VeryVote: A Voter Verifiable Code Voting System*

Rui Joaquim[1], Carlos Ribeiro[2], and Paulo Ferreira[2]

[1] ISEL, Polytechnic Institute of Lisbon
Rua Conselheiro Emídio Navarro 1, 1959-007 Lisboa, Portugal
rjoaquim@cc.isel.ipl.pt
[2] INESC-ID, Intituto Superior Técnico
Rua Alves Redol 9 - 6 andar, 1000-029 Lisboa, Portugal
carlos.ribeiro@ist.utl.pt, paulo.ferreira@inesc-id.pt

Abstract. Code voting is a technique used to address the secure platform problem of remote voting. A code voting system consists in secretly sending, e.g. by mail, code sheets to voters that map their choices to entry codes in their ballot. While voting, the voter uses the code sheet to know what code to enter in order to vote for a particular candidate. In effect, the voter does the vote encryption and, since no malicious software on the PC has access to the code sheet it is not able to change the voter's intention. However, without compromising the voter's privacy, the vote codes are not enough to prove that the vote is recorded and counted as cast by the election server.

We present a voter verifiable code voting solution which, without revealing the voter's vote, allows the voter to verify, at the end of the election, that her vote was cast and counted as intended by just performing the match of a few small strings. Moreover, w.r.t. a general code voting system, our solution comes with only a minor change in the voting interaction.

Keywords: Code voting, Internet voting, election integrity.

1 Introduction

The secure platform problem [1] of remote voting, i.e. the use of unreliable/not trustworthy client platforms such as the voter's computer and the Internet infrastructure connecting it to the election server, is one of the major problems that prevents the spread of electronic remote elections, e.g. Internet Voting.

Code voting [2, 3] is a technique that addresses the secure platform problem establishing a secure connection between the voter and the election server by means of codes printed in a code sheet previously and anonymously delivered to the voter. As explained in Sec. 2.1, this general approach only confirms that the

* This work was supported by the Portuguese Foundation for Science and Technology grants SFRH/BD/47786/2008 and PTDC/EIA/65588/2006.

P.Y.A. Ryan and B. Schoenmakers (Eds.): VOTE-ID 2009, LNCS 5767, pp. 106–121, 2009.

election server receives the right vote code. It does not prove that the candidate selected by the vote code really receives the voter's vote.

However, to guarantee the election integrity a fully (end-to-end) verifiable election system is needed. Therefore, we need to verify that the votes are cast-as-intended and counted-as-cast [4]. Cast-as-intended means that the recorded vote represents the voter's intention, and counted-as-cast means that tally correctly reflects the sum of the recorded votes. In a general code voting system we have none of these properties.

VeryVote addresses the secure platform problem in an end-to-end verifiable way. We achieve this end-to-end verifiability by adapting the MarkPledge cryptographic receipts technique [5] to a general code voting system. In the process of making a code voting system end-to-end verifiable we have compromised some of the voter's privacy to the election server, as described in Sec. 6.3. On the other hand, we provide verification mechanisms that allow an universal verification of the tally and also a simple voter verification that her vote was recorded as cast, while still protecting the voter's privacy from the general public. Additionally, our solution also simplifies the code sheet creation and distribution processes.

In the next section we present the related work, followed by and overview of VeryVote on Sec. 3. Then, in Sec. 4 we present the building blocks of our vote protocol and on Sec. 5 we describe the vote protocol. Finally, we evaluate VeryVote in Sec. 6 and conclude in Sec. 7.

2 Related Work

In this section we first present an overview of code voting systems. Then, we present the MarkPledge technique that we have adapted in order to provide the cast-as-intended and counted-as-cast properties to a generic code voting protocol.

2.1 Code Voting Systems

The first code voting implementation we are aware of was proposed in 2001 by Chaum [2], the SureVote system. Since then the code voting approach was used in the UK on some pilots of Internet, SMS and telephone voting [6,7]. In [8,9,10] you can find an analysis and some solution proposals to the vote selling problematic in a general code voting system. Oppliger and Schwenk present in [11] a proposal to improve the user friendliness of code voting by using CAPTCHAs. A different approach was used in [12], where the code sheets are generated by the voter with the help of a secure token, which in the voting phase translates the vote code to an encrypted vote in order to provide a verifiable vote tally. On the other hand, this last approach requires trust in the secure token, which can undetectably manipulate the voter's vote.

Generally, a code voting system addresses the insecure platform problem of remote voting by means of a vote protocol based on a code sheet. This approach consists in secretly sending, e.g. by mail, code sheets to voters that map their

choices to entry codes in their ballot. While voting, the voter uses the code sheet to know what code to enter in order to vote for a particular candidate. In response, the election server sends back to the voter a confirmation code, which is associated with the vote code in the voter's code sheet. If the confirmation code is right the voter knows that her vote code has reached the election server. However, a correct confirmation code does not imply the use of the voter's selection by the election server when computing the election tally.

2.2 MarkPledge Cryptographic Receipts

The goal of the Andrew Neff's MarkPledge cryptographic receipts [5] is to prove that a particular vote encryption is an encryption of a vote for the selected candidate. The proof is based on a special form of vote encryption, formally described in [5]. For sake of simplicity we present here a more informal description based on [13, 4].

The MarkPledge technique starts by a special encoding of each candidate in the ballot. A vote is formed by a sequence of numbers, one for each voting option. Each number is a special encoding of a 0 or a 1: 1 for the selected candidate and 0 for every other candidate. In consequence of this special vote encoding it is possible to encrypt the vote and then verify the encryption without disclosing the encrypted vote.

Follows a more detailed explanation of the encoding, encryption and verification of a MarkPledge vote.

Special Bit Encoding and Encryption – The encrypted vote is formed by a special bit encryption for each candidate, which we denote $BitEnc$. For the selected candidate it is encrypted a 1, $BitEnc(1)$, and for all other candidates it is encrypted a 0, $BitEnc(0)$.

Consider α as the security parameter that defines the soundness of the MarkPledge technique as $1 - \frac{1}{2^\alpha}$. Then, the $BitEnc$ of a bit b, $BitEnc(b)$, is an α-length sequence of Ballot Mark Pairs (BMPs). Each BMP is composed of two independent El Gamal ciphertexts ($BMP = [l_i, r_i]$). Each ciphertext that forms a BMP is either an encryption of a 1 ($Enc(1)$) or a 0 ($Enc(0)$).[1] The $BitEnc(b)$ is defined as follows:

- If $b = 1$, then all BMPs are of the form $[Enc(0), Enc(0)]$ or $[Enc(1), Enc(1)]$.
- If $b = 0$, then all BMPs are of the form $[Enc(0), Enc(1)]$ or $[Enc(1), Enc(0)]$.

A vote verification is based on the proof that the particular $BitEnc(b)$ of the selected candidate is a $BitEnc(1)$. Due to the special type of encryption used it is possible to prove in zero knowledge and with soundness $1-\frac{1}{2^\alpha}$ that a particular encryption $c = BitEnc(1)$. We describe a variation to the original verification scheme [13, 5], as it is more suitable for our CodeVoting adaptation goals. The

[1] Recall that in normal El Gamal encryption [14] the value 0 is not part of the plaintext domain. Therefore, what we really have are two values v_0, v_1, chosen from the plaintext domain, which respectively represent value 0 and value 1.

verification process of this variation is based on a random challenge (*chal*) to $BitEnc(b)$. Follows the details of the verification steps between a Prover $\mathcal{P}$ and a Verifier $\mathcal{V}$:

1. $\mathcal{V}$ sends $commit(chal)$ to $\mathcal{P}$, e.g. hash(*chal*).
   At this point it is necessary to commit to *chal* in order to prevent an easy vote selling/coercion attack. On the other hand, it is necessary to keep the *chal* secret to prevent the construction of a fake $BitEnc(1)$.
2. $\mathcal{P}$ sends $c = BitEnc(b)$ and a bit string (ChosenString) to $\mathcal{V}$, where bit i of ChosenString corresponds to the bit encrypted within both elements of BMP_i. Therefore, ChosenString is α bits long. Recall that only a $BitEnc(1)$ is composed of BMPs encoding two ones or two zeros.
3. $\mathcal{V}$ sends an α bits long random challenge (*chal*) to $\mathcal{P}$, which must match $commit(chal)$.
4. For each bit i of *chal*, $\mathcal{P}$ reveals the randomness[2] used for BMP_i element l_i if $chal_i = 0$, or for element r_i if $chal_i = 1$. This reveals $OpenBitEnc(b)$, an α-length bit string that results from the concatenation of the decrypted bits, one for each BMP within c.
5. $\mathcal{V}$ checks that this string matches ChosenString.

The soundness of this protocol derives from the randomness of *chal*, which ensures that, if $b = 0$, then $\mathcal{P}$ must guess the single α-length bit string that will be revealed by *chal*.

The proof is zero knowledge since $\mathcal{V}$ "cannot" use the results to prove to a third party that $c = BitEnc(1)$. $\mathcal{V}$ can only reveal one (already chosen) element of each BMP, i.e. she cannot open the $BitEnc$ for any other value of *chal*. Therefore, $\mathcal{V}$ has only a success chance of $\frac{1}{2^\alpha}$ in an attempt to prove to a third party that $c = BitEnc(1)$, i.e. the third party must choose exactly the same challenge chosen by $\mathcal{V}$.

Vote Encryption and Verification – The main idea of the vote encryption and verification steps is quite simple: i) the Voting Machine (VM) creates an encrypted ballot composed of a $BitEnc(0)$ for all candidates except for the chosen one, which will have a $BitEnc(1)$, and ii) the voter runs the verification protocol that will prove that there is a $BitEnc(1)$ associated to the chosen candidate.

We assume that the ballot is well formed, i.e. it contains only one $BitEnc(1)$. There are techniques to verify that the ballot is well formed but they are outside the scope of this paper (e.g. Neff in [5] suggests the use of techniques presented in [15, 16]).

The voter interaction protocol is thus:

1. Alice, our voter, enters her candidate choice (j) and the $commit(chal)$.

[2] Revealing the randomness allows for the ElGamal encryption reconstruction of the BMP element, and thus for the verification of the encrypted value.

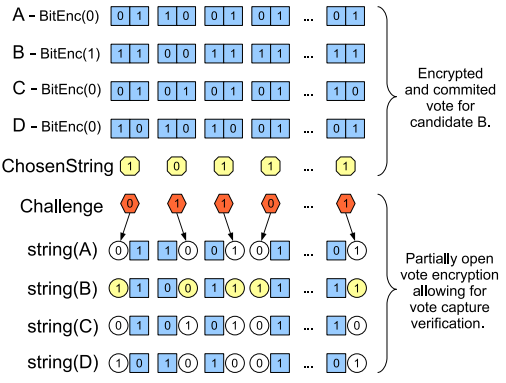

Fig. 1. MarkPledge vote encryption and verification. The vote encryption represents a vote for candidate B ($BitEnc(1)$). The values inside the circles form the $strings(i)$ and correspond to the bits of the $OpenBitEncs(b_i)$, which are revealed accordingly with the value of the challenge.

2. The VM outputs the encrypted vote and displays the ChosenString, which matches the encrypted string for $c_j = BitEnc_j(b_j)$, the bit encryption corresponding to Alice choice j. (c.f. lines B - BitEnc(1) and ChosenString in Fig. 1).
3. Alice enters $chal$, which must match $commit(chal)$.
4. The VM completes the proof that $b_j = 1$, and uses the same $chal$ to simulate the same proof for all others b_i, i.e. it uses $chal$ to create a $OpenBitEnc(b_i)$ for each $i \neq j$.

   As a result of this operation the VM outputs a vote receipt containing a text string for each index/candidate ($string(i)$). The $string(i)$ represent an encoding of the α-length bit string that results from the $OpenBitEnc(b_i)$ (in our example we use the original binary encoding).
5. Alice verifies that the ChosenString appears next to her candidate of choice, i.e. it matches $string_{(j)}$. Additionally, Alice must verify that the text strings present in the receipt match the encrypted vote, i.e. they match the corresponding $OpenBitEncs$. However this process can be leveraged using a third party organization that verifies the correctness of the vote encryption [13].

Note that, the receipt does not reveal which of $string_{(i)}$ was the real ChosenString displayed by the VM to Alice. Alice also cannot convincingly prove which string is the real one.

It is also important to emphasize that the MarkPledge verification technique only works if Alice does not reveal her $chal$ prior to the VM commitment to the encrypted vote. If VM knows the $chal$ in advance it can freely manipulate Alice's vote. This vulnerability can be exploited by message reordering and social engineering attacks [4].

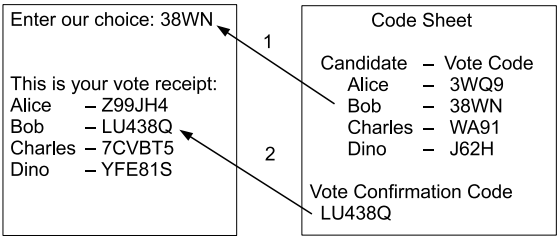

Fig. 2. In step (1) the voter uses the vote code $38WN$ to vote for the candidate Bob. Then, in step (2) the voter gets a MarkPledge receipt for her vote, therefore the vote confirmation code (the $ChosenString$ of $BitEnc(1)$) appears next to the candidate Bob.

3 VeryVote Overview

As explained in Sec.2.1, the simple verification that the election server (ES) replies with the right reply code is no proof that the ES will use the corresponding candidate when performing the election tally. The reply code only serves to prove that the right vote code has reached the ES and nothing else.

VeryVote uses a generic code voting interaction between the voter and the ES. However, it also produces an encrypted vote (and a cryptographic vote receipt) based on the received vote code. The vote encryption allows the publication of the link between the encrypted votes and the voters who casted them.

The publication of the encrypted vote/voter association allows voters to verify their votes using the cryptographic receipts produced by the ES. Additionally, a cryptographic vote tallying protocol can be used to provide proofs that the tally is computed correctly. The cryptographic vote encryption and tally verification makes VeryVote end-to-end verifiable.

To provide the end-to-end verifiable property and to allow the correction of any detected error, it is necessary for the ES to know who is casting the votes, c.f. 6.2. Therefore, the code sheet generation and delivery process can be significantly simplified. Which means that, in opposition to a traditional code voting system, in VeryVote the code sheet delivery is not anonymous, although it needs to be secret, i.e. only known to the ES and the voter. Therefore, the ES creates the code sheets, seals them, e.g. puts them inside a sealed envelop and/or adds a scratch surface to the code sheet, and then sends the code sheets to each voter, e.g. by mail. This procedure allows the ES to easily associate each code sheet to a particular voter on election day.

The code sheet produced by the ES contains a vote code for each candidate and one confirmation code that works similarly to the reply code in a traditional code voting system. However, at the voting phase, the voter besides receiving just the reply code, receives a MarkPledge cryptographic vote receipt with the confirmation code associated with the selected candidate, cf. Fig. 2.

After the election end, there is a claiming period where a voter can verify that her vote was recorded as casted. The check performed by the voter is simply a

match of her MarkPledge receipt with the one published by the ES. Nevertheless, the political parties and other third party organizations must verify the correctness of the published receipts, c.f. Sec.2.2. By using this two step verification of the MarkPledge technique it is possible to split the verification process between the voter and some other organization. The voter performs the simple verification step, leaving the more complex part of the verification to organizations with much more resources than an average voter.

During the claiming period, and if any error is detected in the verification process, a voter should be allowed to invalidate her vote. We also suggest to allow the revoking of the Internet vote at the voter's wishes to mitigate two problems with the VeryVote system: first, since VeryVote is an Internet voting system the voter may not have the desired privacy while voting, and therefore may have been exposed to some kind of coercion/vote selling; and second, the voter may not have completely understood the voting process and therefore may have doubts about the casted vote. Therefore, we argue that a secondary voting channel should be available.[3] The invalidation of the Internet vote is simple because the encrypted votes are not anonymous.

After the claiming stage, any vote that was not invalidated is included in the tally process, which is also cryptographically verifiable. Therefore, anyone can assert on the validity of the tally.

4 Protocol Building Blocks

We use some well known constructions as building blocks of our vote protocol. Therefore, and before entering into the details of our vote protocol, we present a short description of the building blocks used.

Threshold encryption scheme – We need a threshold encryption scheme to enable a vote and tally verification without compromising the voters' privacy. In a threshold encryption scheme the secret key s is shared among several authorities, the trustees $\mathcal{T}$. To recover a message encrypted with the public key it is required the cooperation of t (a configurable threshold) trustees.

Our protocol specification relies on the El Gamal encryption scheme [14]. More precisely, a variant described by Cramer et al. [16].

Mix Net – To provide an anonymous and verifiable tally we propose the use of a mix net [18]. Namely, we propose the use of a verifiable re-encryption mix net such as the one proposed by Neff in [19].

BitEnc Implementation – As in the original MarkPledge scheme [5], we use the $BitEnc$ construction based on El Gamal encryption (cf. section 2.2).

[3] A possible solution to allow the voter to cast a new vote is the one used in Estonia [17], where on election day any voter who casts a paper ballot automatically revokes the Internet vote.

Public Bulletin Board – To enable public verification of the election's integrity all non private data should be public, for which we use the usual public bulletin board construction [13, 16]. All data in the bulletin board is authenticated by means of digital signatures.

Shared Random Number Generation Protocol – In our protocol we need a random number generation by a set of trustees $\mathcal{T}$. We propose the use of a simple two round protocol to generate the shared random number:

1. In the first step each trustee t_i of $\mathcal{T}$ secretly generates a random number r_{t_i} and commits to it by publishing a signed hash of the random number.
2. After the commitment of all trustees, each trustee reveals its random number. Then, all trustees verify the correctness of the commitments published in the first step. If all commitments are correct, the shared random number is computed by applying a bitwise exclusive or to all the random numbers r_{t_i} generated by the trustees.

5 Vote Protocol

In this section we specify the vote protocol. Here we explain in detail how to securely generate a MarkPledge cryptographic vote receipt on top of a traditional code voting solution.

5.1 Protocol Notation

$\mathcal{V}, \mathcal{V}_{id}$ - respectively the voter and the voter's identifier.
CS - code sheet containing a vote code (vc) for each candidate and a vote confirmation code (vcc) used to verify the MarkPledge receipt.
ES - election server. After the election end it works like a public bulletin board publishing the encrypted votes, the final results, and all the verification data.
APP - vote client application running at the voter's PC.
$\mathcal{T}$ - the trustees. The trustees can be representatives of political parties and/or representatives of independent organizations.
E_{sk} - election threshold shared private key.
E_{pk} - election threshold generated public key.
$\{M\}_A$ - digital signature of message M with the private key of entity A. More precisely, digital signature on the result of an hash function to message M.
$A \rightarrow B : M$ - message M sent from entity A to entity B.
$A \rightarrow B \rightarrow C : M$ - message M sent from entity A to entity C using the capabilities of entity B.
$BitEnc(b)_{E_{pk}}$ - the $BitEnc$ construction for bit b using the election public key.
MP_{Venc} - MarkPledge vote encryption composed by a $BitEnc(1)_{E_{pk}}$ for the voter's chosen candidate and $BitEnc(0)_{E_{pk}}$ encryptions for all other candidates.
$chal$ - challenge which determines the elements of $BitEnc(b)_{E_{pk}}$ to be open.

$O(MP_{Venc}, chal)$ - open MarkPledge vote encryption accordingly with the *chal*, i.e. the MarkPledge receipt.

$SO(MP_{Venc}, chal)$ - the strings associated with each candidate that results from opening MP_{Venc}, i.e. the strings that represent the $OpenBitEncs$.

5.2 Election Setup

Some time before the election day the ES creates the code sheets, seals them, e.g. using a special type of envelop or a scratch surface on the code sheet, and sends them to each voter, e.g. by mail. The seal and delivery process must ensure that the code sheet remains secret to anyone but the ES and the voter.

Also some time before the election day, the trustees $\mathcal{T}$ generate the threshold election key pair. The key pair should be generated in a way that only the cooperation of t trustees is able to decrypt a message encrypted with the election public key.

The ES then uses the election public key to generate all the $BitEncs(b)_{E_{pk}}$ that will form the MarkPledge vote encryptions for all voters. Then, the ES publishes all the generated $BitEncs(b)_{E_{pk}}$ and commits to them by means of a digital signature. Being n the number of candidates, the ES generates for each voter $(n-1)$ $BitEnc(0)_{E_{pk}}$ and one $BitEnc(1)_{E_{pk}}$. The $BitEnc(1)_{E_{pk}}$ must correspond to the *vcc* of the voter's CS. Only the ES knows which $BitEnc(b)_{E_{pk}}$ is the $BitEnc(1)_{E_{pk}}$.

Our protocol is independent of the authentication method used. Therefore, we assume that sometime before the election day the voters receive their voting credentials, e.g. a username/password or an electronic voter ID card able to authenticate the voter.

Just before the election start, and after the ES commitment to all the $BitEncs(b)_{E_{pk}}$, the trustees create a shared random election value *srev* using the shared random number generation protocol. The *srev* will be used during the election to facilitate the creation of a random and unpredictable challenge to the MarkPledge vote encryption. Therefore, it is crucial for the vote protocol security that the *srev* creation occurs only after the publication of the ES commitment to the $BitEncs(b)$.

5.3 Vote Procedure

The vote procedure starts when the $\mathcal{V}$ opens the APP. Then, the following takes place:

1. $\mathcal{V} \rightarrow APP \rightarrow ES : \mathcal{V}_{id}$ The $\mathcal{V}$ authenticates herself to the ES. For simplicity, we only show a message containing $\mathcal{V}_{id}$. However, in practice a message containing a username/password would be exchanged or, in the case of strong authentication by means of digital signatures, a strong mutual authentication protocol should be used, e.g. the X.509 three-pass authentication.
2. $\mathcal{V} \rightarrow APP \rightarrow ES : vc_f$ After a successful authentication, the voter votes by looking into her CS and typing the vote code associated with her favorite candidate vc_f, which is then sent by the APP to the ES.

3. $ES \rightarrow APP : O(MP_{Venc}, chal), \{O(MP_{Venc}, chal)\}_E S$ After receiving the vote code selected by the voter vc_f, the ES prepares a MarkPledge encryption (MP_{Venc}) and the corresponding receipt ($O(MP_{Venc}, chal)$):

   (a) The ES builds a MP_{Venc} with the previously committed $BitEncs(b)_{E_{pk}}$. The MP_{Venc} is composed by the committed $BitEnc(1)_{E_{pk}}$ for the selected candidate and a random selection of the committed $BitEncs(0)_{E_{pk}}$ for each other candidate.
   (b) To prove the correction of the vote encryption to the voter, the ES then generates a random challenge to the MP_{Venc}. The challenge $chal$ is simply the hash of the concatenation of the MP_{Venc} with the $srev$. Since the $srev$ value was not known to the ES at the time of the CS creation and distribution, this process of challenge generation results in a kind of Fiat-Shamir heuristic [20] making possible a non interactive proof of the MP_{Venc} correction.
   (c) Finally, the ES creates the MarkPledge receipt $O(MP_{Venc}, chal)$ by opening the MP_{Venc} accordingly with the generated $chal$.

   The MarkPledge signed receipt is then sent to the APP.
4. $APP \rightarrow \mathcal{V} : SO(MP_{Venc}, chal)$ The APP then verifies the signature and the correctness of the $O(MP_{Venc}, chal)$. If the verification is successful it presents to the voter a vote receipt composed of the strings ($OpenBitEncs$ encodings) that result from $O(MP_{Venc}, chal)$. The voter then performs a first receipt check and confirms that the vcc appears associated with the selected candidate in the $SO(MP_{Venc}, chal)$, cf. Fig. 2.

   The vote receipt can also be printed in order to facilitate a second, and stronger, verification at the claiming stage.
5. (optional) $APP \rightarrow ES : \{O(MP_{Venc}, chal)\}_{\mathcal{V}}$ If a strong authentication mechanism is used to authenticate the voters, e.g. a digital signature performed by an electronic voter ID card, then a last message validating the vote is sent to the ES. This message consists in a signature on the MarkPledge receipt.

5.4 Claiming Stage

When the election ends there is a small time period, that we call the claiming stage, where a voter can check and revoke her vote.

Right after the election end the ES publishes all the generated $O(MP_{Venc}, chal)$, the $SO(MP_{Venc}, chal)$ and the identification of the voter who "owns" that vote. At this point anyone can verify the correctness of the $O(MP_{Venc}, chal)$ by checking: i) the $BitEncs(b)$ used in its construction, ii) the correctness of the value of $chal$, and iii) the correctness of the $SO(MP_{Venc}, chal)$. Since this validation process is somewhat complex we only assume that the political parties and/or independent organizations perform this validation. Additionally, these third party organizations should also verify that all the MP_{Venc} are well-formed, c.f. sec. 2.2.

If the $SO(MP_{Venc}, chal)$ is validated by a third party the voter only has to check that the receipt matches the one published in her name. A match means that with a $(1 - \frac{1}{2^\alpha})$ probability the encrypted vote represents the voter choice.

5.5 Tallying the Votes

Publishing the election's tally is straightforward because the ES knows the exact contents of each vote; however, to prove the correctness of the tally it is necessary to perform some additional steps.

After the claiming stage all votes that were not revoked by the voters are considered valid for the tally. Then, the validated votes go through a verifiable mix-net protocol [21, 19]. At the end of the anonymization process, the trustees decrypt and publish the votes in a shared and verifiable way [16]. The published vote decryption allows the political parties and other interested entities to verify the correctness of the vote decryption. With the validated vote decryption everyone can use the clear votes to perform/validate the tally.

Note that the mixing and decryption of the votes can not be performed over the $BitEnc(b)$ constructions of the encrypted vote. The reason why is the following: if the individual elements of the $BitEnc(b)$ construction were decrypted, even after mixing, it would be trivial to correlate a decrypted vote with the corresponding encrypted vote, therefore loosing all vote anonymity. To solve this problem an additional standard El Gamal bit encryption for each $BitEnc(b)$ must be added to the MP_{Venc} [22]. It is then proved that the each single standart bit encryption corresponds to the bit encrypted within the corresponding $BitEnc$ construction. Finally, the mixing and decryption processes take as input these new bit encryptions instead of the $BitEnc$ constructions, therefore protecting the voter's privacy. These new bit encryptions do not change the voter interaction in any way. The validation of the new bit encryptions is part of validation of the MP_{Venc} well-formedness.

Note that, the reason we use a threshold shared election key pair instead of one generated by the ES is that if the records of the CS creation are destroyed jointly with the information of which $BitEncs(b)$ are $BitEncs(1)$ the votes become secret even to the ES. The destruction of such data can be assured by physical procedures under the supervision of the political parties. Therefore, at that stage the privacy of the voters is in the hands of who has the election key, and that is the reason why we suggest the use of a threshold election key shared among several trustees. Nevertheless, we must point out that if such measures take place it is also necessary to prevent subliminal channels in the randomness used by the ES, i.e. kleptograhic attacks [23]. If there is simple a key pair generated by the ES the tally procedure is as previously, only now the vote decryption step is done by the ES instead of being done by the trustees.

6 Evaluation

In this section we show that VeryVote, besides addressing the uncontrolled client platform problem of Internet voting, also gives strong overall election integrity

guarantees, while still protecting the voter's privacy. We start by presenting the assumptions used in our evaluation followed by the election integrity evaluation. Finally, we present an analysis on the privacy guarantees of VeryVote.

6.1 Assumptions

We assume the following:

- The codes in the code sheets are randomly generated by the ES and secretly delivered to the voters, i.e. only the voter and the ES know the codes in the voter's code sheet.
- No more than $t-1$ of the trustees are dishonest.
- The cryptographic primitives and constructions used are secure and verifiable: El Gamal threshold encryption, mix net and $BitEnc$ constructions; digital signatures and hash functions.
- The vote codes and the vote confirmation code are composed of respectively 4 and 6 alphanumeric symbols. We assume the use of 62 symbols, all uppercase and lowercase letters and digits, which gives 62^4 (roughly 14.7 millions) possible values for the vote codes and 62^6 (roughly 56.8 thousand millions) possible values for the vote confirmation code.
- The voter has in her possession the printed vote receipt.
- The political parties and/or independent organizations verify the data published by the ES.
- At the claiming stage the voter verifies her vote with the printed receipt.
- When the election ends the records of the CS creation and the information of which $BitEncs(b)$ are $BitEncs(1)$ are physically destroyed or in alternative we assume that the ES is trustworthy in what concerns the voters' privacy.

6.2 Election Integrity

Election Integrity at the Uncontrolled Client Platform – In VeryVote the APP running at the uncontrolled client platform has only a negligible chance to manipulate the voter's vote, i.e. it would need to guess a valid vote code different from the one used by the voter. In an election with n candidates the chances of that happening are $(n-1)/62^4$. Only in this case the APP can change the voters vote and produce a fake vote receipt that fools momentarily the voter, i.e. the voter will easily detect the vote manipulation while verifying her vote receipt at the claiming stage.

In order to prevent a simple denial of service attack by the APP, a voter should be able to submit several vote codes at least until she submits one valid vote code (the vote update issue will be discussed later in privacy analysis). Therefore, measures should be taken to prevent the APP from trying a significant amount of vote codes guesses, such as the introduction of a delay or requiring the solution of a CAPTCHA in each try.

Nevertheless, even if the APP guesses a valid vote code the voter would catch the misbehavior at the claiming stage, while confirming her vote receipt with the ES published data.

Election Integrity at the Vote Record and Tallying Processes – Since we assume the use of secure and verifiable constructions for the threshold encryption and mix nets, we automatically have the integrity verifications of the tallying process. What we have new in VeryVote is the possibility to verify that the vote is recorded as cast. Therefore, we only present here an analysis on the integrity of the vote record process.

Because the ES publishes all encrypted votes along with the vote receipts and the association with the voter who "cast" it, it is trivial for the voter to detect a receipt different from the one she has. Therefore, the only way the ES can pass such verification is to give the voter a "tampered" vote receipt, i.e. a vote receipt that shows the right association between the vcc and the candidate selected by the voter but which in fact encrypts a vote for a different candidate.

Since the ES commits to the $BitEncs(b)$ that form the vote before the generation of the $srev$, there is no possibility to known in advance what the challenge will be. Therefore, the only possibility to construct a tampered receipt is to try all the combinations possible for the $BitEncs(b)$ positions in the receipt, and hope that luck provides the correct challenge to the vote encryption.

However, if the committed $BitEnc(1)$ corresponds to the confirmation code on the voter's code sheet, then in a tampered receipt there will be the same value (the confirmation code) associated to two distinct candidates. To prevent that from happening such type of receipts should be considered invalid, which prevents the attack. In the case of accidental creation of an invalid receipt the ES must create another one using another combination of the $BitEncs(0)$.

Consequently, the only way the ES has to produce a valid tampered receipt is to commit to a $BitEnc(1)$ that is not related to the confirmation code in the voter's code sheet. However, this would not be a smart move because then the ES only has a probability of $n!/62^6$ to create a valid (tampered or not) receipt for the voter. For instance, if $n = 10$ the probability of success is less than 0.01%.

Although, it is worth noticing that because of the exponential nature of the factorial function, increasing much more the number of candidates will force the use of a very long vote confirmation code, which makes the system unusable. On the other hand, for a smaller number of candidates and the same probability of success a smaller confirmation code can be used, which makes the receipt shorter and more easily verifiable.

Another problem that could affect the election integrity verification of the vote record is someone tampering with the CS in the distribution process. In this case the voter will catch the misbehavior with a very high probability $(1 - (1/62^6))$. The voter can then revoke her vote a correct the problem.

The Advantages of Using MarkPledge Receipts – Saying that in a traditional code voting system the voter cannot verify that her vote is used in the tally is not entirely true. If each code sheet is published after the election with the corresponding selection made by the voter who used it, then the voter can verify that her vote was correctly used. Note that this publication implies the publication of the election tally because now anyone can sum up the casted votes.

Assuming that the code sheets were anonymously distributed to the voters, the verification described above is anonymous. However, now a voter cannot complain and correct her vote. Changing a vote after knowing the final results of the election is not democratic and implies that the voter must prove that a particular vote is hers.

Note also that, if both the code sheets distribution and the voting process are anonymous, there is nothing that prevents the entity that created the codes sheets to impersonate an anonymous voter and cast a vote for her. No one knows who has voted.

Using MarkPledge receipts we allow the publication of the encrypted vote along with the identity of the voter who "casted" the vote. Then, and before anyone knowns the election results, every voter can verify her vote and correct it without revealing the contents of the encrypted vote. Therefore, only validated votes will be part of the election tally. Additionally, any attempt to introduce votes for voters who did not voted can be identified or prevented if a strong voter's authentication mechanism is used.

6.3 Voter's Privacy

VeryVote makes possible the verification that the voter's intention is really used in the tally. However, to allow this verification part of the voters' privacy was lost, namely in what concerns the ES.

In VeryVote, the ES knows each voter's choice, at least before the destruction of the creation records of the code sheets and $BitEncs(b)$ by physical procedures at the end of the election. Assuming that there are no subliminal channels, what remains are the votes encrypted with a threshold encryption key shared among the trustees.

The encrypted votes are then anonymized in a verifiable way by the mix. At last the anonymized votes are decrypted by the trustees. Assuming a threshold value of t, then only the cooperation of t trustees can decrypt the anonymized votes. Consequently, if no more than $t - 1$ trustees are dishonest only the anonymized votes are decrypted and the voters' privacy is protected. No one, not even the trustees known which vote belongs to which voter.

Finally, it is important to analyze the implications of the MarkPledge receipt in the voter's privacy. In all code voting systems the code sheet together with the vote receipt (reply code or MarkPledge receipt in our case) can be considered a proof of vote. Therefore, the voter plays a big role in protecting her own privacy, i.e. a voter to protect her own privacy must keep the code sheet secret. However, in opposition to a simple reply code, the MarkPledge receipt allows for an anonymous verification of the vote, as described in Sec. 6.2.

Vote Buying and Coercion – As described in Sec. 6.3, the voter can build up a receipt if she joins the MarkPledge receipt with her code sheet. Therefore, this proof can be used to facilitate vote buying/selling or coercion.

Nevertheless, it is possible to discourage vote buying using the vote update technique [24]. VeryVote already has the vote update possibility in the claiming stage. Since a voter must not justify why she is revoking the previous casted vote, the

vote revoke facility can be used as merely an opportunity to update a previous casted vote. Therefore, the attacker can only confirm the vote receipt validity after the claiming stage. This late verification automatically discourages vote buying attacks: if the attacker pays in advance it can be cheated by the voter, and if the attacker only pays after the election end, the voter can be cheated by the attacker. Therefore, the vote update possibility and the mutual distrust between the voter and the vote buyer should be enough to discourage such attacks.

The coercion problem is not so easily mitigated because it is more a psychological attack, and therefore may work independently of the voting technology used. Nonetheless, there are particular cases of coercion, e.g. family voting, that work better in uncontrolled voting environments, e.g. vote by mail and Internet voting. It is worth noticing that the possibility of vote update offered by VeryVote can in some extent minimize the coercion problem but it does not solve it.

7 Conclusions

VeryVote is a code voting sytem that addresses the secure platform problem. However, unlike other code voting systems, and due to the use of MarkPledge vote receipts, VeryVote is end-to-end verifiable. In VeryVote the submitted votes are encrypted and published in a non anonymous way. Therefore, a voter can check that her vote was correctly recorded by using her MarkPledge vote receipt. After a claiming stage where the voters can revoke and update their previously submitted vote, the valid votes enter into a verifiable vote tally process. The verification of both the correctness of the recorded votes and of the election tally process makes VeryVote end-to-end verifiable.

Nevertheless, the end-to-end verifiability also carries more responsibility to the voter. Now the voter can verify that her vote is counted as intended but she must also take an active role protecting her own privacy by keeping her code sheet secret. It is also worth noticing that in order to introduce the end-to-end verifiability we have made the voting interaction between the voter and the ES more complex, which may cause some usability problems.

As future work it would be interesting to study the usability of the system, and if it is possible to eliminate the election server as a trustworthy entity with respect to the voters' privacy.

Acknowledgments. The authors would like to thank the valuable comments of the anonymous reviewers.

References

1. Rivest, R.L.: Electronic voting. In: Syverson, P.F. (ed.) FC 2001. LNCS, vol. 2339, pp. 243–268. Springer, Heidelberg (2001)
2. Chaum, D.: Surevote. International patent WO 01/55940 A1 (2001), `http://www.surevote.com/home.html`
3. Oppliger, R.: How to address the secure platform problem for remote internet voting. In: Erasim, E., Karagiannis, D. (eds.) 5th Conference on Sicherheit in Informationssystemen (SIS 2002), pp. 153–173. vdf Hochschulverlag, Vienna (2002)

4. Karlof, C., Sastry, N., Wagner, D.: Cryptographic voting protocols: A systems prespective. In: 14th USENIX Security Symposium, pp. 33–50 (2005)
5. Neff, A.: Practical high certainty intent verification for encrypted votes (2004), http://www.votehere.com/old/vhti/documentation/vsv-2.0.3638.pdf
6. UK's Electoral Commission: Technical report on the may 2003 pilots (2003), http://www.electoralcommission.org.uk/about-us/03pilotscheme.cfm
7. UK's National Technical Authority for Information Assurance: e-voting security study (2002), http://www.ictparliament.org/CDTunisi/ict_compendium/paesi/uk/uk54.pdf
8. 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)
9. Oppliger, R., Schwenk, J., Helbach, J.: Protecting code voting against vote selling. In: An Analytical Description of CHILL, the CCITT High Level Language. LNI, vol. 128, pp. 193–204. GI (2008)
10. Helbach, J., Schwenk, J., Schage, S.: Code voting with linkable group signatures. In: EVOTE 2008 (2008)
11. Oppliger, R., Schwenk, J.: Captcha-based code voting. In: EVOTE 2008 (2008)
12. Joaquim, R., Ribeiro, C.: Codevoting protection against automatic vote manipulation in an uncontrolled environment. In: Alkassar, A., Volkamer, M. (eds.) VOTE-ID 2007. LNCS, vol. 4896, pp. 178–188. Springer, Heidelberg (2007)
13. Adida, B., Neff, A.: Ballot casting assurance. In: EVT 2006, Vancouver, B.C., Canada, USENIX/ACCURATE (2006)
14. ElGamal, T.: A public-key cryptosystem and signature scheme based on discrete logarithms. IEEE Transactions on Information Theory IT-31(4), 469–472 (1985)
15. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (1994)
16. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 103–118. Springer, Heidelberg (1997)
17. Estonian National Electoral Commitee: Internet voting in estonia (2007), http://www.vvk.ee/engindex.html
18. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM 24(2), 84–88 (1981)
19. Neff, C.A.: Verifiable mixing (shuffling) of elgamal pairs (2004)
20. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)
21. Jakobsson, M., Juels, A., Rivest, R.: Making mix nets robust for electronic voting by randomized partial checking. In: 2002 USENIX Security Symposium, San Francisco, CA, USA, pp. 339–353 (2002)
22. Adida, B.: Advances in Cryptographic Voting Systems, PhD thesis. MIT (2006)
23. Gogolewski, M., Klonowski, M., Kubiak, P., Kutyłowski, M., Lauks, A., Zagórski, F.: Kleptographic attacks on e-election schemes with receipts. In: Müller, G. (ed.) ETRICS 2006. LNCS, vol. 3995, pp. 494–508. Springer, Heidelberg (2006)
24. Volkamer, M., Grimm, R.: Multiple casts in online voting: Analyzing chances. In: Robert Krimmer, R. (ed.) Electronic Voting 2006, Castle Hofen, Bregenz, Austria, GI. LNI, vol. P-86, pp. 97–106 (2006)