Cast-as-Intended Verification in Norway

Jordi Puiggalí Allepuz, Sandra Guasch Castelló

Scytl Secure Electronic Voting
08006 Barcelona, Spain
{jordi.puiggali | sandra.guasch}@scytl.com

Abstract: The Norwegian Ministry started an initiative to implement Internet-voting trials during the municipal elections in 2011. One of the security requirements of the chosen e-voting system to not to put any trust in the voting client: a malicious application controlling the voting client should not be able to modify the voting options selected by the voter without being detected. This paper describes the voter verification return-code scheme that was implemented for this project. Furthermore, this paper explains the implementation details of the final solution and the workflow of the system during the different election phases. The aim of this paper is to provide a general overview of the cast-as-intended scheme implemented in eValg2011.

1 Introduction

In August 2008, the Norwegian Ministry started a project whose initial target was to implement remote electronic voting trials in selected municipalities during the municipal elections in 2011. The final objective was to introduce the system throughout the country in subsequent elections.

The eValg2011 voting platform was successfully used in ten municipalities during the municipal and county elections in 2011. Voters in these municipalities had the opportunity to vote on the Internet from their homes. In total, 53,481 votes were cast within an electoral roll of about 165,000 voters (ten municipalities), representing 73% of the advance votes and 16.6% turnout when compared to the federal census. Authorities plan to use the same voting platform in future municipal elections and referendums.

Many of the e-voting system's security requirements [EV09] to be implemented for the eValg2011 project were defined during the bidding phase. Specifically required was the ability to detect potential vote manipulations by a malicious voting client when casting a vote. Therefore, absolute trust in the voting client software was not mandatory.

In remote electronic elections, the voting client software is generally in charge of receiving the voting options chosen by the voter and encrypting them before sending the vote to a server, meaning that voters have to trust that the voting client is not going to change their selections before being encrypted. However, in case the voting client would do it, the probability of being detected is very low. Cast-as-intended verification methods have been designed to prevent such deception: voters do not need to trust the voting

client software to encode the selected voting options properly, since they can audit the process. This has been achieved in the eValg2011 project by using a cast-as-intended verification scheme based on using return codes.

The aim of this paper is not to describe the full cryptographic voting scheme implemented in the eValg2011 voting system, only the cast-as-intended verification scheme implemented in the system. This paper starts by describing the differences between the initial protocol proposed by Puiggali-Guasch [PG11] and the final protocol implemented in the eValg2011 voting system. It also describes how the design of some parts of the verification mechanisms (mainly the return codes) evolved during the project until reaching the final design used in the 2011 elections.

This paper is organized as follows: In section 2, the existing proposals for cast-as-intended verification are presented. Section 3 briefly presents the changes made to the original scheme for the eValg2011 project. Section 4 shows an overview of the voting system as seen by the voter. Section 5 presents the building blocks of the underlying protocol designed for the cast-as-intended verification mechanism. Section 6 explains the election configuration process. In section 7, the voting phase is presented. Section 8 shows the SMS formats used to provide the voters values for the cast-as-intended verification, and the paper concludes with some final remarks in section 9.

2 Cast-as-Intended in Remote Voting

There are mainly two different approaches for providing cast-as-intended verification in remote voting: methods based on challenging the voting client and methods based on using return codes.

In methods that challenge the voting client, such as the one implemented in the Helios system [Ad08], the voting application commits first to the encrypted vote before it is cast and asks the voter later if she wants to verify the correct encryption of her choices before casting the vote. The commitment is usually the hash value of the encrypted vote that is shown at the top of the voter screen in a user-friendly format (e.g., base64 text encoding). If the voter decides to challenge the system, the voting application discloses the encryption parameters. The voter can then reproduce the same encryption operation of her voting options to verify if the resulting ciphertext has the same hash value as the one committed by the voting application. To perform the encryption and verification of the commitment, the voter can use a tool provided by any independent, trusted party, or the voter can just send the commitment and disclosed information to an external auditor along with the selected voting options. Each time the voter challenges the system, the encrypted vote is discarded and the voter is allowed to change the intent and cast a new one. The challenging process is shown each time before casting a vote. Therefore, the voter can challenge the system as many times as requested.

The systems based on return codes require sending a special voting card to voters in advance of the election. This card contains a list of short codes (e.g., four digit numbers) correlated to the possible voting options. These voting cards are unique and different for each voter and therefore, voters never have the same codes for their voting options. The verification process is usually implemented after casting the vote. In this case, the voting server usually performs a cryptographic operation over the cast vote that generates a code that is returned to the voter. The voter then checks in the voting card if the received code has the same value as the code present on the card for her selected choice. Within the return codes-based systems, it is possible to distinguish between two systems, one that includes an additional code used to cast the vote on the same voting card [St07], [MSP09], [MMP02], [Ch01], [Ce02], [VZ05], [HS07], [CCE11] (known as pollsterless or pre-encrypted ballot systems) and one that does not include this code [PG11], [Li11].

The eValg2011 voting system was based on the latter, and, more specifically, it is a variation of the Puiggali-Guasch proposed scheme.

3 Changes Made Over the Original Scheme

The modifications made to the Puiggali-Guasch scheme to develop the eValg2011 project were mainly focused on moving cryptographic processes implemented in the voting client to the voting servers.

In the original Puiggali-Guasch proposal, the voting client implements a set of cryptographic operations over the voting options to generate a special ciphertext with deterministic properties, which allow for the generation of the return codes of the selected voting options contained in the encrypted vote. This ciphertext is sent to the voting server along with the encrypted vote and a proof of content equivalence between this special ciphertext and the encrypted vote. A set of cryptographic operations are implemented by the voting server and another independent server (known as the return code generator) for generating the return codes.

In the eValg2011 protocol, the voting client does not generate any special ciphertext for the return codes; it simply encrypts and casts the vote. The special ciphertext with deterministic properties is generated in the voting server by executing a set of cryptographic processes over the encrypted vote cast by the voter. This ciphertext is then forwarded to the return code generator server which applies a second set of cryptographic operations for generating the return codes. This change implied a complete re-design of the cryptographic operations and content equivalence proofs implemented by the scheme. The re-design was lead by Kristian Gjøsteen, and its security is further discussed in [Gj10].

There are several advantages that this re-imagined scheme offers:

- A reduction of the cryptographic operations implemented in the voting client: the voting client does not generate the special ciphertext nor the proof of content equivalence of the original scheme; it only encrypts the vote.
- The improvement in usability of the voting process: the voter iss not required to introduce any voting card identifier for verifying the return codes (as required in the original scheme).

However, these advantages have some side effects:

- Special measures must be implemented to prevent any collusion between the voting server and the return code generator, otherwise both servers could compromise the voter privacy.
- The number of cryptographic operations performed in the servers increases substantially, since the operations initially executed in the voting terminal for generating the special ciphertext, must be now executed by the servers.

It is of special importance to mention that one of the security requirements under which both schemes were designed was that one single component or participant in the voting system (voting client, voting servers, etc.) should not be able to cheat in the election process without being detected: i.e., one single component should not be able to act in a different way that what is described in the protocol in order to break voter privacy or affect the integrity of the election. The way this is fulfilled is further analyzed throughout the following sections, as well as in [SVK11].

4 Overview of the Voting Process

In order to better understand the return code scheme implemented for the eValg2011 project, we will present a brief overview of the voting process as seen by the voter:

Before or during the voting phase, the voter receives a voting card containing the return code values assigned to each possible voting choice, which will be used to verify that the voter's selections have been correctly received by the voting server.

During the voting process, the voter is authenticated by the system. Once the eligibility of the voter has been verified, the voter receives her credentials, which will be used to digitally sign her vote. The voter uses a voting Java applet to select her choices. Once the voter has finished making her selection, the completed ballot is encrypted using an election public key and digitally signed using the voter credentials. The vote is then sent to a voting service (known as the vote collector server or VCS), where it is stored in the electronic ballot box. The voting service forwards the vote to a validation service (called the return code generator or RCG), where the return codes representing the selected voting options are generated and then sent to the voter via SMS message. The voter uses the voting card to verify that the return codes correspond to her completed ballot.

The cast-as-intended scheme can be split in two levels: the core level, where the cryptographic operations are implemented, and the presentation level, which manages how the results of the cryptographic operations are shown to the voter.

In the core level, each voting option is linked to a unique return code value. However, at the presentation level, unique return code values could be linked to a new, shared return code in order to improve the usability of the voter verification process. For instance, the presentation level could link the unique return code of a candidate obtained from the core level to a generic return code signifying the position of the candidate inside the party list. Therefore, the number of return codes managed by the voter is drastically reduced: all the candidates having the same position in different party lists would have the same position return code.

Currently, the eValg2011 system can generate three different types of return codes for voters at the presentation level:

- Unique return codes for each voting choice: they are a direct representation of each return code generated at the core level.
- Position return codes related to the position of voting options within a list of options: in this case, core level return codes of different candidates will share the same position return code if they are located in the same position on a selection list (e.g., the first candidates of different party lists will share the same return code representing the first position within a list). These position return codes are usually combined with unique return codes identifying the list that the candidate position is related to (i.e., every candidate is represented by a tuple composed by a unique party return code and a position return code).
- No return codes, but information related to the number of selections made within a list: this approach is used when the voter makes selections within different lists. In this case, the presentation layer combines the use of a unique return code representing the list (e.g., a party return code) with the number of selections made within the list (i.e., an explicit message documenting the number of selections made instead of candidate or position return codes). This is the specific scheme used in the municipal and county elections conducted in 2011 as part of the eValg2011 project.

All these return code representation options are configurable at the voting system and have been tested in different trials before the 2011 municipal and county elections.

For simplicity, we will describe the system using the unique return code representation used at the core level as reference. The different return code representations at presentation level are discussed in the sections related to the generation of the voting cards and return codes sent by SMS. The usability and security implications of the approach of working with each return code representation at presentation level will be discussed in Section 8.

5 Building Blocks

The return code generation scheme is composed of the following building blocks:

Underlying Cryptosystem: The vote is encrypted using a probabilistic encryption algorithm suitable for use with zero-knowledge proof schemes [MOV96]. In this specific implementation, the encryption algorithm is ElGamal [El84]. The election cryptosystem is composed of three public parameters: *p*, *q*, *g,* with *p=2q+1*; an election public key h_e; and an election private key x_e defined in the ElGamal scheme.

We denote a vote composed of several encrypted voting options as $v_{opt_i} = (a_i, b_i) = (g^{r_i}, v_i \cdot h_e^{r_i})$, where the encryption exponents r_i are chosen as random values from Z_q, the operations are done modulo *p,* and each value v_i represents a voting option.

Besides the election keys, two ElGamal key pairs are used for the return code generation process: one for the VCS (h_{vcs} , x_{vcs}) and one for the RCG (h_{rcg} , x_{rcg}). Both key pairs are defined by the same parameters (*p*, *q*, *g)* of the ElGamal scheme as the election key pair For the purpose of the protocol, these keys have the following mathematical relationship: $x_{rcg} - x_{vcs} \equiv x_e \pmod p$.

The security threats and countermeasures regarding this key relationship are discussed further in [Gj10].

Voter Secret Parameter: In order to be able to generate different return codes for different voters, the voting options cast by a voter are raised to a value *s* that is different for each voter (voter secret parameter) in the VCS, in order to get a *personalized,* random encryption for each voter. These values are used during the configuration and the voting phase. Therefore, they cannot be generated on-the-fly and must be stored in a secure way. A hardware security module (HSM) could be use to securely store this information. However, there may be millions of values to store (one per voter), which could be a problem. To solve this, only a private key is securely stored and *s* values are derived in the VCS using this cryptographic key and a pseudorandom function. Therefore, the output of this pseudorandom function will be random for someone without the cryptographic key.

In this specific implementation, the pseudorandom function used to generate the voter secret parameter *s* is a symmetric encryption algorithm (AES - CBC mode [FP01]). Therefore, the voter secret parameter *s* is generated as the AES encryption of a random voter identifier in the election (*voterID*) using a secret key stored in the VCS, K_{vcs}. The *voter ID* must be padded or transformed in such a way that it is long enough to generate a 2048-bit value for *s*. It is important to have a large value for *s*, since it is in charge of protecting the secrecy of the vote in several specific steps of the process.

Zero-knowledge Proofs: Return code values are generated with the collaboration between the VCS and the RCG, in the sense that the first makes some partial calculations and sends them to the second, which generates the final values. This way, the knowledge needed to generate valid return codes is split into two independent components of the voting system, so that both have to be compromised in order to cheat the voters. However, each component has to prove to the other one that it is following the protocol properly. If not, one component would be able to cheat in the election. For example, VCS could use the vote of one voter to make the RCG generate the return code values for another voter. Therefore, return codes corresponding to the selections made by the first voter are sent to the second one, invalidating the first voter's privacy. Non-Interactive zero-knowledge proofs (like Schnorr proofs in [Sc91]) are used by the VCS to demonstrate to the RCG that the partial calculations actually belong to a specific valid vote.

6 Election Configuration Process

The main objectives of the election configuration process are to create the keys used for computing the return codes and to generate the voting cards used by the voters to verify the correct representation of their voting options inside the encrypted vote.

6.1 Generation of Election Keys

The eValg2011 voting system mainly uses two different sets of keys for implementing the cast-as-intended verification scheme:

- *Asymmetric keys*: used to protect the privacy of the vote.
- *Symmetric keys*: used to generate a deterministic value of the encrypted vote contents in order to calculate return codes.

Asymmetric Key Generation: As presented in Section 5, the eValg2011 solution relies on the following relation between the x_{vcs} private key of the VCS, the x_{rcg} private key of the RCG, and the x_e election private key $x_{rcg} - x_{vcs} \equiv x_e \pmod{p}$. This relationship is required for retrieving ciphertexts with deterministic properties from the encrypted votes.

The VCS and RCG keys are generated in two different, isolated environments to prevent both keys from being used to reconstruct the election private key. We will identify these environments as voting card generation (VCG) modules. During the key generation process, VCS and RCG private keys are split into shares (using a Shamir secret sharing scheme [Sh79]) that are distributed among the members of an electoral board. The shares are stored using PIN protected smartcards owned by the members. Since VCS and RCG keys are generated in two different environments, electoral board members participate in two different processes. Finally, each member will hold two shares, each one from a different private key (x_{rcg}, x_{vcs}). The election's private key is never generated, since it

can be reconstructed at the end of the election from the shares owned by the electoral board members. Only the public key is generated, using the public keys of the VCS and RCG. The private keys (x_{rcg}, x_{vcs}) are also uploaded to the corresponding servers VCS and RCG in a secure way (encrypted).

Symmetric Key Generation: The VCS and RCG also require symmetric secret keys for implementing the cryptographic operations to generate a deterministic value related to the encrypted vote contents (i.e., the return code sent to the voter). These keys (K_{vcs}, K_{RCG}) are generated using a secure random number generator. They are uploaded to the corresponding servers in a secure way (encrypted).

6.2 Generation of Voting Cards

According to the different return code representation options at the presentation level explained in Section 4, the voting cards may have different formats: they may have unique return code values for each option and/or return code values representing positions. For the sake of simplicity, we will explain how the voting cards are generated when position return codes are used to represent the candidates from party lists, and unique return codes are used to represent each party list. This is the most complete case of return code representation options. The municipal and county election used a simplified presentation with only unique return codes per party lists.

The voting card is a paper sheet containing a unique return code for each party list and for each position on the party list. Although the scheme supports return codes per candidate, candidates are represented by their position on the party list in order to make the voting card management and the return code comparison process (for voter verification) more usable for the voter. Certain Norwegian elections could have 25 parties with, in some cases, 99 candidates. If individual return codes per candidate are used, the amount of codes on the voting cards could be approximately 2,500 codes (25+(99*25)). Using position codes, the voting card will only need 124 codes (25+99). Other return code representation options were also implemented in the different elections and pilots carried out. All of them, as well as their risks and impact, are discussed in Section 8 of this paper.

Voting cards are used to verify that the voter's intent was properly recorded (cast-as–intended verification) by the ballot box located in the VCS. To this end, after the voter casts a vote, the RCG calculates (in collaboration with the VCS) and returns the return codes of the party and the candidate's position in this party for each selected candidate. Since these values are obtained from operations using the encrypted vote, voters can verify if their cast votes contain their selected voting options by comparing the return codes returned by the RCG with the ones available on the voting card for the same selected voting options. The fact that the voting card is only available on paper and is only known by the voter makes it impossible for a compromised component of the voting platform (the voting client, the RCG, etc.) to subvert the cast-as-intended verification method, by profiting from the knowledge of the return code values. This

could allow the component to change the voting options cast by the voter and send the return codes corresponding to the original selections. These return codes are sent to the voter through a different channel (SMS) than the one used for casting the votes.
An example of how this verification of parties and positions works is shown in the Figure 1.

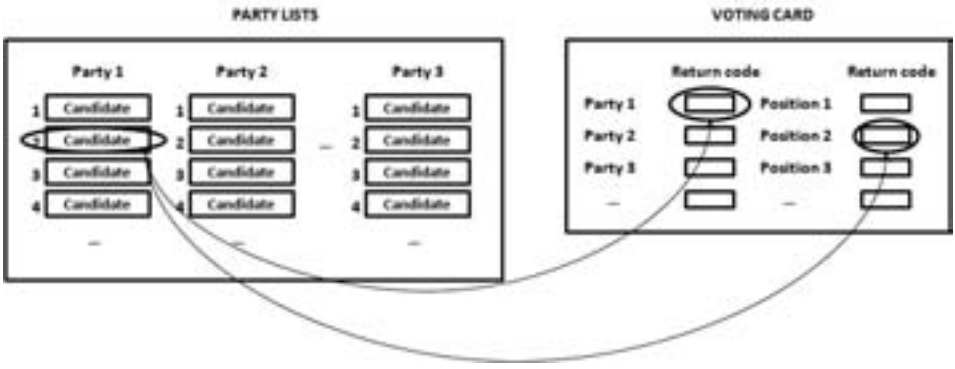

Fig. 1: Cast-as-intended verification with voting cards

Therefore, the voting card contains two sets of return codes:

- Party return codes
- Position return codes

Due to usability and SMS message length constraints, the party and position return codes sent via SMS are limited to 4 numerical characters (original codes obtained by the return code generator have 256 bits length, equivalent to 43 characters in base64 representation). This could generate collision issues between two different options if the original codes are only truncated (i.e., two different choices could end up with the same code on the voting card). Therefore, these SMS codes are generated in advance (controlling possible collisions) and mapped to the original codes in a secure way by combining a hash and encryption function. This mapping database is stored in the RCG during the election configuration phase.

A multi-party generation process is used to calculate the return code data during the election configuration phase. To this end, the two different and isolated VCG environments (VCG1 and VCG2 modules) are used to reproduce the same *deterministic* transformation of the votes that will be carried out by the VCS and RCG during the voting process: VCG1 implements the VCS transformation, and VCG2 implements the RCG transformation and links the result to the return codes that will be sent to the voter. This separation of duties prevents both VCG1 and VCG2 from correlating the generated return codes with the identity of the voters they belong to (VCG1 knows the voter identities; VCG2 knows the return code values). Therefore, an attacker controlling only one of these modules cannot influence the election results without being noticed.

The voting card and return code generation process done during the election configuration phase is divided into the following steps:

Calculation of Initial Candidate and Party (long) Codes: These are the codes obtained after applying the VCS and RCG cryptographic operations over each individual ciphertext that composes the encrypted vote cast by the voter (containing the code of the party list or candidate). This process is split between the VCG environments.

In a first step, the VCG1 generates random voter identifiers *voterID* and computes for each one a partial calculation of party and candidate codes as:
$s = AES_{K_{vcs}}(voterID')$, $P_i' = P_i^s \bmod p$, $C_i' = C_i^s \bmod p$

These partial calculations of party and candidate codes and related random voter identifiers are passed to the VCG2 module using an offline (air-gapped) channel.

Secondly, VCG2 calculates the final values of the party and candidate codes using the partial calculation from VCG1: P_i' and C_i', an HMAC function, a secret key K_{rcg}, and the random voter identifier *voterID*.
$PartyCode_i = HMAC(P_i' \,||\, voterID,\ K_{rcg}),\ CandCode_i = HMAC(C_i' \,||\, voterID,\ K_{rcg})$

Calculation of Party and Position (short) Return Codes: These are the short codes representing the parties and positions of candidates on party lists, which are printed in the voting cards. Since the SMS position and party return code values will be different for each voter, they are calculated by VCG2 follows:
$PartyReturnCode_i = HMAC(voterID \,||\, party_i,\ K_{rcg})$
$PosReturnCode_i = HMAC(voterID \,||\, position_i, K_{rcg})$,
where $party_i$ and $position_i$ are constant numeric values assigned to parties and positions.

Mapping Party and Candidate (long) Codes with Party and Position Return (short) Codes: VCG2 hashes each possible party or candidate code and stores it in the table connected to the party or position return code corresponding to it:
$H(PartyCode_i) \leftrightarrow PartyReturnCode_i$
$H(CandCode_i) \leftrightarrow PosReturnCode_i$

This table is randomized and finally deployed in the RCG, so that it is able to correlatevthe party and candidate codes with the party and position return codes during the voting process (without knowing the connection to the original party and candidate names). As we mentioned before, the return codes sent to the voter shall not be known by any component of the platform.
Otherwise, the voter selections could be changed and the attacker could send the return codes corresponding to the original vote to cheat the voter.

Therefore, in order to prevent the RCG from knowing the party and position return code values in advance, each return code is encrypted using the corresponding party or candidate code (which has to be generated in collaboration with the VCS from a valid vote) as a symmetric key ($AES_{PartyCodei}(PartyReturnCode_i)$ / $AES_{CandCodei}(PosReturnCode_i)$).

Printing and Assigning Voting Cards to Voters: Finally, party and position return codes and random voter identifiers (*voterID)* are given to the printing service for printing the voting cards. Once printed and in an envelope, each *voterID* is assigned to a valid voter identity and an envelope containing the voting card is sent to the voter address. The link between the *voterID* and the voter identity is kept on the electoral roll to allow the VCS to retrieve the correct *voterID* value during the voting process.

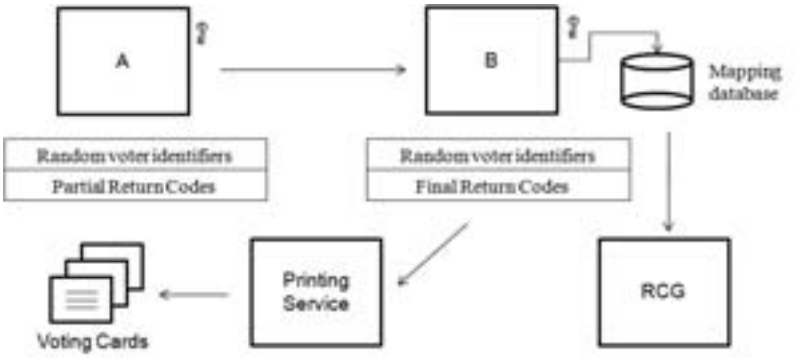

Fig. 2: Return code information generation during the configuration phase

A diagram of the return code information generated during the election configuration phase is shown in Figure 2 (modules A and B in the picture represent the VCG1 and VCG2 environments).

7 Voting Phase

As already mentioned, during the voting phase the return code generation process is split into two processes performed by two independent modules, the VCS and the RCG. This prevents a malicious single entity from cheating voters without being detected.

During this phase, the VCS executes a first set of cryptographic operations over the encrypted, cast vote, which are then forwarded to the RCG. The RCG executes a second set of operations to generate the final return code values. The VCS and RCG keys generated in the election configuration phase are used during the voting phase to perform such operations. In order to ensure that the calculations in the VCS are fair (e.g., to prevent the VCS from trying to make the RCG return codes from another vote or voter), several zero-knowledge proofs (ZKPs) are generated, relating the partial calculations from the VCS to a specific voter.

Once completed, the following steps are carried out:

Vote Encryption and Casting: The voting options chosen by the voter are individually encrypted using the election public key and sent to the VCS: $v_{opt_i} = (a_i, b_i) = (g^{r_i}, v_i \cdot h_e^{r_i})$

Vote Re-encryption and Partial Decryption: VCS applies some sort of re-encryption of the voting options using a voter-secret parameter *s*. This re-encryption is used to get a *personalized,* random encryption for each voter, which will be used to generate the return codes. The *s* parameter is calculated using the random voter identifier and the secret key K_{vcs} ($s = AES_{K_{vcs}}(voterID')$). The re-encryption consists of raising the encrypted voting options to this *s* value: $v_{opt_i}' = (a_i', b_i') = (a_i^s, b_i^s)$

After re-encrypting the voting options, the VCS performs a partial decryption of the result: $b_i'' = b_i' \cdot a_i'^{x_{vcs}}$

Finally, the VCS generates non-interactive ZKPs of the calculations made on the cast vote. These ZKPs allow the RCG to validate the correctness of such operations. The VCS generates two sets of ZKPs to prove the validity of the values:

- A proof that demonstrates that the VCS identified and used the correct voter secret parameter *s* to re-encrypt the vote (i.e., that is not using the parameter *s* of another voter)
- A proof demonstrating that the VCS identified and used its ElGamal private key x_{vcs} for partially decrypting the re-encrypted vote.

The encrypted vote (as originally cast by the voter) and the result of the VCS and ZKPs are sent to the RCG.

Vote Partial Decryption and Generation of Return Codes: The RCG verifies the ZKPs in order to ensure that the VCS calculations are correct and done over a specific vote. If they are correct, it partially decrypts the vote (already partially decrypted by the VCS) using its private key: $b_i'' \cdot a_i'^{-x_{rcg}} = b_i^s \cdot a_i^{s \cdot x_{vcs}} \cdot a_i^{s \cdot (-x_{rcg})} = (b_i \cdot a_i^{x_e})^s = v_i^s$

and retrieves the party and candidate codes related to the contents:
$\{Party/Cand\}Code_i = HMAC(v_i^s \,||\, voterID,\ K_{rcg})$

The RCG uses a hash of these codes to retrieve the related return codes from the database:
$H(PartyCode_i) : AES_{PartyCodei}(PartyReturnCode_i)$
$H(CandCode_i) : AES_{CandCodei}(PosReturnCode_i)$

In case the hash of the code is not found in the database, the RCG assumes that the vote which was cast does not contain a valid value. If so, an error is reported to the voter and the vote is rejected. This mechanism prevents the acceptance of votes containing invalid options. The return codes are formatted and sent to the mobile phone of the voter via an SMS gateway.

The process is shown in Figure 3:

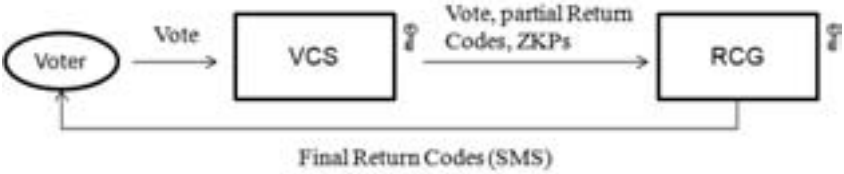

Fig. 3: Return code generation during the voting phase

8 SMS Format

As mentioned before, using SMS messages introduces length and usability constraints for verifying the vote. This has a direct impact in the soundness of the verification process. The format and contents of the SMS messages have been reviewed and tested in several pilots to achieve a good balance between usability and verifiability soundness. Two different SMS formats based on the different return code representation options given at the presentation level, are used: position return codes and the number of candidate selections. In both cases, party return codes are always reported.

SMS with Candidate Position Return Codes: In this case, the message initially contains the party return code, followed by the return codes of the selected candidate's position (if any) for that party.

Figure 4 shows a sample SMS sent to a voter:

You cast a vote for party $PartyReturnCode_1$ candidate positions $PosReturnCode_1$, $PosReturnCode_{2...}$

Fig. 4: SMS format with Position Return codes

SMS with the Number of Candidate Selections: This is the approach used in the last election carried out using this voting platform. In this case, the SMS message only contains the party return code value and a text mentioning the number of selected candidates for that particular party (if any).

Figure 5 shows a sample SMS sent to a voter.

You selected 3 candidates from party $PartyReturnCode_1$

Fig. 5: SMS format with position Return codes

Soundness of the Verification Process: Different return code representation options have a significant impact on the soundness of cast-as-intended verification. When only the party return code and the number of candidates selected are sent to the voter, she cannot verify if the candidate options registered in the voting platform are actually what she selected. She only knows that the number of selections is correct but not if the actual candidate from the party was cast as intended. When party and position return codes are used, the verification process could be subverted if the candidates are shown to the voter in a different position than the official one. Therefore, it is clear that there is a significant tradeoff between usability and soundness. The government's decision to use these representations was made after evaluating these risks and finding out that they did not apply to voters who only select party lists (about 98% of the voters).

9 Final Remarks

One of the aspects highlighted by this paper is how usability influenced several implementation details of the proposal. Initially, usability influenced the decision of to re-design the original cryptographic scheme. This made the system less dependent on the resources available from the voter's computer (enhancing the response time of the voting process). Usability aspects were also of paramount importance for designing the format and contents of the voting cards and SMS messages. In this case, the verification soundness was reduced to achieve a better voter understanding of the verification process (e.g., reporting the number of candidates selected in a party instead of which candidates or candidate positions). Finally, the cast-as-intended method described here protects the integrity of the vote from malicious software installed in the voting terminal. However, it does not protect the voter from other malware attacks, such as capturing voter credentials. This could be considered a serious risk in Norway since voters are allowed to cast multiple ballots. However, the current use of an authentication method based on digital certificates and one-time passwords mitigates this type of attack.

Bibliography

[Ad08] Adida, B. 2008. "Helios: web-based open-audit voting", in Proceedings of the 17th Conference on Security Symposium (San Jose, CA, July 28 - August 01, 2008). USENIX Association, Berkeley, CA, 335-348.

[CCE11] Chaum D., Clark J, Essex A, Rivest R, Sherman A, Vora P., Zagorski F. "Remotegrity". Crypto 2011 rump session. August 16, 2011.

[Ce02] CESG (Communications and Electronic Security Group). "E-voting security study", annex C. 2002. Available at: http://www.edemocracy.gov.uk/library/papers/study.pdf 2002

[Ch01] Chaum D. "SureVote: Technical Overview", Proceedings of the Workshop on Trustworthy Elections (WOTE '01), presentation slides, August 2001.

[El84] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In Proc. of CRYPTO 84, pp 10-18.

[EV09] e-Vote 2011 Security Objectives. 2009. Online: http://www.regjeringen.no/upload/KRD/Kampanjer/valgportal/e-valg/tekniskdok/Security_Objectives_v2.pdf

[FP01] FIPS PUBS 197: Advanced Encryption Standard (AES). 2001. Online: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[Gj10] Gjøsteen, K. 2010. "Analysis of an internet voting protocol". Cryptology ePrint Archive, Report 2010/380. Online: http://eprint.iacr.org/2010/380

[HS07] Helbach, J., Schwenk, J. "Secure Internet Voting with Code Sheets". E-Voting and Identity, First International Conference, VOTE-ID 2007, Bochum, Germany, October 4-5, 2007, Revised Selected Papers. Springer 2007, ISBN 978-3-540-77492-1, pp.166-177.

[Li11] Lipmaa, H. "Two Simple Code-Verification Voting Protocols". IACR Cryptology ePrint Archive 2011: 317 (2011).

[MMP02] Malkhi D. Margo O., and Pavlov E. "E-voting without 'cryptography'". In Matt Blaze, editor, Financial Cryptography, 6th International Conference, FC 2002, Revised Papers, volume 2357 of Lecture Notes in Computer Science, pages 1–15, Southampton, Bermuda, 2003. International Financial Cryptography Association, Springer.

[MOV96] A. Menezes, P. van Oorschot, and S. Vanstone. "Handbook of Applied Cryptography" CRC Press, 1996.

[MSP09] Morales-Rocha, V., Soriano, M. and Puiggalí, J. "New voter verification scheme using pre-encrypted ballots". Comput. Commun. 32, 7-10 (May 2009), 1219-1227.

[PG11] Puiggalí, J. Guasch, S. "Internet Voting System with Cast-as-intended Verification". VoteID 2011. Tallinn, Estonia, September 2011.

[Sc91] Schnorr C. "Efficient Signature Generation by Smart Cards". Journal of Cryptology vol. 4 (3) 1991.

[Sh79] Shamir A. 1979. How to share a secret. Commun. ACM 22, 11 (November 1979), 612-613.

[St07] Storer, T. "Practical Pollsterless Remote Electronic Voting". Thesis, 2007.

[SVK11] O. Spycher, M. Volkamer, R. Koenig. "Transparency and Technical Measures to Establish Trust in Norwegian Internet Voting". VoteID'11, 3rd International Conference on E-Voting and Identity. Tallin, Estonia, 2011.

[VZ05] Voutsis, N., Zimmermann, F. "Anonymous code lists for secure electronic voting over insecure mobile channel"s. Proceedings of Euro mGov 2005, Sussex University, Brighton, U.K., 10-12 July 2005.