list modifications.

The protocol relies on a group $G$ with a generator $g$. There is an encoding function $f$ that maps lists and list modifications to group elements. We also have a hash function $H$ and two pseudorandom functions $PRF_1$ and $PRF_2$.

#### Setup

The setup phase must produce one election public key $y_1$ satisfying $y_1 = g^{a_1}$, where $a_1$ is secret-shared among the electoral commission. A second public key $y_3$ satisfying $y_3 = g^{a_3}$ is produced, where the $a_3$ is known only by the return code generator. (Note that $y_2$ and $a_2$ will appear later.)

For return codes, two symmetric keys $k_2$ and $k_3$ are generated for the ballot box and the return code generator, respectively. Also a secret $s$ is generated for each voter.

For a per-voter secret $s$, we define the function $rc$ taking a list or list modification $v$ to a return code as follows:

$$\begin{aligned} \hat{rc}(v) &= f(v)y_1^{H(j||s)}, \\ \check{rc}(v) &= \hat{rc}(v)y_1^{PRF_1(k_2,H(s))}, \\ rc(v) &= PRF_2(k_3, \check{rc}(v)). \end{aligned} \tag{5.1}$$

The voter receives the function $rc$ in the form of a table listing every list and list modification $v$ along with the corresponding value $rc(v)$. This table is printed on the voter's poll card.

## Ballot Submission

Ballot submission begins when the voter authenticates himself to the voting system through his computer.

When the voter has authenticated himself, his computer receives the per-voter secret $s$, the election public key $y_1$ and the return code generator public key $y_3$.

The computer first sends a hash of the per-voter secret $H(s)$ to the ballot box, which then computes $PRF_1(k_2, H(s))$ and sends the result back to the computer.

The voter enters his ballot into the computer. Recall that the ballot consists of a party list and a sequence of zero or more list modifications. The sequence of length $l$ is padded with blank values to reach a given, fixed length $n$. Then each of these values is encrypted independently.

To encrypt the $i$th value $v_i$, the computer chooses a random number $r_i$ and computes a ciphertext

$$(x_i, w_i) = (g^{r_i}, y_1^{r_i} f(v_i)). \tag{5.2}$$

The computer then computes an encryption of the return code under the return code generators public key

$$\begin{aligned} \hat{w}_i &= y_3^{r_i} y_1^{H(v_i||s)} f(v_i), \\ \check{w}_i &= \hat{w}_i y_1^{PRF_1(k_2,H(s))}. \end{aligned} \tag{5.3}$$

Note that $\hat{w}_i x_i^{-a_3} = \hat{rc}(v_i)$ and $\check{w}_i x_i^{-a_3} = \check{rc}(v_i)$.

Finally, the computer generates NIZK proofs that show that the computer knows the decryption of the ciphertext, that the ciphertext decrypts to a valid vote, and that the partial ciphertext is consistent with $(x_i, w_i)$ in the above sense. To simplify the NIZK proofs, a number of secondary values are computed as well.

The computer then submits the ciphertexts $(x_1, w_1), \ldots, (x_n, w_n)$, the partial ciphertext $\check{w}$, the NIZK proofs, a number of secondary values and a signature on everything to the ballot box.

The ballot box verifies the signature and the NIZK proofs. It then passes everything to the return code generator.

The return code generator also verifies the signature and the NIZK proofs. Then it computes the $i$th return code as $PRF_2(k_3, \check{w}_i x_i^{-a_3}) = rc(v_i)$. (The return code generator must know the result of $\check{w}_i x_i^{a_3}$ for the padding value, so that it can determine the real ballot length $l$.) It sends some of the values $rc(v_1), \ldots, rc(v_l)$ to the voter's mobile phone, signs a hash of the ciphertexts $(x_1, w_1), \ldots, (x_n, w_n)$ and sends the signature to the ballot box.

Upon receipt of the signature, the ballot box stores the voter's ciphertext and sends the signature to the voter's computer.

The voter's computer reports success. When the text message with the return codes arrives, the voter consults his table to verify that he received the correct return codes.

## Counting

At the start of the counting phase, the ballot box selects the ballots that should be counted. If a voter submitted a paper ballot, all their electronic ballots are discarded. Otherwise, all but the last electronic ballot are discarded.

Before the encrypted ballots to be counted are passed to the decryption service, they are "compressed." An encrypted ballot consisting of $n$ encryptions $(x_1, w_1), (x_2, w_2), \ldots, (x_n, w_n)$ of the values $v_1, v_2, \ldots, v_n$ is compressed to the ciphertext $c = (x, w)$, where

$$x = \prod_{i=1}^{n} x_i \qquad \text{and} \qquad w = \prod_{i=1}^{n} w_i.$$

When the decryption service decrypts this ciphertext, it decrypts to the product

$$\prod_{i=1}^{n} f(v_i).$$

The group $G$ lies inside a prime field, and the function $f$ encodes values $v_i$ to field elements corresponding to small primes. By factoring the product, the values $v_1, v_2, \ldots, v_n$ can be recovered up to order.

In the event that order matters, we can use the equations

$$x = \prod_{i=1}^{n} x_i^i \qquad \text{and} \qquad w = \prod_{i=1}^{n} w_i^i.$$

which will produce the decryption

$$\prod_{i=1}^{n} f(v_i)^i.$$

Since values cannot repeat, this allows us to recover all the values in the correct order.

The decryption service first shuffles the compressed ballots using a Scytl-designed verifiable shuffle [53], then it verifiably decrypts the ballots.

The verifiable shuffle depends on the underlying cryptosystem being homomorphic: the decryption of the product of two ciphertexts is the product of the decryptions of the two ciphertexts.

We describe the shuffle only for a square number $n^2$ of ciphertexts.

The process begins when the shuffler receives the ciphertexts $c_1^{(0)}, c_2^{(0)}, \ldots, c_{n^2}^{(0)}$ to be counted from the ballot box. It rerandomizes and shuffles the ciphertexts several times, creating $c_1^{(1)}, \ldots, c_{n^2}^{(1)}$, $c_1^{(2)}, \ldots, c_{n^2}^{(2)}$, etc.

The verifier chooses a partition $P_0$ of the ciphertexts $\{c_i^{(0)}\}$ into $n$ subsets each containing $n$ ciphertexts and sends this partition to the shuffler. The shuffler reveals the corresponding partition $P_0'$ of the ciphertexts $\{c_i^{(1)}\}$, and proves that for each subset $S \in P_0$ and corresponding subset $S' \in P_0'$, the product of the ciphertexts in $S$ and $S'$ have the same decryption.

Next, the verifier chooses a partition $P_1$ of the ciphertexts $\{c_i^{(1)}\}$ into $n$ subsets each containing $n$ ciphertexts, such that for any $S \in P_0'$ and $S' \in P_1$, the intersection of $S$ and $S'$ contain exactly one ciphertext. The shuffler reveals the corresponding partition $P_1'$ of the ciphertexts $\{c_i^{(2)}\}$, and proves that for each subset $S \in P_1$ and corresponding subset $S' \in P_1'$, the product of the ciphertexts in $S$ and the product of the ciphertexts in $S'$ have the same decryption.

If the decryption service simply modifies or replaces some ballots, it is extremely unlikely that it will be able to provide the correct proofs for corresponding subsets. The decryption service could try to make modifications that cancel out, but since he is unlikely to be able to predict the partition chosen by the verifier, this will also most likely fail.

The verifier, on the other hand, will be unable to trace ciphertexts through the subsequent shuffles. This means that even though he knows both who submitted each input ciphertext and the decryption of each output ciphertext, he cannot correlate the two.

## Challenges

As is well known, understanding and analyzing cryptographic protocols is a very hard problem, and a lot of work is involved in establishing confidence in a protocol's correctness. The cryptographic discipline of provable security has been developed to make it easier to establish this confidence.

Careful analysis of the proposed protocol shows that the voter's computer does not actually prove that the ballot and the return code are consistent. While it is not difficult for the voter's computer to create an encryption of a random return code,

this is not sufficient to actually break the system, since the voter should notice the incorrect return code. We were not able to find unfixable attacks against the system during our brief analysis.

We tried to prove various properties for Scytl's proposed protocol or minor variations of it, but we were unsuccessful. Obviously, we cannot claim that it is impossible to prove interesting properties about this protocol, but it would seem to be difficult.

The conclusion was that, after various minor changes, we were unable to break Scytl's proposed protocol, but we were also unable to argue convincingly for its correctness. While the protocol was considered acceptable, it did not inspire great confidence.

### 5.5.2 Modifications

We were least happy with the ballot submission phase of Scytl's proposed protocol. A short time after it was clear that Ergo and Scytl would win the tender, the author realized that there was an alternative to Scytl's proposed way of computing and encrypting return codes. The essential insight was that we were already using a homomorphic cryptosystem, namely ElGamal, which allows us via ciphertext operations to apply certain functions to the contents of ciphertext. What if the return code function was one of those functions?

Originally, all of the modifications to Scytl's protocol were conceived at the same time, but for ease of understanding, we introduce them as two separate modifications. The first modification is about how we compute the return codes. The second modification reduces the amount of work the voter's computer has to do, and simplifies handling the per-voter secret.

#### Computing Return Codes

The author had earlier proven that the exponentiation map $\zeta \mapsto \zeta^s$ [254] could be considered a pseudorandom map when the elements $v$ were restricted to carefully chosen subsets of the group elements.

Again, we begin by defining the return code function $rc$. For a per-voter secret $s$, we define the function $rc$ taking a list or list modification $v$ to a return code as follows:

$$\begin{aligned} \check{rc}(v) &= f(v)^s, \\ rc(v) &= PRF_2(k_3, \check{rc}(v)). \end{aligned} \tag{5.1′}$$

Observe now that given the encryption $(x, w)$ of $f(v)$ from (5.2), $(x^s, w^s)$ is an encryption of $\check{rc}(v)$.

This suggests the following approach. To encrypt the $i$th value $v_i$, the computer chooses a random number $r_i$ and computes $(x_i, w_i)$ as in (5.2). It also computes the

ciphertexts

$$\begin{aligned}(\bar{x}_i, \bar{w}_i) &= (x_i^s, w_i^s)\\ (\check{x}_i, \check{w}_i) &= (\bar{x}_i, y_3^{r_i s} f(v_i)^s).\end{aligned} \tag{5.3′}$$

Note that $w_i x_i^{-a_1} = f(v_i)$ and $\bar{w}_i \bar{x}_i^{-a_1} = \check{r}c(v_i) = \check{w}_i \check{x}_i^{-a_3}$.

The computer creates three NIZK proofs. The first proves that it knows the decryption of $(x_1, w_1), \ldots, (x_n, w_n)$. The second proves that the ciphertext $(\bar{x}_i, \bar{w}_i)$ has been created by raising $(x_i, w_i)$ to the correct power. The third proves that the two ciphertexts $(\bar{x}_i, \bar{w}_i)$ and $(\check{x}_i, \check{w}_i)$ decrypt to the same value. All of these NIZK proofs are completely standard and quite efficient.

One of the significant computational improvements relative to Scytl's proposed protocol is that there is no longer any need for a computationally expensive proof that the ciphertext contains a valid ballot. (This proof is needed in Scytl's proposed protocol to prevent an attack that destroys a ballot. After our modifications, invalid ballots will almost always result in incorrect return codes, which the voter should notice.)

Two problems remain. First of all, the protocol is computationally heavy for the voter's computer. Second, handling the per-voter secret $s$ is complicated. (The original Scytl proposal envisaged the user typing it into the computer, which does not work very well. Presumably, a new infrastructure player could be introduced to handle these secrets.)

Øberg [426] analyzed essentially this protocol and proved a number of strong security properties. In fact, this protocol is slightly more secure than the protocol used in the trials. But handling the per-voter secret would have been a significant problem for deployment.

## The Ballot Box Takes Over

The goal of the next set of modifications is to reduce the computational work at the voter's computer and remove the need for distributing the per-voter secret $s$. The changes required for this do not come for free. The computational workload for the infrastructure increases slightly and the security properties of the protocol change.

There are two stages to this solution. The first is that only the ballot box should know the per-voter secrets, and it should create the ciphertexts $(\bar{x}_i, \bar{w}_i)$ and $(\check{x}_i, \check{w}_i)$. Intuitively, this seems unsafe, but careful analysis actually shows that this is actually safe.

Shifting the computational burden like this introduces a new problem. Unlike the client, the ballot box does not (and must not) know the voter's ballot, so $(\check{x}_i, \check{w}_i)$ cannot be computed as in (5.3′). Thus the second stage of our solution was to tamper with the election keys, which allows the ballot box to turn a ciphertext encrypted under the election public key into a ciphertext encrypted under the return code generator's public key.

The setup phase must now be changed as follows. First, the per-voter secret $s$

should still be generated, but all of them should be given to the ballot box. Next, a third public key $y_2$ satisfying $y_2 = g^{a_2} = y_3 y_1^{-1}$ is produced, where $a_2$ is known only by the ballot box. Note that the three decryption keys now satisfy $a_2 = a_3 - a_1$ modulo the group order.

The return code function $rc$ remains as defined in (5.1′).

To encrypt the $i$th value $v_i$, the voter's computer chooses a random number $r_i$ and computes $(x_i, w_i)$ as in (5.2). It also creates a single NIZK proof, proving that it knows the decryption of $(x_1, w_1), \ldots, (x_n, w_n)$. The computer then signs the ciphertext and the NIZK proof on behalf of the voter and passes it on to the ballot box.

The ballot box verifies the signature and the NIZK proof of knowledge. The ballot box knows the voter's secret $s$ and for each ciphertext $(x_i, w_i)$ computes two ciphertexts

$$\begin{aligned} (\bar{x}_i, \bar{w}_i) &= (x_i^s, w_i^s), \\ (\check{x}_i, \check{w}_i) &= (\bar{x}_i, \bar{w}_i \bar{x}_i^{a_2}). \end{aligned} \tag{5.3″}$$

Note that $\bar{w}_i \bar{x}_i^{-a_1} = \hat{rc}(v_i) = \check{w}_i \check{x}_i^{-a_3}$, since

$$\check{w}_i \check{x}_i^{-a_3} = \bar{w}_i \bar{x}_i^{a_2} \bar{x}_i^{-a_3} = \hat{rc}(v_i) \bar{x}_i^{a_1} \bar{x}_i^{a_2 - a_3} = \hat{rc}(v_i) \bar{x}_i^{a_1 + a_2 - a_3} = \hat{rc}(v_i).$$

The ballot box also produces two NIZK proofs showing that it did its computations correctly. These NIZK proofs are completely standard and quite efficient.

### 5.5.3 The Modified Protocol

We now give a brief summary of the modified protocol [255].

#### Setup

The setup phase must produce one election public key $y_1$ satisfying $y_1 = g^{a_1}$, a ballot box public key $y_2$ satisfying $y_2 = g^{a_2}$ and a return code generator key $y_3$ satisfying $y_3 = g^{a_3}$. The three public keys satisfy $y_3 = y_1 y_2$. The election decryption key $a_1$ is secret shared among the electoral commission, while $a_2$ is given to the ballot box and $a_3$ is given to the return code generator.

For each voter, a secret number $s$ is generated and given to the ballot box. Also, for each voter, a random injective function $h$ from $\{f(v)^s\}$ (where $v$ ranges over all the lists and list modifications) to the set of return codes is chosen. These functions are given to the return code generator.

For a per-voter secret $s$ and random injective function $h$, we define the function $rc$ taking a list or list modification $v$ to a return code as follows:

$$\begin{aligned} \check{rc}(v) &= f(v)^s, \\ rc(v) &= h(\check{rc}(v)). \end{aligned} \tag{5.1‴}$$

The voter receives the function $rc$ in the form of a table listing every list and list modification $j$ along with the corresponding value $rc(v)$.

## Ballot Submission

Ballot submission begins when the voter authenticates himself to the voting system through his computer.

When the voter has authenticated himself, his computer receives the election public key $y_1$.

The voter enters his ballot into the computer. Recall that the ballot consists of a party list and a sequence of zero or more list modifications. The sequence is padded with blank values to reach a given, fixed length. Then each of these values is encrypted independently.

To encrypt the $i$th value $v_i$, the computer chooses a random number $r_i$ and computes a ciphertext

$$(x_i, w_i) = (g^{r_i}, y_1^{r_i} f(v_i)). \tag{5.2'''}$$

The computer generates a non-interactive zero knowledge proof of knowledge that shows that the computer knows the decryption of the independently encrypted ciphertexts.

The computer then submits the ciphertexts, the NIZK proof and a signature on everything to the ballot box.

The ballot box verifies the signature and the NIZK proof. For each ciphertext $(x_i, w_i)$, it then computes a ciphertext and a partial ciphertext:

$$\begin{aligned} (\bar{x}_i, \bar{w}_i) &= (x_i^s, w_i^s), \\ \check{w}_i &= \bar{w}_i \bar{x}_i^{a_2}. \end{aligned} \tag{5.3'''}$$

It also creates two NIZK proofs showing that the ballot box has computed these values correctly.

The ballot box then sends everything it received, the ciphertexts and partial ciphertexts and the NIZK proofs to the return code generator.

The return code generator verifies all the NIZK proofs and the voter's signature, then for each ciphertext $(\bar{x}_i, \check{w}_i)$ computes the return codes using the formula

$$h(\check{w}_i \bar{x}_i^{-a_3}).$$

If any of these computations fail (which can only happen if $\check{w}_i \bar{x}_i^{-a_3}$ is not in the domain of $h$), the return code generator knows that the original ballot is invalid and informs the ballot box of this fact. In this case, the ballot box will reject the ballot and inform the voter's computer that the ballot was rejected.

Otherwise, a subset of the return codes is sent to the voter's phone. The return code generator then signs a hash of the encrypted ballot (which consists of $n$ ciphertexts $(x_1, w_1), \ldots, (x_n, w_n)$ and a NIZK proof of knowledge $\pi$)

$$H(\text{voter identity}, (x_1, w_1), \ldots, (x_n, w_n), \pi) \tag{5.4}$$

and sends this signature to the ballot box, which verifies it and sends it on to the voter's computer.

When the voter's computer receives a valid signature on the encrypted ballot it submitted, it informs the voter that the ballot has been correctly submitted.

When the voter's phone receives the return codes, the voter uses the function *rc* to verify that the return codes match the submitted ballot. When the voter's computer says that the ballot has been correctly submitted, the voter accepts the ballot as cast.

### Counting

Counting proceeds exactly as in Scytl's original protocol. The only minor difference is that the role of the auditor was clarified.

The auditor receives the entire contents of the ballot box. It also receives a list of every ballot seen by the return code generator, in order. The auditor verifies that the two parties agree on which ballots were submitted.

Next, the auditor receives a list of ballots that the decryption service received from the ballot box. Based on the contents of the ballot box, the auditor recomputes which ballots should be counted and compares this list with the list received by the decryption service.

Finally, the auditor participates in the decryption service's verifiable shuffle and also verifies the final decryption.

### Verifiability

The protocol can be modified to provide limited verifiability. The voter's computer presents the hash of the submitted ballot (5.4) and the return code generator's signature to the voter. The ballot box publishes a signed list of hashes of accepted ballots. Finally, the auditor verifies the ballot box's list.

The voter can now download the ballot box's list of hashes and verify that the hash displayed by the computer is present in the list. In the event that the hash is not present, the return code generator's signature allows the voter to complain convincingly.

### What Was Achieved?

The main achievement is that it is possible to prove clearly stated security claims about this protocol relative to reasonable cryptographic assumptions. (Though there remains a technical problem related to the proof of knowledge [256].) A secondary achievement was that the computational load was significantly reduced.

Provided that at most one of the infrastructure players is corrupt and the voter follows the protocol correctly, we can prove that:

- If the auditor accepts the result, at most one ballot per voter was counted.
- If the voter accepts the ballot as cast as intended, and does not later revote or complain about a forgery, the ballot is counted as intended up to certain changes to the list modifications.
- 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.
- A corrupt return code generator learns the number of list modifications in each ballot submission, and if a voter submits multiple ballots, learns where these ballots differ.

We developed attacks against the protocol to prove that the above claims cannot be significantly strengthened.

The proof is a mix of classical protocol analysis techniques and standard mathematical analysis relying on variants of the Decision Diffie–Hellman problem.

The Decision Diffie–Hellman variants are used to show that no single infrastructure player can see the ballots encrypted by honest computers. A crucial ingredient is the computer's proof of knowledge which prevents a corrupt ballot box from using the return code generator as a decryption oracle.

During ballot submission, the voter's computer will not accept a ballot unless both the ballot box and the return code generator has seen the ballot. This ensures that the auditor can detect any cheating by either the ballot box or the return code generator. Furthermore, the auditor can ensure that the decryption service receives the correct encrypted ballots.

The proofs produced by the decryption service ensures both that the decryption service cannot cheat and that the auditor cannot learn anything about particular encrypted ballots.

If we accept the basic premise underlying the security proof (that at most one infrastructure player is compromised), the cryptographic protocol ensures correctness and secrecy, and it makes a decent attempt at providing coercion resistance. Even though verifiability is not important, it can be modified to provide limited verifiability. The cryptographic protocol was therefore a good match for Norwegian elections.

### Further Improvements

After the 2011 election, the author developed a second version of the protocol [256] (incorporating ideas worked out by Lund [373]) that significantly reduced the computational effort required. There are two main ideas behind this improvement.

First, instead of encrypting every ballot value separately, all the values were included in a single encryption. Technically, the encryption key now consists of $n$ values $y_{11}, y_{12}, \ldots, y_{1n}$, and given a (padded) ballot $v_1, v_2, \ldots, v_n$, the computer chooses a single random number $r$ and computes

$$(x, w_1, w_2, \ldots, w_n) = (g^r, y_{11}^r f(v_1), y_{12}^r f(v_2), \ldots, y_{1n}^r f(v_n)). \tag{5.2''''}$$

The ballot box key $a_2$ must now be split into $n$ parts $a_{21}, a_{22}, \ldots, a_{2n}$, and the ballot box now computes two ciphertexts

$$\begin{aligned}(\bar{x}, \bar{w}_1, \bar{w}_2, \ldots, \bar{w}_n) &= (x^s, w_1^s, w_2^s, \ldots, w_n^s),\\ (\check{w}_1, \check{w}_2, \ldots, \check{w}_n) &= (\bar{w}_1 \bar{x}^{a_{21}}, \bar{w}_2 \bar{x}^{a_{22}}, \ldots, \bar{w}_n \bar{x}^{a_{2n}}).\end{aligned} \tag{5.3''''}$$

The return code generator's key $a_3$ is also split into $n$ parts $a_{31}, a_{32}, \ldots, a_{3n}$, and the return code generator computes the $i$th return code as

$$h(\check{w}_i \bar{x}^{-a_{3i}}).$$

Second, the NIZK proofs were replaced by batch variants. This amounts to a significant performance improvement.

The security proof was also significantly improved, and we were able to improve our security claims through better modelling of certain underlying infrastructures.

## 5.6 Deployment

As 2010 progressed it became clear that remote internet voting was politically controversial. The main objection was coercion, and the politicians were not convinced by the countermeasures planned (i.e., revoting). Most opposition parties declared loudly that they would vote against remote internet voting. As the debate unfolded, it turned out that the largest (by far) of three governing parties was also essentially opposed to remote internet voting.

Curiously, even though there was a clear majority in parliament essentially opposed to remote internet voting, the three government parties (which commanded a majority in parliament) voted to go through with trials during the 2011 local elections.

Due to political controversy, the decision to hold a new trial in 2013 was not made immediately after the 2011 trial. Until the second half of 2012, little work was done on preparing a new trial. When the government eventually did decide to hold a new trial, there was limited time for new developments.

### *5.6.1 The 2011 Election*

A small number of municipalities were selected for the 2011 trial, with approximately 160,000 voters in total. Two government organizations were selected to run

the ballot box and the return code generator, while the responsible government department would run the decryption service. A professional election observer was hired to run the auditor.

Scytl produced the implementation of the software. The software that would run on the voter's computer was delivered via a web browser. The user interface was a fairly standard web application, but when the voter pressed the button to submit his ballot, that ballot was sent to a Java applet running in the browser, which would then run the voting protocol and deal with all the cryptography. The Java applet did not have a user interface and was, provided the voter's web browser was correctly configured, hidden from the voter.

The trial was largely uneventful. Most of the mishaps were in the election administration system. There was one very interesting problem with the internet voting system, however. Because of problems at the printer service responsible for printing poll cards, incorrect return codes were printed on some poll cards. This mainly affected one of the larger municipalities participating in the trials.

Though there exists no exact data on the number of errors, the best available estimate is that about 1% of the poll cards in the affected municipality were printed with incorrect return codes. Out of the nearly 50,000 eligible e-voters in that municipality, about 8,500 voters chose to cast an electronic vote.

It seems reasonable to assume that opting for electronic voting and receiving a misprinted poll card are independent events. In other words, if our assumptions hold, we would expect that about 85 voters both voted electronically and received a misprinted poll card. During voting, misprinted poll cards caused 74 voters to call the election help desk to report incorrect return codes. That 74 out of an estimated 85 voters noticed the incorrect return codes is a remarkably high number. (Coincidentally, during earlier pre-election pilots there was a self-reported 90% verification rate, which is remarkably consistent with the above numbers.)

Unfortunately, the estimate on the number of misprinted poll cards is too uncertain to say anything about how efficient return codes are as a security mechanism for detecting certain attacks. The best we can say is probably that if a corrupt computer just tampers with the ballot, the probability of discovery may very well be large.

During the local elections, 28,001 voters voted electronically. A total of 55,785 electronic ballots were cast in the two available contests (municipal elections and county elections). Of these, 2428 ballots were cancelled by re-voting, of which 653 by a paper vote. This makes up 4.35% of the votes.

While this does indicate that many voters were aware that they could re-vote, it does not indicate that voters in general could use this countermeasure against coercion.

### 5.6.2 The 2013 Election

The principal problem with the 2011 trial was the Java applet used to do the cryptography in the voter's computer. A significant number of voters found it hard to establish and maintain a working Java web browser installation. Java in the browser was also a significant general security liability.

While the desirability of Java had decreased significantly since 2010, the effective (and recognized) capabilities of JavaScript with respect to cryptography had increased significantly over the same time period. Therefore it was decided to develop a JavaScript implementation of the cryptography for the voter's computer.

It was also decided to use the further improvements to the cryptographic protocol described at the end of Section 5.5.3. Since the part that ran on the voter's computer had to be completely redone anyway, the effort involved in modifying the other parts of the system was modest. The performance improvements would significantly benefit a JavaScript implementation.

While verifiability was never important for the politicians or the general public, it was still felt that improving verifiability would be worthwhile. This was implemented more or less as described in Section 5.5.3.

Another area for improvement was the verifiable shuffle used during decryption. It was felt that a proper non-interactive proof of shuffle would be an improvement, and some work was done to identify a suitable shuffle to implement or buy an implementation of. Unfortunately, this turned out to be impractical and was not done before the 2013 trial.

#### A Bug with Consequences

About half-way through advance voting, a "statistical anomaly" was discovered with some test data. After a brief investigation, a serious bug was found in Scytl's implementation.

For a long time, cryptography in JavaScript was difficult since there was no standard source for randomness. Eventually, this was solved by entropy gathering code and documented interfaces in modern browsers providing pseudorandomness suitable for cryptography.

Scytl's JavaScript cryptography relied on a deterministic pseudorandom bit generator seeded with randomness extracted by entropy gathering code and delivered by whatever randomness sources were available in the browser. While one could imagine simpler constructions, this is a reasonable architecture.

Unfortunately, Scytl's generator implementation replaced its state with a fixed, public state after every invocation. The implication is that the first time Scytl's cryptography used the generator, it would provide proper randomness. But for any subse-

quent use, the generator would provide the exact same response,[2] and this response was completely predictable by anyone.

Looking at the equations in Section 5.5.3, we see that a single random number $r$ appears in (5.2$'''$). Since the first use of the generator returns proper randomness, one could hope that the encryption would be properly done.

However, due to the way the random number was sampled, the generator was often used more than once, and the result would be a fixed random number. The consequence was that about 60% of all ballots submitted before the bug was fixed were encrypted with the same, known random number. Anyone observing any such ciphertext could trivially decrypt it.

Unfortunately, the remaining ballots were not secure. The voter's computer also generates a proof of its knowledge of the ballot encrypted. It turns out that when this proof is generated with predictable randomness, it reveals the randomness $r$, which in turns reveals the ballot. (A Schnorr proof is used to prove knowledge of the random number $r$ used for encryption, but when you know the randomness used in the Schnorr proof, you can easily compute $r$.) The consequence was that anyone observing any such ciphertext could easily decrypt it.

While all of this was bad, it was not yet a disaster. The system was designed so that the cryptography protected against the compromise of the ballot box, the return code generator and the auditor. But when these players were not compromised, the cryptography was no longer a single point of failure. For example, the encrypted ballot was sent from the voter's computer to the ballot box through an encrypted channel (TLS), so no eavesdropper would see the ciphertexts.

The ballot box and the return code generator were well-protected systems and very few people (none of which were Scytl employees) had access to them. After the bug was discovered, the number of people who had access was reduced even further and security was tightened even further. A subsequent audit of all logs concluded that the encrypted ballots had not leaked. During counting and auditing, great care was taken to ensure that encrypted ballots did not leak.

The conclusion was that this had been a cryptologic disaster, since the cryptology completely failed to protect the confidentiality of the ballots. However, fortunately it was only an electoral near-disaster, since other security measures allowed us to conclude that no ballots leaked.

### Not All Bad Things Must Come to Pass

During the response phase, we recalled that the cryptologic specification [256] specified that the return code generator should compute the hash to be signed as in (5.4). When the computer's random numbers are replaced by fixed values, this hash depends only on the voter's identity and his ballot, nothing else.

[2]The generator was periodically reseeded, but this seems not to have saved the day.

This would not be an extra problem for the system as originally designed [256]. But after the design, the system had been modified to provide limited verifiability as described in Section 5.5.3. And as part of these modifications, the hashes computed by the return code generator had been published on the internet.

Based on the available evidence, it did at first look as if several thousand voter-ballot pairs had been published on the internet, essentially in clear text.

Fortunately, it quickly transpired that Scytl's implementation deviated (a perfectly harmless and accepted deviation) from the specification by also including an authentication token in the hash. This authentication token contained sufficient randomness to erase any connection between the ballot and the hash. There was no electoral disaster.

In passing, it is interesting to note that publishing the hashes actually breaks the security proof. Adding some randomness before hashing (like Scytl's implementation did) makes the proof work again.

### Why Did the Bug Go Undiscovered?

Any manual testing of Scytl's generator (e.g., by invoking the generator repeatedly from the JavaScript console) would have revealed the bug. A thorough code review should also have revealed the bug. A visual inspection of any one of the numerous test datasets generated during testing would also have revealed the bug. (In fact, the bug was discovered essentially by visual inspection of test data, but far too late.)

It is clear that this bug should have been discovered, but it was not discovered. Why?

Clearly, such a vital piece of cryptographic code should have been carefully written, but it was not. Scytl's internal code reviews failed to spot the bug.

The Norwegian government hired contractors to do an independent code review of critical cryptographic code. Unfortunately, due to time limitations, the code that would run on the voter's computer was excluded from the review. To the author, it seems likely that the contractors would have found the bug if they had looked at the code.

Before the 2011 elections, numerous test datasets were generated and a large number of statistical tests were applied to the ciphertexts. Due to the limited time available, these tests were not repeated ahead of the 2013 elections. It is certain that the statistical tests used in 2011 would have discovered the bug.

Obviously, the bug should never have been in the code. But it seems clear that if the decision to do trials in 2013 had been made in a more timely fashion, this bug would have been discovered.

## 5.7 Concluding Remarks

### Politicians Worry about Coercion

Before the two trials, the politicians' main worry was coercion. There is no solid evidence that coercion would be a problem for a nationwide election, but the two trials and their evaluation did not provide any evidence to the contrary, namely that coercion would not be a problem.

Norwegian voters abroad are allowed to vote by mail, a form of voting highly susceptible to coercion. It is the author's opinion that if coercion is a real problem, Norwegian voters abroad will be better served by internet voting than by the current system.

The then government lost its parliamentary majority in the 2013 elections, and the new majority installed a minority government. The new government has decided not to have any further trials of this electronic voting system. In fact, there seems to be a complete lack of enthusiasm for electronic voting in general, and there are currently no plans for electronic voting of any kind.

### The World Has Changed

Since the system was designed, the world has also moved on. The system relies absolutely on the independence of the voter's computer and his phone, and in particular, that if the voter's computer is compromised, the voter's phone will not be compromised. This was a reasonable assumption in 2009–10. What about today?

Many modern phones will now happily forward text messages to their owner's computer, and this forwarding is relatively easy to turn on. If a compromised computer can prevent the return code generator's text message from reaching the voter, the compromised computer will be able to vote without the user noticing.

Furthermore, modern phones have begun to cooperate more closely with computers. This means that if the voter's computer is compromised, it is much more likely that the voter's phone could also be compromised.

Finally, it turns out that many people now use modern phones instead of computers. Forcing these people to use a computer to vote will be quite unnatural (which was an issue during the 2013 election), but allowing them to vote from their phone compromises the protocol's security properties.

Since the underlying assumptions may no longer be valid, it seems reasonable to doubt the security of the system. On that basis, it seems perfectly reasonable not to do any further trials.

### Nobody Cares about Bugs

Curiously, nobody cared about the near-disaster caused by the bug in the random number generator. The closest the bug came to being discussed in the press was a

complaint about the title of the government's press release. It is clear that people in general and politicians in particular do not care about the same things as security experts and cryptographers.