Internet Voting System with Cast as Intended Verification

Jordi Puiggalí Allepuz and Sandra Guasch Castelló

Scytl Secure Electronic Voting

Abstract. In remote electronic elections the voting client software is usually in charge of encoding the voting options chosen by the voter. Cast as intended verification methods can be used to audit this process, so that voters do not need to trust the voting client software. In this paper we present the revision of our initial proposal for the eValg2011 project for an Internet voting protocol providing cast as intended verification functionalities, and evaluate its security.

1 Introduction

In remote electronic elections, the voting client software is generally in charge of encoding the voting options chosen by the voter before sending the encrypted vote to a remote voting server. Most of the times, that means that voters have to trust that the voting client is not going to change their selections before they are encrypted. In case the voting client would do it, the probability of being detected is very low. For this purpose, cast as intended verification methods have been designed: voters do not need to trust the voting client software to properly encode the selected voting options, since they can audit the process.

As stated in [1], the purpose of the eValg2011 project is to establish a secure electronic voting solution for general, municipal and county council elections. For this purpose, a set of Objectives of Security [2] for the eVoting system to be implemented for the eValg2011 project was defined for the bidding phase. Specifically, the ability of detecting potential vote manipulations by a malicious voting client when casting a vote was required (zero trust in the voting client).

The aim of this paper is the presentation of our initial proposal for the eValg2011 project for an Internet voting protocol providing cast as intended verification functionalities. The current eValg2011 eVoting system is a modification of this proposal in order to perform some cryptographic operations in the voting server instead of in the voting client. It is not an objective of this paper to compare both proposals but to introduce a revised version of the original proposal. A description of the eValg2011 system currently implemented can be found in [8].

Specifically, we are going to describe the processes of the original proposal that provide cast as intended verifiability of the voting process and evaluate the security of this proposal.

A. Kiayias and H. Lipmaa (Eds.): VoteID 2011, LNCS 7187, pp. 36–52, 2012.

2 Previous Work

The verification done by a voter to check that her vote has been registered correctly in the election platform server is a critical task. This verification must be done in such a way that the voters obtain a verification proof that unequivocally prove if their voting intent has been properly recorded and, at the same time, cannot be used latter to correlate the vote with the voter (in order to prevent vote buying). Another consideration is that the proofs must be verifiable by human means or with assisted means that do not pose any usability and privacy issue. For instance, verification processes that require voters to perform mathematical operations should consider the human limitations.

Several proposals for cast as intended verification in remote electronic voting schemes have been proposed in the literature. In code voting methods, also known as Pollsterless or pre-encrypted ballots [18], [15], [13], [5], [6], [19], [9], [12], the voter receives in advance a voting card with voting codes and validation codes related to the voting options eligible in the election. In order to vote, the voter enters the codes representing her choices in the voting client, which sends them to the voting server. The voting server then replies to the voter with validation codes that are calculated from the received voting codes. The voter can compare them with those assigned to the selected voting options in her voting card, so that she can verify if the encrypted voting options received in the voting server represent her voting intent.

There are other alternative proposals to code voting that do not require the use of voting and validation codes for providing cast as intended verification. This makes the election configuration easier and reduces logistic costs. For instance, in the method proposed in [4], the voter challenges the voting application in order to verify that the encrypted values match her selections: before the vote is cast, the voting application commits to the generated encryption (e.g., showing the digest value of the cipher text) and the voter can opt to challenge the voting application to disclose the contents of the encrypted vote. If the voter requests the verification, the random factor used to encrypt the vote is revealed by this application, so that the voter can verify that the cipher text (according to the commitment generated by the voting application) corresponds to the chosen voting options. After that, the disclosed encrypted vote is dropped and a new encryption with the same selected options is generated again with new random values (in order to prevent vote selling practices). If the voter does not want to verify her encrypted vote, this is sent to the voting service. Since the voting client does not know when the voter will ask for verifying her vote, it is difficult to cheat the voter when encrypting the voting options without being noticed.

When considering which method could be the basis of our proposal, the approach of challenging the voting client was discarded due to usability and independent verification issues: the operations for checking the proper encryption of a vote cannot be done by human means. Since a validation mechanism provided by the voting application could be also modified by the malicious client, the voter needs an external tool or the collaboration of a third party to verify the correct encryption of her vote. Even the election talliers could decrypt the

votes to be audited (at the end of the election) in order to let the voters verify the decrypted contents match their voting intent. However, the process can be complex and difficult to understand for an average voter.

Regarding code voting methods, the approach of comparing the received codes after casting the vote with the validation codes present in a voting card, seems more familiar for the voters (e.g., it is similar to looking if you have won the lottery) and can be done without any support or tool. The drawback, compared with the challenge method, is that the voter needs to trust that the voting cards have been generated properly and their information is not stored anywhere. However, security controls can be implemented to verify that voting cards are correctly generated and there is no disclosure of information.

Furthermore, there are still some usability and accessibility issues remaining in the code voting methods related to the voting process: these methods require voters to enter codes representing the candidates for casting a vote, which could lead to mistakes and is less usable than a click and select interface (which is feasible in challenge methods).

To overcome these issues, our proposal for cast as intended verification introduced a new variant of code voting that does not use codes for casting votes, but keeps the validation codes for allowing voter verification. The novelty of this scheme is that, instead of using voting codes, it uses a click and select scheme for vote casting (in order to make the system more usable) combined with validation codes for voter verifiability.

3 Proposed Solution

3.1 Design Requirements

The main challenge when designing a code voting solution that only uses validation codes, is how to combine probabilistic and deterministic encryption schemes: votes shall be encrypted using a probabilistic encryption scheme in order to maintain voter privacy. However, validation codes are calculated from the encrypted votes and their values have to be always the same (i.e., they should be deterministic) in order to generate the voting cards in advance. Finally, the system shall provide a high level of usability.

3.2 Overview of the Proposed Solution

Voters use validation codes (called return codes in our scheme) for verifying the proper recording of their vote contents in the voting server. To this end, before the voting phase, the voters are equipped with voting cards containing the return codes assigned to each voting option. Voting cards are not linked to a specific voter, so that they may be exchanged between voters or a voter can request more than one card in case she thinks her first card may have been intercepted by an attacker.

The generation of such voting cards is done during the election configuration phase. Some secret keys are involved in the generation of the return codes of

the voting cards, and these keys will be required by the voting platform for generating the return codes from the votes cast. Therefore, during the election configuration stage, these keys are generated, used in the return code generation process and installed in the voting platform. It is assumed that a secure method for protecting the keys is used.

The eVoting platform server side is composed by two main modules: the voting server (Vote Collector Server or VCS), which contains the Ballot Box storing the votes to be counted at the end of the election and participates in the generation of the return codes for voter validation, and the validation server (Return Code Generator or RCG), which generates the final values of the return codes and sends them to the voters.

When the voting phase starts, voters connect to the voting platform, authenticate themselves and select their preferences. After the voter confirms her choices, the voting client encrypts the vote using the election public key.

The voter proceeds to the audit phase in order to verify that her vote is cast as intended. In this phase, she is asked to introduce a special code present in the voting card: the voting card identifier *VCID*. A second encryption is then generated using this value and the RCG public key. The first encryption is the vote which is received in the VCS, stored in the Ballot Box and counted in the tallying process. The second encryption is used by the RCG for generating the return codes associated to the vote contents.

In addition, the voting client generates a cryptographic proof correlating the contents of both encryptions. This prevents a malicious voting client from generating two encryptions based on different voting options in order to cheat the voter (e.g., the encrypted vote over which the return codes are generated contains the selections made by the voter, but the encrypted vote stored in the Ballot Box contains different selections).

Both encryptions and the proof are sent to the remote voting server VCS, which forwards the contents to the RCG.

The RCG verifies the proof and uses its cryptographic keys to generate the return code from the second cipher text. The return code is sent back to the voter using a channel independent from the one used to cast the vote (e.g., SMS).

After that, the RCG notifies the VCS about the successful process sending back a digitally signed hash of the encrypted vote (called the voting receipt). The VCS then stores the first cipher text in the Ballot Box and forwards the voting receipt to the voting client, which shows it to the voter. This receipt can be used as a proof that the vote has been *recorded as cast*: as the VCS publishes the voting receipts corresponding to the contents in the Ballot Box, voters can verify that their votes are correctly stored.

The voter receives the return code and compares it with the code assigned to the selected voting option in her voting card in order to check the correctness of the encrypted vote.

It is assumed that multiple voting is allowed (the system supports it), so that in case wrong return codes are generated the voter can opt to cast her vote from another computer.

If multiple voting is not allowed, the verification of correct encoding of the vote could be done before the casting process: the second encryption is generated in first place and sent to the RCG, which generates the return code and sends it to the voter. In case the verification is successful, the voter can decide to cast the vote (the first encryption), so that it is stored in the Ballot Box.

The ability of asking multiple times for return codes, by multiple voting or by this second system, prevents vote selling practices based on the values of the received return codes.

4 Detailed Cryptographic Protocol

The participants of the voting process are the voter V, the voting client C, the voting server VCS containing the Ballot Box and the validation server RCG, which generates the return codes to be sent to the voter.

The encryption algorithm is ElGamal [7]. During the election configuration phase, the election cryptosystem parameters p, q, g are defined:

- The modulo p is chosen as a large safe prime, where $p = 2q + 1$ and q is a prime number.
- g is a generator of G_q , a q-order subgroup of Z_p^* .

As it has been explained in the overview section, the two encryptions of the vote are calculated under different keys: the election and the RCG public keys.

The election and RCG private keys x_e and x_{rcg} are independently selected from Z_q, and the public keys h_e and h_{rcg} are calculated as $h_e \equiv g^{x_e} \bmod p$, $h_{rcg} \equiv g^{x_{rcg}} \bmod p$ (modular operations will be obviated from now).

Symmetric keys K_{vcs} and K_{rcg} for the VCS and the RCG respectively are also generated in the configuration phase. The K_{vcs} key will be used to generate the second encryption of the vote, while K_{rcg} will be used to generate the return code values.

The following sections describe the steps of the voting process.

4.1 Vote Preparation for Vote Casting

After the voter confirms her selections, the voting client C proceeds to encrypt the vote using the ElGamal encryption algorithm and the election public key.

The vote is encrypted using a random exponent r in Z_q as:

$$c_{prob} = (v \cdot h_e^r, g^r) = (\alpha, \beta) \tag{1}$$

A proof of knowledge $proof_1$ of the random exponent using the Schnorr Identification Protocol [11] is generated for the encrypted vote in order to prevent reply attacks based on re-using a valid vote from a voter to discern the voter intent based on the return codes received by the attacker.

4.2 Vote Preparation for Verification

A second deterministic encryption using a fixed exponent (in order to be able to generate the return codes) and a cryptographic proof relating both encryptions are generated in the voting client.

This second encryption is based on generating a cipher text using the ElGamal encryption algorithm as a deterministic algorithm by using a fixed exponentgenerated from information only known by the voter and by the VCS, so that the RCG cannot learn about the voting options selected by the voter when generating the return codes. It is generated in such a way that the resulting cipher text is different for each voter and voting option in order to calculate suitable values for the return codes from it. The cipher text is then re-encrypted using the RCG public key h_{rcg} and the ElGamal encryption algorithm with a random exponent, so that VCS never learns about the selected voting options.

Generation of a Fixed Exponent in the Voting Client. A first fixed exponent is generated in the voting client using information only known by the voter (the voting card identifier *VCID* and the vote content):

$a = H(VCID, v)$, where H is a pseudorandom function[1].

Request of a Second Fixed Exponent to the VCS. The voter enters the *VCID* code identifying her voting card into the voting client application, which is used to request a value for a fixed exponent to the voting server VCS without revealing it. The VCS uses its secret key K_{vcs} to generate that value:

$C \rightarrow VCS : b = H(VCID)$
$VCS \rightarrow C : d = H(K_{vcs}, b)$,

A blind signature mechanism could be used in order prevent the VCS from knowing the value d.

For example, the VCS could have a keypair t_{vcs} (public), s_{vcs} (private) from an RSA scheme. The Voting Client could then send the value b to the VCS, blinded by a random value r: $b' = r^{t_{vcs}} \cdot b$. The VCS would then answer with $b'' = b'^{s_{vcs}}$, and the Voting Client could recover the exponent $d' = b^{s_{vcs}}$ as $d' = b''/r$.

Encryption with Fixed Exponents Generated by the Voting Client and the VCS. The deterministic encryption of the voting options is generated as:

[1] Since a is part of the fixed exponent used to generate the deterministic encryption of the vote, the function H might be hard to compute in order to prevent effective brute-force attacks to disclose the voter intent. For example, H could be a Password Based Key Derivation Function ([3], [10]), which can be defined as a pseudorandom function slow to compute, and it is usually used to derive cryptographic keys from non-strong passwords. The number of bits generated by the PBKDF shall be defined in such a way that this computation is not too slow to be executed by the voting client while maintaining the robustness of the encryption.

$$c_{det} = (v \cdot h_e^{a+d}, g^{a+d}) = (\alpha', \beta') \tag{2}$$

Generation of Proof of Equality of Plaintexts. The proof ($proof_2$) is used to demonstrate to the RCG that both encryptions, the one which will be stored in the Ballot Box and the one used to generate the return codes, contain the same plaintext (selected voting options). This way, it is ensured that the information which is audited (the contents over which the return codes are calculated) is the same stored in the Ballot Box.

This proof is a Non-Interactive Zero Knowledge Proof based in the Schnorr Signature protocol [17], which is used to prove that both encryptions (α, β), (α', β') contain the same plaintext (see [14] with corrections in [16]).

In order to do that, both cipher texts are divided and the voting client proves knowledge of the equivalent encryption exponent $(a + d)/r$.

$$(\alpha'/\alpha, \beta'/\beta) = (h_e^{(a+d)/r}, g^{(a+d)/r}) \tag{3}$$

Re-encryption Using the RCG Public Key. The cipher text obtained from the encryption of the vote with fixed exponents is re-encrypted using the ElGamal encryption algorithm, the RCG public key h_{rcg} and the same random exponent r used for the first encryption. This re-encryption prevents VCS from being able to perform a brute force attack on the value $v \cdot h_e^a$. The RCG will need to remove this re-encryption layer in order to recover α', which is needed to verify $proof_2$. The value β' is re-encrypted with a different random exponent in order to prevent an observer to infer if the voter has chosen twice the same voting option or not.

$$\alpha'' = \alpha' \cdot h_{rcg}^r \tag{4}$$

$$\beta'' = \beta' \cdot g^{r'} \tag{5}$$

$$c'_{det} = (v \cdot h_e^{a+d} \cdot h_{rcg}^r, g^{a+d} \cdot h_{rcg}^{r'}, g^{r'}) = (\alpha'', \beta'', \gamma) \tag{6}$$

4.3 Vote Casting

Both encryptions and the proofs are digitally signed using the voter credentials and sent to the VCS.

$C \rightarrow VCS : \mathrm{Sig}((\alpha, \beta), (\alpha'', \beta'', \gamma), proof_1, proof_2; S_{voter})$

The $proof_1$ and the voter digital signature are verified at reception in the VCS. In case the verification is successful, both encryptions and the proofs are forwarded to the RCG.

$VCS \rightarrow RCG : \mathrm{Sig}((\alpha, \beta), (\alpha'', \beta', \gamma), proof_1, proof_2; S_{voter})$

4.4 Generation of Return Code

The RCG verifies the $proof_1$ and the voter digital signature at reception. If the verification is successful, it performs the following steps in order to generate the return code from the cipher text $(\alpha'', \beta'', \gamma)$.

Decryption with RCG Private Key. The RCG uses its ElGamal private key to decrypt the second cipher text in order to obtain the original deterministic encryption with the fixed exponents generated in the voting client (see Eq. 2):

$$(\alpha', \beta') = (\alpha'' \cdot (\beta^{-x_{rcg}}), \beta'' \cdot (\gamma^{-x_{rcg}})) = (v \cdot h_e^{a+d}, g^{a+d}) \tag{7}$$

Verification of Proof of Equality of Plaintexts. Once the RCG has obtained the cipher text (α', β'), it can verify that this cipher text is a re-encryption of (α, β)using $proof_2$ and following the steps described in [14] and [16].

Calculation of Return Code. Finally, the RCG generates the return code value using its symmetric key as:

$$\text{RC} = H(\alpha'; K_{rcg}) \tag{8}$$

4.5 Delivering Return Code and Voting Receipt

The return code is sent to the voter using an alternative channel to the one used to cast the vote (e.g. SMS). After sending the return code to the voter, a confirmation is sent to the VCS in order to store the encrypted vote in the Ballot Box. This confirmation is the signed hash of the encrypted vote to be stored in the Ballot Box (the voting receipt). Once the VCS has stored the vote, it forwards the voting receipt to the voting client as a confirmation of the recording of the vote. The VCS publishes the voting receipt (hash of the stored vote) in a Bulletin Board.

$RCG \rightarrow V : RC$
$RCG \rightarrow VCS : \text{Sig}(H(\alpha, \beta); S_{rcg})$
$VCS : \text{Publish Sig}(H(\alpha, \beta); S_{rcg})$

In order to prevent some attacks hidden as transaction problems (e.g., a malicious VCS claims not having received the voting receipt from the RCG, so that the vote is not stored, but the RCG claims having sent that receipt to the VCS), both the VCS and the RCG store the whole set of information generated during the voting phase: the VCS stores the set of information cast by the voter (digitally signed by the voter) and the voting receipt (digitally signed by the RCG). The RCG stores the information cast by the voter and forwarded by the VCS (digitally signed by both the voter and the VCS), the vote encryption with fixed exponent over which the return code is generated (Eq. 2), and the voting receipt.

4.6 Validation of Return Code and Voting Receipt

The voter receives the return code and compares it with the code assigned to the selected voting option in her voting card in order to check the correctness of the encrypted vote.

The voter also receives the voting receipt digitally signed by the RCG, so that she can verify that her vote has been stored in the Ballot Box by comparing this value with the list published by the VCS in the Bulletin Board.

This explanation abstracts a vote as a container of one selected voting option. In case of multiple selections they are independently encrypted and the same process is repeated for each one, so that independent return codes are obtained.

All the information sent from one component to another is digitally signed in order to provide integrity and proof of origin (e.g., it is assumed that encrypted votes are digitally signed by voters).

5 Generation of Voting Cards

Both the voting card identifier *VCID* and the VCS symmetric key K_{vcs} are very sensitive, since they together can be used to guess the voting options selected by the voter during the voting phase.

Therefore, it is recommended to generate them in two isolated and independent environments. Each environment can be identified as a Voting Card Generation (VCG) module, VCG1 and VCG2. The same environments can then be used to implement a multiparty generation process to calculate the return codes for the voting cards: one environment for generating partial values, and another for generating the final ones.

The generation of partial and final return code values follows similar steps to those defined in the voting protocol. The main difference is that the starting point is not a cipher text, but a cleartext voting option.

Voting Card generation in VCG1

- A random voting card identifier *VCID* is generated for each set of return codes belonging to a voting card. The length of this *VCID* must consider both security and usability requirements.
- A fixed exponent is calculated for each voting option using the ballot identifier and a pseudorandom function: $a = H(VCID, v)$.
- A partial return code for each voting option is calculated following the equation:

$$rc' = v \cdot h_e^a \tag{9}$$

- The set of partial return codes generated for each voting card, related to the voting card code *VCC*, where $VCC = H(VCID)$, is passed to the following module.

Voting Card generation in VCG2

- A second fixed exponent is generated using the VCC value and the VCS symmetric key K_{vcs} as $d = H(K_{vcs}, VCC)$ (see 4.2). In case a blind signature scheme is used, d may be generated as $d = VCC^{s_{vcs}}$.
- The final return code value for each voting option is calculated in two steps, using the fixed exponent and then the RCG symmetric key K_{rcg}:

$$rc'' = rc' \cdot h_e^d, \tag{10}$$

$$\mathrm{RC} = H(rc''; K_{rcg}) \tag{11}$$

Two different environments are used to generate the return code values for the voting cards, in such a way that a single entity has no knowledge of the return codes values assigned to a specific voting card. It is assumed that the service in charge of printing the voting cards processes this information in a separate way in order to preserve the secrecy of the voting cards.

6 Security Analysis

The proposal is designed to provide cast as intended and recorded as cast properties, without compromising voter privacy. However, it is important to analyze the robustness of the proposed method against any possible compromise of its components and under which assumptions these components are expected to be deployed to reduce the risks. We are organizing this analysis based on the different stages of the election process and which components could be compromised. For each possible component compromise, the impact, difficulty of implementing an attack and the probability of success are briefly analyzed. It is not the aim of this section to make a strict threat assessment but to provide an initial analysis of the risks.

6.1 Election Configuration

During the election configuration phase the return codes and cryptographic keys needed during the protocol execution are generated. The disclosure of any of these components could potentially compromise the security properties of the proposal.

a) Return codes disclosure: If the return code values are intercepted by a third party at this stage they could be potentially used to compromise voter privacy or to cheat the voter without being noticed, changing the vote to be cast and sending the return codes of the original one. However, these potential attacks cannot be implemented without compromising other components of the system.

Voter privacy only can be compromised if the attacker also intercepts the return codes of the selected voting options when the vote is cast (i.e., intercepts the communications in the alternative channel or gains access to the RCG). Furthermore, voting cards are not linked to voters, can be exchanged between voters, and one return code value is probable to be connected to different options in different voting cards. Therefore, an attacker also needs to gain access to the voting card identifier to make the attack effective. Nevertheless, if multiple voting is allowed, voters can cast a vote latter on using a different voting machine and a different voting card. Therefore, the attacker needs to intercept all these communications as well.

For cheating the voter, the malicious voting client needs to send the manipulated vote to the VCS and the real selections made by the voter to the RCG. That way, the RCG could send the return codes of the options selected by the voter instead of the options contained in the vote. Otherwise, the voter could detect the attack when receiving the SMS messages with return codes of the manipulated vote. The protocol does not require any direct connection between the client and the RCG and therefore, this should be explicitly done without being detected (i.e., requires also control of the RCG server infrastructure). The success of the attack will depend also on the ability to the voter for casting another vote in another machine. If so, the attacker needs also to control any machine that could be used by the voter. The scalability of this attack depends also in the number of machines controlled, the number of voting cards stolen and the ability to control the RCG and hide unexpected connections from Internet to this machine (initially it should not accept any connection from Internet).

As a general assumption, it is expected that the return code generation process is implemented using the two-step process mentioned in Section 5, using an isolated environment under the supervision of auditors. The files with the return codes to be printed should be exported in two different files encrypted using two private keys, so that two separate services process them independently. Voting cards will be printed in tamper evident sealed blind envelopes (e.g., PIN envelopes). The storage devices used in this process are destroyed or completely wiped as soon as they are not longer required. This prevents accidental disclosure of information that could potentially be used in other attacks. At this stage of the voting process, the main risk is compromising the whole set of voting cards, since this could allow a large scale attack. Compromising only one or a small set makes attacks more difficult to target to a specific voter or reduces their impact.

b) VCS K_{vcs} symmetric key disclosure: in case this key is compromised, the attacker could try a brute force attack focused to disclose the voter intent from the deterministic encrypted vote (see Eq. 9), if it also colludes with the RCG. If so, the hardness of the attack will depend on the length of the voting card identifier *VCID* (e.g., for a 16 char base32 identifier, it will be 80bits). The use of a slow function as a PBKDF to generate the fixed exponent in Eq. 9 would increase the time needed to disclose the vote contents with such a brute force attack.

As a general assumption, this K_{vcs} key is generated using a cryptographically secure PRNG in an isolated environment (e.g., HSM) and it has a length of at least 128 bits. The length of the voting card identifier *VCID* should be of at least 80 bits and it should be generated using also a cryptographically secure PRNG.

c) VCS Signature key disclosure: assuming that the RCG also stores the vote, if this key is compromised, an attacker colluding with the RCG could store in this machine a vote that has not been stored in the VCS but with a proof that says it should (the digital signature of a vote stored in the RCG using the VCS private key). The main motivation of this attack is to disrupt the election and try to discredit the VCS, since privacy of the voter is not compromised. This attack also requires having access to a vote cast by a valid voter but that has not been stored in the VCS (since the voter has to be digitally signed using voter credentials). The attack could be detected if the contents stored in the RCG are crosschecked against the voting receipts published by the VCS in the Bulletin Board.

As a general assumption, this key is generated using a cryptographically secure PRNG in an isolated environment (e.g., HSM) and it has a length of at least 2048 bits. It is also expected that the contents of the RCG and the Bulletin board are checked periodically to detect discrepancies.

d) RCG x_{rcg} ElGamal private key disclosure: if this key is compromised, the attacker can decrypt the second encryption cast by the voter (see Eq. 2) in order to recover the vote encrypted by the fixed exponents. However, this information does not disclose enough details to discern the voter intent. Only colluding with the VCS the attacker could try a brute force attack over the voting card identifier *VCID*.

As a general assumption, this key is generated using a cryptographically secure PRNG in an isolated environment (e.g., HSM) and it has a length of at least 2048 bits. The length of the voting card identifier should be at least of 80 bits and it should be generated using also a cryptographically secure PRNG.

e) RCG Signature key disclosure: if this key is compromised and the attacker is colluding with the VCS, she could sign the voting receipt of a vote in order to provide confirmation to the voting applet without forwarding the vote the RCG. In this case, if the vote is not stored in the Ballot Box, this could be detected, since the voting receipt is not published in the Bulletin Board. Publishing the voting receipt without storing the vote can be detected crosschecking both repositories. In any case, the voter could detect this attack since she does not receive the return codes of the cast vote.

As a general assumption, this key is generated using a cryptographically secure PRNG in an isolated environment (e.g., HSM) and it has a length of at least 2048 bits. It is also expected that the contents of the Ballot Box and the bulletin board are checked periodically to detect discrepancies.

f) Election x_e ElGamal private key disclosure: if this key is compromised and attacker could decrypt any vote and compromise voter privacy.

As a general assumption, this key is generated using a cryptographically secure PRNG in an isolated environment and split in shares using a secure multiparty computation protocol with a pre-defined threshold. Shares shall be distributed among members of an Electoral Board with divergent interest, using a HSM (smartcards). The system shall not store any copy of the shares or the whole election private key. All the storage devices (except the smartcards of the electoral board members) shall be wiped or destroyed. The key shall have a length of at least 2048 bits.

6.2 Voting Process

During the voting phase, attacks can be targeted to the voters that are participating (or not) in the election, and against the votes cast. Assuming that attackers did not succeed on compromising the components in the configuration phase, we will consider in this stage risks in case of component compromise during the voting phase. Risks based on a previous compromise are the same as described before. In this phase, we will consider the risks more related to a system component compromise (Voting Client, VCS and RCG) rather than a cryptographic component compromise (keys). The only exception is the set of return codes (voting cards).

a) Return codes disclosure: In this phase, compromising a meaningful number of voting cards is more complex since they are already in possession of voters and the information of the configuration phase has been destroyed (i.e., it will require physical access to each voter location). Therefore, the attacks can be mainly focused on coercion or vote buying practices. Since voting cards are not linked to voters, if voters are allowed to cast more than one vote using different voting cards, coercers or vote buyers cannot use the voting cards to be sure of the voter intent even though they keep these voting cards.

As a general assumption, it is expected that the return code generation process destroys all the information of the return codes after voting cards are printed. It is also an assumption that voters are allowed to vote multiple times. Since the vote casting and vote verification processes can be executed in different order, in case the voter only can cast one vote, the processes could be reversed, sending first the deterministic encrypted vote to check the return codes before casting the vote. This is an option whose impact we are still evaluating in case the voter does not cast the vote, makes changes in the selection and requests again the return codes (sending another deterministic vote).

b) Voting client compromise: If the voting client is compromised (virus, malware, etc.), any attempt to make changes in the voter intent will be detected by the voter when checking the return codes. RCG can also detect any attempt of using a deterministic encryption with a different content than the vote (by

means of a cryptographic proof, see Section 4.2). Only collusion between the voting client and the RCG could pose some risk, as described in the section a) of the Risks during the Election configuration process. However, this will require also compromise other components of the infrastructure to hide any direct dialog between the voting client and the RCG.

As a general assumption, it is expected that the RCG is completely isolated from the public network (i.e., Internet) and connections restricted to those coming from the VCS. Voting client is expected to be digitally signed to prevent the use of manipulated or non-validated client software.

c) VCS compromise: If the VCS is compromised, it cannot gain access to the voter intent (it cannot decrypt the information) or manipulate the vote, since it is assumed that is digitally signed by the voter. It could disrupt the election by not storing the votes but publishing the voting receipts (to prevent voters to detect that their votes were not recorded as cast). However, this malicious practice could be detected by periodically cross-checking the Ballot Box contents with the Bulletin Board ones. If the RCG is also storing a copy of the valid votes from which it has generated return codes, votes eliminated by the VCS could be recovered. Only collusion between VCS and RCG could prevent the recovery of the vote. Attacks described in section b) and c) also apply in this case.

As a general assumption, it is expected that the keys of the VCS are stored in a cryptographic device that prevents them from being exported (HSM). Periodical checks of the Ballot Box and Bulletin Board contents are also expected.

d) RCG compromise: If the RCG is compromised, it can send return codes to voters without sending the voting receipt to the VCS, so that the votes are not stored in the Ballot Box. However, this will be detected by voters when a timeout is reached in their voting client since the receipt is not received. In the other hand, trying to guess the selected voting options after decrypting the deterministic vote (see Eq. 7) is not feasible. Only with the collaboration of the VCS, that could recover its part of the fixed exponent, could reduce the complexity of the brute force attack to the length of the voting card identifier value (usually 80 bits). The other attacks described in sections d) and e) of the Election configuration phase also apply in this case.

As a general assumption, it is expected that the keys of the RCG are stored in a cryptographic device that prevents them from being exported (HSM). Periodical checks of the RCG stored information (validated votes and voting receipts) and Bulletin Board contents are also expected.

6.3 Post Election

Once the election is done, the main risks are related to threat the data that has been registered during the election. Mainly, the following datasets will be vulnerable to attacks: the votes encrypted under the election public key, the votes encrypted under the RCG public key, votes decrypted with the RCG private

key and the voting receipts. The attack objectives in all the cases could be to compromise voter privacy or to disrupt the election.

As a general assumption, it is expected that the encryption and digital signature keys of the VCS and RCG, as well as the electoral board shares are destroyed once they are no longer required (usually when the results are obtained). It is also expected that vote encryption sets are not public disclosed. Only the Bulletin Board contents (voting receipts) and the decrypted votes will be public available.

a) Attacks on the votes encrypted under the election public key or the RCG public key: Crypto-analysis attacks for disclosing the encrypted voter intent are not expected in these votes. In fact it is expected that these attacks will be more probably studied in the votes decrypted by the RCG for the nature of the deterministic encryption exponent. In the other hand, any attempt to manipulate these votes to disrupt the election could be detected by checking the digital signatures and the ZKP that connect them to the deterministically encrypted votes. Therefore, no successful attacks are expected.

b) Attacks on the votes decrypted with the RCG private key: Since these votes derivate the encryption exponent in a deterministic way, it is expected that attacks will be concentrated on breaking this key derivation mechanism. Three components are used for the exponent derivation: the voting card identifier, the selected voting option and a VCS key. Considering that the set of voting options is very limited, we can consider that the strength will depend on the voting card identifier and the VCS key strength. The VCS key will have 2048 bits length and, therefore, is the main one that provides long term protection. If this key is compromised, the strength will depend on the length of the voting card identifier that usually is limited for usability issues since the voters need to type it (e.g., 80 bits). This will make an attack more feasible in a shorter timeframe, since at the end the fixed exponent protecting the privacy of the voting options in the RCG is derived from a master key with lower entropy. Solutions to increase this strength without compromising usability are under examination for future improvements. It is also important to consider that if the voting card identifiers are known in advance, the attack could be straight forward. Therefore, preserving the *VCID*'s as secret and destroying them at the end of the election is of paramount importance to prevent any possible success of this attack.

c) Voting Receipt attacks: Attacks to voting receipts could be focused to discern the original cipher text cast by the voters (since initially these encrypted votes are not public available). However, this will be practically unfeasible since the receipt value is obtained using a hash function.

6.4 Summary

The system is not vulnerable to individual component compromise, requiring in most cases the collusion of several components for staring to design an attack. The main risks are located in the election configuration period, where the voting cards and keys could be compromised. However, these keys can be created in isolated environments (i.e., do not require online processes) and the voting cards can be created using incremental steps in different isolated environments that will prevent having access to all the information at the same time.

The second important risk is related to the protection of the VCS symmetric key. Since this key is required to compensate the strength limitations of the voting card identifier (for usability reasons), it is important to it keep safe and destroy it when the election is done. Otherwise, crypto-analysis attacks could be designed that could make possible an attack in an acceptable timeframe (not confirmed yet). Further work is done in this area.

Other attacks require complex collusions that are more difficult to implement without being detected. Furthermore, the impact in case of success seems more limited to disrupt the process.

References

1. http://www.regjeringen.no/en/dep/krd/prosjekter/e-vote-2011-project/about-the-e-vote-project/presentation-of-the-project.html?id=598565
2. evalg 2011 security objectives, http://www.regjeringen.no/upload/KRD/Kampanjer/valgportal/e-valg/tekniskdok/Security_Objectives_v2.pdf
3. Pkcs#5, note = http://www.rsa.com/rsalabs/node.asp?id=2127
4. Adida, B.: Helios: Web-based open-audit voting. In: van Oorschot, P.C. (ed.) USENIX Security Symposium, pp. 335–348. USENIX Association (2008)
5. Chaum, D.: Surevote: Technical overview. In: Proceedings of the Workshop on Trustworthy Elections, WOTE 2001 (2001)
6. CESG (Communications and Electronic Security Group). E-voting security study, annex C (2002), http://www.edemocracy.gov.uk/library/papers/study.pdf2002
7. El Gamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31(4) (1985)
8. Gjøsteen, K.: Analysis of an internet voting protocol. Cryptology ePrint Archive, Report 2010/380 (2010), http://eprint.iacr.org/
9. Helbach, J., Schwenk, J.: Secure Internet Voting with Code Sheets. In: Alkassar, A., Volkamer, M. (eds.) VOTE-ID 2007. LNCS, vol. 4896, pp. 166–177. Springer, Heidelberg (2007)
10. American National Standards Institute. Accredited standards committee x9 working draft. ANSI X9.42-1993: Public Key Cryptography for the Financial Services Industry: Management of Symmetric Algorithm Keys Using Diffie-Hellman (1994)
11. Jakobsson, M.: A Practical Mix. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 448–461. Springer, Heidelberg (1998)

12. Joaquim, R., Ribeiro, C.: Codevoting: protecting against malicious vote manipulation at the voter's pc. In: Chaum, D., Kutylowski, M., Rivest, R.L., Ryan, P.Y.A. (eds.) Frontiers of Electronic Voting. Dagstuhl Seminar Proceedings, vol. 07311. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany (2007)
13. Malkhi, D., Margo, O., Pavlov, E.: E-voting without 'Cryptography'. In: Blaze, M. (ed.) FC 2002. LNCS, vol. 2357, pp. 1–15. Springer, Heidelberg (2003)
14. Markus, J., Ari, J.: Millimix: Mixing in small batches. Technical report (1999)
15. Morales-Rocha, V., Soriano, M., Puiggali, J.: New voter verification scheme using pre-encrypted ballots. Computer Communications 32(7-10), 1219–1227 (2009)
16. Nguyen, L., Safavi-Naini, R.: Breaking and Mending Resilient Mix-Nets. In: Dingledine, R. (ed.) PET 2003. LNCS, vol. 2760, pp. 66–80. Springer, Heidelberg (2003)
17. Schnorr, C.-P.: Efficient signature generation by smart cards. J. Cryptology 4(3), 161–174 (1991)
18. Storer, T.W.: Practical pollsterless remote electronic voting (2006)
19. Voutsis, N., Zimmermann, F.: Anonymous code lists for secure electronic voting over insecure mobile channels. In: Proceedings of Euro mGov. Sussex University, Brighton (2005)