Vote Validatability in Mix-Net-Based eVoting

Pedro Bibiloni[1(✉)], Alex Escala[2,3], and Paz Morillo[3]

[1] Departament de Matemàtiques i Informàtica,
Universitat de Les Illes Balears, Palma, Spain
pedro@bibiloni.es
[2] Scytl Secure Electronic Voting, Barcelona, Spain
alex.escala@ma4.upc.edu
[3] Departament de Matemàtica Aplicada IV,
Universitat Politècnica de Catalunya, Barcelona, Spain
paz@ma4.upc.edu

Abstract. One way to build secure electronic voting systems is to use Mix-Nets, which break any correlation between voters and their votes. One of the characteristics of Mix-Net-based eVoting is that ballots are usually decrypted individually and, as a consequence, invalid votes can be detected during the tallying of the election. In particular, this means that the ballot does not need to contain a proof of the vote being valid.

However, allowing for invalid votes to be detected only during the tallying of the election can have bad consequences on the reputation of the election. First, casting a ballot for an invalid vote might be considered as an attack against the eVoting system by non-technical people, who might expect that the system does not accept such ballots. Besides, it would be impossible to track the attacker due to the anonymity provided by the Mix-Net. Second, if a ballot for an invalid vote is produced by a software bug, it might be only detected after the election period has finished. In particular, voters would not be able to cast a valid vote again.

In this work we formalize the concept of having a system that detects invalid votes during the election period. In addition, we give a general construction of an eVoting system satisfying such property and an efficient concrete instantiation based on well-studied assumptions.

Keywords: Electronic voting systems · Mix-Nets · Formal definitions

1 Introduction

Even though many electronic voting schemes have been proposed, we could argue that two of the most important conceptual categories are *Homomorphic Tallying* based voting schemes and *Mix-Net* based voting schemes. Both types of schemes consist on having the voter encrypt her selected voting option on her voting device and having an electoral authority decrypting these encrypted voting options, after

P. Bibiloni was partially supported by the Spanish project TIN 2013-42795-P and the fellowship FPI/1645/2014, which was cofinanced by the European Social Fund.

R. Haenni et al. (Eds.): VoteID 2015, LNCS 9269, pp. 92–109, 2015.
DOI: 10.1007/978-3-319-22270-7_6

some anonymization procedure. This anonymization procedure is what defines each category and has implications on the whole voting system.

In *Homomorphic Tallying* based voting systems [8], the anonymization procedure consists on homomorphically aggregating the encryptions of the voting options from different voters to obtain the encryption of the aggregated selections. For instance, if each voter computes as many encryptions as voting options exist in the election, the aggregate would be the encryption of the number of votes for the first voting option, the encryption of the number of votes for the second voting option and so on. This would imply that electoral authorities only need to compute one decryption operation per voting option in the election, whereas each voter would need as many encryptions.

This does not fit well within our paradigm, since electronic voting is an extremely asymmetric scenario: the computational power of a single voter's device is much smaller than the computational power of the electoral authority. This is due to two factors. First, the resources available to a single voter (personal computers, smartphones or tablets) are usually considerably lower than those available to electoral authorities (multiple servers with many cores per server), specially in big elections. Second, the recent trend seems to be to implement the voting client in JavaScript, which performance is orders of magnitude lower than Java or C, the languages in which the back-end of the system is usually implemented. There are technologies which improve the performance of JavaScript but they are not available in all web browsers. Finally, the time it takes to tally an election is less critical than the time it takes to cast a vote. As estimated in [2], encrypting a single candidate in the JavaScript implementation of Helios, the most popular implementation of a Homomorphic Tallying-based voting system, takes up to 1 s. This clearly does not scale well when hundreds or thousands of candidates are eligible.

On the other hand, in *Mix-Net* based voting systems [7], the computational cost for the voter is much smaller than the computational cost for the electoral authority and even smaller than the computational cost in a Homomorphic Tallying-based scheme. This is achieved by changing the anonymization procedure, which consists on shuffling the encrypted voting options to break any correlation between the ballot and the voter. In this case, the voter only needs to encrypt an encoding of her selected voting options, which might be as efficient as computing a single encryption. On the other hand, the electoral authority will need to decrypt all the ciphertexts individually, but that's a reasonable trade-off.

To have a fair comparison between these two categories of voting systems, one has to consider how many voting options exist in an election. If the election is a single referendum answer, there will be usually three answers (yes, no, blank), which implies that using a Homomorphic Tallying-based system is more than reasonable. However, there are elections where a voter might choose between close to a thousand candidates. We find such an example in elections where tens of parties are eligible in an election, each party having close to a hundred candidates and a voter being able to choose candidates from any party, in which case the benefits of using a Mix-Net-based system outweight its disadvantages.

Lastly, it should be noted that the Damgård Jurik cryptosystem [9] allows encrypting several candidates in a single ciphertext, pushing the boundaries of Homomorphic Tallying-based voting systems. However, only a certain number of candidates can be encrypted, which depends on the number of voters. Above this number, more than one ciphertext needs to be computed and casting a ballot becomes more costly. Determining at which point Mix-Net-based systems outperform Homomorphic Tallying-based systems (with respect to the voting client) is outside the scope of this paper.

1.1 The Problem of Invalid Votes

When building a Homomorphic Tallying based voting system, a technical requisite is that the voter must construct a proof that her vote conforms to the election rules. Otherwise, the homomorphic aggregation of invalid votes could produce completely unreasonable results. Current Homomorphic Tallying based systems consider this requirement, so we can consider it a solved problem.

Mix-Net-based systems do not have this requirement. As votes are individually decrypted, it can be checked whether each decrypted vote conforms to the election rules and, in case it does not, consider it an invalid vote. From a technical perspective this is completely reasonable, we do not need to ask voters for a proof of her vote conforming to the election rules in order to have a secure Mix-Net-based system. Indeed, this is how paper voting systems work nowadays.

Despite proofs of ballot well-formedness not being necessary to implement a secure eVoting system, the lack of such proofs might affect the reputation of the system. Firstly, from a non-technical voter's perspective, it is reasonable to assume that if the voting interface does not allow for an invalid vote to be cast then invalid votes should be impossible to cast. Therefore, modifying the voting client to cast an invalid vote might be seen as an attack against the system, *even if it has no effect on the result on the election*. Besides, it would be impossible to track the attacker due to the anonymity provided by the Mix-Net. The paper voting scenario is slightly different: in paper voting the voting interface allows voters to easily cast an invalid ballot. In addition, should there be a software bug which created invalid votes inadvertently, this would be only detected at the tallying phase. Depending on the amount of invalid votes, the election might even have to be restarted – with the reputation loss that it represents.

1.2 Introducing Vote Validatability

In this work we introduce the concept of *vote validatability*, which attempts to solve the problem mentioned above. We consider that an electronic voting system has vote validatability if it can be *publicly verified* that a ballot contains a vote conforming to the election rules – we want to be able to detect whether a vote is invalid before it is decrypted. This means that (a) no invalid votes will appear during the tallying of the election and (b) any software bug in the voting devices

will be detected during the election period, so it can be quickly fixed, providing the voters another attempt to vote before the end of the election.[1]

As we discussed above, Homomorphic Tallying systems have vote validatability since it is a requisite in order to have a secure voting system, in contrast with Mix-Net-based systems. Adding this property to a Mix-Net-based scheme is not a theoretical problem: there are inefficient cryptographic tools such as general-purpose zero-knowledge proofs which can be used to achieve it. The challenge is thus using appropriate proofs to retain low computational cost from the voters' side, which is one of the advantages of using a Mix-Net-based system.

There is a trivial approach to achieve vote validatability: considering all possible contents of a ballot as valid. This can be done by defining an encoding for all-but-one eligible candidates and assigning any other encoding to the last candidate. However, this has some drawbacks. First, having several encodings for the same candidate opens the door to facilitating vote selling by making it possible to introduce the voter's identity in the encoding of the candidate. In addition, requiring a specific encoding might limit the amount of features of the eVoting system, such as the so called return codes [11], which require that each candidate has only one encoding, or using special encodings to aggregate encryptions before the tallying, as also done in [11]. Therefore, we prefer a modular solution to vote validatability which does not require a specific encoding of the candidates.

In this work, we introduce a formal definition for the concept of vote validatability in Sect. 3. Then, we give a general construction of a Mix-Net-based scheme achieving vote validatability and privacy. This construction is based on basic cryptographic primitives and is given in Sect. 4, along with its security properties. Finally, we give a concrete, efficient instantiation of a Mix-Net-based system with vote validatability in Sect. 5.

2 Preliminaries

2.1 Encryption Schemes

An encryption scheme consists of three probabilistic polynomial time (p.p.t.) algorithms: $\mathsf{KeyGenEnc}, \mathsf{Enc}, \mathsf{Dec}$. On input a security parameter 1^k, the $\mathsf{KeyGenEnc}$ algorithm outputs a public key pk and a secret key sk, implicitly defining a message space M_e. The Enc algorithm takes as input the public key pk and a message $m \in M_e$ and outputs a ciphertext C. Dec takes as input a secret key sk and a ciphertext C and outputs a message $m \in M_e$ or halts outputting $\perp$.

An encryption scheme is NM-CPA (Non-Malleability under a Chosen Plaintext Attack) if, loosely speaking, no adversary can find a non-trivial relation between the plaintexts hidden in some ciphertexts generated by him, querying the encryption oracle as in the IND-CPA experiment [5].

[1] There could be bugs in the software which verifies vote validatability. However, this verification can be done in parallel by different implementations done by different entities, leveraging this risk.

2.2 Signature Schemes

A signature scheme consists of three p.p.t. algorithms $\mathsf{KeyGenSign}, \mathsf{Sign}$, $\mathsf{VerifySign}$. On input a security parameter 1^k, the $\mathsf{KeyGenSign}$ algorithm outputs a public key pk and a secret key sk, implicitly defining a message space M_s. The Sign algorithm takes as input the secret key sk and a message $m \in M_s$ and outputs a signature σ. The $\mathsf{VerifySign}$ algorithm takes as input a public key pk and a signature σ and outputs success (1) or reject (0).

One usual notion of security for a signature scheme is EUF-CMA [13]. In such a scheme, no adversary is able to forge a new valid signature for any message not already signed, regardless the number of signatures issued.

2.3 Pseudo-Random Permutations

A Pseudo-Random Permutation family [15] is a family of efficient functions $F_{(\cdot)} : \mathcal{X} \to \mathcal{X}$ parametrized by a key $k \in K_{PRP}$.

The pseudo-random property of a PRP family states that it is difficult to distinguish the outputs of a function F_k for a random key $k \in K_{PRP}$ from those of a function f chosen at random from the space of random permutations of $\mathcal{X}$.

2.4 Non-Interactive Zero-Knowledge Proof of Knowledge

Let R be a relation, containing pairs (x, w) such that, given (x, w) it can be verified in polynomial time whether $(x, w) \in R$. We call x the statement and w the witness. We define the language L_R as the set of statements x for which there exists a witness w such that $(x, w) \in R$.

A non-interactive zero-knowledge proof of knowledge (NIZKPK) for a language L_R consists of three p.p.t. algorithms: $\mathsf{GenCRS}, \mathsf{Prove}, \mathsf{VerifyProof}$. GenCRS takes as input a security parameter 1^k and outputs a common reference string crs. Prove takes as input the common reference string crs, a statement x and a witness w such that $(x, w) \in R$ and outputs a proof π. $\mathsf{VerifyProof}$ takes as input a common reference string crs, a statement x and a proof π and outputs 1 if it accepts the proof or 0 if it rejects it.

A NIZKPK must satisfy the properties of completeness, witness extraction and zero-knowledge (see, for instance, [16]). Intuitively, completeness states that $\mathsf{VerifyProof}$ will always return 1 on correctly generated proofs. Witness extraction states that (a) there exists an algorithm $\mathsf{ExtGenCRS}$ which outputs a common reference string $\widetilde{\mathsf{crs}}$, indistinguishable from a common reference string output by GenCRS, and a trapdoor key tk; and (b) that there exists an algorithm $\mathsf{Extract}$ that, on input the trapdoor key tk, a statement x and a valid proof π it returns a witness w such that $(x, w) \in R$. Finally, zero-knowledge states that (a) there exists some $\mathsf{SimGenCRS}$ which outputs a common reference string $\widetilde{\mathsf{crs}}$, indistinguishable from a common reference string output by GenCRS, and a simulation key fk; and (b) that there exists an algorithm $\mathsf{SimProve}$ that, on input a statement x and the simulation key fk it can generate a proof indistinguishable from a proof generated using the Prove algorithm with a valid witness.

3 Definitions

In this section we present the syntactical definition of an eVoting scheme and we define some security properties. We have not considered all the desirable security properties of an eVoting scheme – all the end-to-end verifiability properties, including the handling of voters' credentials, are considered to be out of the scope of this paper. However, the solution given in this paper can be combined with the usual techniques for achieving end-to-end verifiability.

3.1 Syntactical Definition

We now give the syntax of a voting scheme. We will consider single-pass voting schemes as defined in [5], which are characterized by the fact that voters interact with the system only by submitting their ballots.

We will consider the following entities regarding an election. First, *election authorities* are in charge of defining the election parameters, generating any required cryptographic keys and tallying the result of the elections. The *bulletin board* is a repository of information containing public keys and ballots. It can be read by any entity but only the *bulletin board manager* and the election authorities can write to it. *Voters* participate in the election by choosing their preferred voting options and submitting their ballots. For the sake of simplicity, we will assume that there is only one election authority. This assumption can be avoided with well-known tools such as multi-party computation.

A voting scheme is parametrized by the set of possible votes $\mathbb{V}$, a result space R and a result function $\rho : (\mathbb{V} \cup \{\perp\})^* \rightarrow R$, where $\perp$ denotes an invalid vote.

The result function states how votes should be tallied, i.e., which counting function should be applied to votes. One such result function which we are interested in is the *multiset function*. As defined in [4], the multiset function discloses the sequence of all the cast votes, in a random order. In this case, an invalid vote is treated as any other vote.[2]

A voting scheme is defined by the following p.p.t. algorithms:

- $\mathsf{Setup}(1^\lambda)$ on input a security parameter 1^λ it outputs an election public key pk and an election secret key sk.
- $\mathsf{Vote}(pk, v)$ on input the election public key pk and a vote $v \in \mathbb{V}$, outputs a ballot b.
- $\mathsf{ValidateBallot}(BB, b)$ takes as input a bulletin board BB and a ballot b. It outputs either success (1) or reject (0).
- $\mathsf{Tally}(sk, BB)$ on input the election secret key sk and the bulletin board BB. It outputs the tally $r \in R$ together with a proof of correct tabulation Π.
- $\mathsf{VerifyTally}(BB, r, \Pi)$ takes as input the bulletin board BB, the tally r and a proof of correct tabulation Π. It outputs either success (1) or reject (0).

A single-pass protocol is executed in three phases.

[2] As in [4], the result function can be used to model revote policies. In this work we just consider the scenario where each voter can only cast one vote.

1. In the *setup* phase, the election authority runs the Setup algorithm. It publishes the election public key pk in the bulletin board BB and keeps the election secret key sk.
2. In the *voting* phase, each voter can vote. To vote, the voter chooses a vote v and retrieves the public key pk from the bulletin board. Both v and pk are used to create a ballot b using the Vote algorithm, which is sent to the bulletin board manager. The bulletin board manager then executes the ValidateBallot algorithm on the ballot. If the algorithm returns 1, then the bulletin board manager adds the ballot to the bulletin board. Otherwise, it rejects the ballot and notifies both the voter and the electoral authority for auditability purposes.
3. In the *counting* phase, the election authority runs the Tally algorithm on the bulletin board using the election secret key. The output of the Tally algorithm, which consists of the result r and the correct tabulation proof Π, is published to the bulletin board. The proof Π can then be verified by any entity using the VerifyTally algorithm.

A voting system as defined above is *correct* if, when the three phases are run with all the participants behaving correctly, then (a) the result r output by the Tally algorithm is equal to the evaluation of the result function ρ on the voting options corresponding to the ballots cast by the voters and (b) the algorithm VerifyTally on input the result of the Tally algorithm returns success.

3.2 Privacy

Intuitively, a voting system has ballot privacy if an adversary with access to all the ballots and the public key of the election is not able to get any information about the voters' preferences. Formalizing this intuition turns out to be non-straightforward, and it is not until recently that *good* definitions have been given. We adopt the formalization given in [4], a game-based definition of ballot privacy, proven to be equivalent to the intuitive simulation-based security notion.

Ballot privacy is defined by using two experiments between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. As usual, the goal of the adversary is to distinguish between the two experiments. In both experiments, the adversary may corrupt voters and submit ballots on their behalf. In addition, for each honest voter the adversary can specify two votes to be used for casting her ballot. However, the electoral authority is assumed to remain honest. Depending on the experiment, the challenger will cast a ballot containing either of those two votes. To prevent trivial attacks, the same tally is always shown to the adversary regardless of which experiment is being played.

For compactness, we present the two experiments as a single experiment which depends on a bit $\beta \in \{0, 1\}$. Both experiments assume given the set of voting options $\mathbb{V}$, the result space R, the result function ρ and use an algorithm $\mathsf{SimProof}(BB, r)$ which, given a bulletin board and a result, simulates a correct tabulation proof. The experiment Exp_β is run in these phases:

1. **Setup phase.** The challenger sets up two empty bulletin boards BB_L and BB_R. It runs the $\mathsf{Setup}(1^\lambda)$ protocol to obtain the election public key pk and the election private key sk. It then posts pk on both bulletin boards. The adversary is given read access to either BB_L if $\beta = 0$ or BB_R if $\beta = 1$.
2. **Voting phase.** The adversary may make two types of queries:
   - **Vote**(v_L, v_R) queries. The adversary provides two votes $v_L, v_R \in \mathbb{V}$. The challenger runs $\mathsf{Vote}(pk, v_L)$ and $\mathsf{Vote}(pk, v_R)$ obtaining two ballots b_L and b_R respectively. $\mathcal{C}$ then obtains new versions of the boards BB_L and BB_R by running $\mathsf{ValidateBallot}(BB_L, b_L)$ and $\mathsf{ValidateBallot}(BB_R, b_R)$ and updating the boards accordingly.
   - **Ballot**(b) queries, which model queries made on behalf of corrupt voters. The adversary provides a ballot b, with which $\mathsf{ValidateBallot}(BB_L, b)$ is run by the challenger. If the algorithm returns 1, BB_L is updated and $\mathsf{ValidateBallot}(BB_R, b)$ is executed, updating BB_R accordingly. Otherwise, if the algorithm returns 0, it does nothing.
3. **Tallying phase.** The challenger evaluates $\mathsf{Tally}(sk, BB_L)$ obtaining the result r and the proof of correct tabulation Π. If $\beta = 0$, the challenger publishes (r, Π) on the bulletin board BB_L. If $\beta = 1$, the challenger runs $\mathsf{SimProof}(BB_R, r)$ obtaining a simulated proof Π' and posts (r, Π') on the bulletin board BB_R.
4. **Output.** The adversary $\mathcal{A}$ outputs a bit α, which depends on $\mathcal{A}$, $\mathbb{V}$, R, ρ and $\mathsf{SimProof}$.

We say that a voting protocol for $(\mathbb{V}, R, \rho)$ as defined in Sect. 3.1 provides ballot privacy if there exists an algorithm $\mathsf{SimProof}$ such that for any p.p.t. adversary $\mathcal{A}$ the following advantage is negligible in the security parameter λ.

$$\mathbf{Adv}^{priv}_{\mathbb{V},R,\rho,\mathsf{SimProof}}(\lambda) := |\Pr[\alpha = 1 | \beta = 1] - \Pr[\alpha = 1 | \beta = 0]|$$

We remark that honest voters are assumed to generate the ballots correctly (i.e., proper randomness is used and it is not leaked to the adversary).

3.3 Strong Consistency

In order to define *vote validatability* we will first define the notion of strong consistency. Strong consistency states that the tally of the bulletin board must correspond to the result of applying the result function to the *contents* of the ballots in the bulletin board. As shown in [4], this property is needed to avoid having leaky tallying algorithms.

In our case, we also use it to define what a meaningful content extractor is. This content extractor will be useful to define the concept of vote validatability.

Strong consistency is given by the following game, where we assume given election parameters $(\mathbb{V}, R, \rho)$ and uses an algorithm $\mathsf{Extract}(sk, b)$ which takes the election secret key and a ballot and outputs either a vote or the error symbol $\perp$ denoting an invalid vote.

1. **Setup Phase.** The challenger runs $\mathsf{Setup}(1^\lambda)$ to obtain the election public key pk and the election secret key sk. It gives both pk and sk to the adversary $\mathcal{A}$.
2. **Bulletin Board**(BB)**.** The adversary submits a bulletin board BB to the challenger.
3. **Output.** The challenger runs $\mathsf{Tally}(sk, BB)$ to obtain a result r and a correct tabulation proof Π. The output of the game is a bit γ, which depends on $\mathcal{A}$, $\mathbb{V}$, R, ρ and $\mathsf{SimProof}$. This bit is defined as 1 if $r \neq \rho(\mathsf{Extract}(sk, BB))$ and 0 otherwise, where $\mathsf{Extract}$ is applied on the bulletin board by applying it to each individual ballot.

We say that a voting protocol for $(\mathbb{V}, R, \rho)$ as defined in Sect. 3.1 has strong consistency with respect to an extract algorithm $\mathsf{Extract}$ if the following conditions are satisfied:

(i) For any (pk, sk) in the image of Setup, for any vote $v \in \mathbb{V}$ it is satisfied that $\mathsf{Extract}(\mathsf{Vote}(pk, v)) = v$

(ii) For any p.p.t. adversary $\mathcal{A}$, the following advantage is negligible in the security parameter λ:

$$\mathbf{Adv}^{s-const}_{\mathbb{V},R,\rho,\mathsf{SimProof}}(\lambda) := \Pr[\gamma = 1]$$

3.4 Vote Validatability

We now present the definition of vote validatability, which is the first contribution of this paper. Simply stated, vote validatability states that a ballot which passes all validations *must* correspond to a valid vote. This is modeled by stating that the algorithm $\mathsf{Extract}$, the one from the strong consistency property, never returns the error symbol $\perp$ on ballots for which $\mathsf{ValidateBallot}$ returns 1.

Vote validatability is given by the following game, which assumes that the election parameters $(\mathbb{V}, R, \rho)$ are given and uses an algorithm $\mathsf{Extract}(sk, b)$, which takes the election secret key and a ballot and outputs either a vote or the error symbol $\perp$ denoting an invalid vote.

1. **Setup phase.** The challenger runs $\mathsf{Setup}(1^\lambda)$ to obtain the election public and private keys (pk, sk), giving both of them to the adversary.
2. **Ballot**(b)**.** The adversary submits a ballot b to the challenger.
3. **Output.** The output of the game is a bit δ, which depends on $\mathcal{A}$, $\mathbb{V}$, R, ρ and $\mathsf{SimProof}$. This bit is defined as 1 if $\mathsf{Extract}(sk, b) = \perp$ and $\mathsf{ValidateBallot} = 1$, and as 0 otherwise.

We say that a voting protocol for $(\mathbb{V}, R, \rho)$ as defined in Sect. 3.1 has vote validatability with respect to an extract algorithm $\mathsf{Extract}$ if the following conditions are satisfied:

(i) The voting protocol for $(\mathbb{V}, R, \rho)$ is strongly consistent with respect to $\mathsf{Extract}$

(ii) For any p.p.t. adversary $\mathcal{A}$, the following advantage is negligible in the security parameter λ:

$$\mathbf{Adv}^{val}_{\mathbb{V},R,\rho,\mathsf{SimProof}}(\lambda) := \Pr[\delta = 1]$$

One implication of the definition given above is that, if the protocol has vote validatability, then it must be satisfied that, for any honestly-generated keys and any adversarially generated bulletin board, the result output by the tally can be obtained with only valid votes, $r \in \rho(\mathbb{V}^*)$.

We want to remark that vote validatability does not depend on the secrecy of the election secret key. However, it assumes that the Setup is run honestly. Even though this can be achieved by distributing the trust among multiple authorities, we have decided to give the definition assuming that there is only one authority for the sake of simplicity.

4 General Construction

4.1 Core Idea

In an electronic voting system, voters might be able to vote for more than one candidate, so we will consider a generic scenario in which votes are subsets of n distinct candidates from a larger but specified list of them. Treating the set of votes as the set of combinations of candidates would result in a terribly inefficient system. Therefore, each of the selected candidates will be encrypted independently. To prove that each candidate hidden in its respective encryption belongs to the list of candidates, we will use a set membership protocol based on digital signatures. In addition, we will use another technique to demonstrate that the candidates hidden in these encryptions are distinct.

The main idea of our new construction is inspired by the set membership protocol proposed by Camenisch et al. [6]. In that work, the authors construct a protocol for proving that a value is a commitment to a member of a pre-defined set. Their protocol works as follows. First, there is a trusted third party which produces signatures on each element of the set. Then, the prover constructs a zero-knowledge proof that she knows a signature on the committed value which verifies under the trusted third party's secret key. When the encryption scheme and the signature scheme being used have *nice* structural properties, the size of the proof is small and constant on the size of the set. In our case, the electoral authority will sign all candidates, and the voter will prove that she knows a signature on each selected candidate. However, this would still allow the voter to choose repeated candidates.

To detect this last situation, we use a technique inspired by the compact e-cash scheme given in [3]. In e-cash, detecting double-spending is essential, and this problem is similar to detecting repeated candidates in a vote. We will ask the voter to choose a pseudo-random permutation key and to publish the image of each chosen candidate under the pseudo-random permutation defined by such key. Given that the pseudo-random permutation is deterministic, if the voter chooses the same candidate more than once this will be detected by any entity. Finally, the prover needs to prove that the images of the pseudo-random permutation correspond to the candidates which she encrypted and that she knows a signature for each candidate.

In Sect. 4.2 we describe a generic protocol built on the mentioned cryptographic primitives. In general, computing non-interactive proofs of knowledge for such statements might be inefficient. In Sect. 5 we show that by instantiating the cryptographic primitives with adequate schemes the resulting protocol can be made as efficient as currently deployed e-voting systems.

4.2 Detailed Protocol

We begin by characterizing the set of allowed votes $\mathbb{V}$. Given a set of candidates $\mathcal{V}$, we define the set of votes as $\mathbb{V} = \{v \mid v \subset \mathcal{V} \wedge |v| = n\}$ for some fixed value of n. Here, we are assuming that a voter must vote for n candidates. We can handle blank votes and undervotes by designating n different blank candidates.

Our voting scheme uses a common setup generation algorithm, $\mathsf{ComSetupGen}$, in order to generate some common information that might be shared among the rest of algorithms like, for instance, the description of a mathematical group. This will be useful for efficiency reasons.

It also uses, as building blocks, an encryption scheme ($\mathsf{KeyGenEnc}$, Enc, Dec), a signature scheme ($\mathsf{KeyGenSign}$, Sign, $\mathsf{VerifySign}$), a PRP family $F_{(\cdot)}$ and a NIZK proof system (GenCRS, Prove, $\mathsf{VerifyProof}$) for the relation R defined as:

$$\begin{aligned} R = \{(x, w) |\, & x = (C_1, \ldots, \mathcal{C}_n, p_1, \ldots, p_n, pk_e, pk_s) \wedge \\ & w = (\nu_1, \ldots, \nu_n, r_1, \ldots, r_n, \sigma_1, \ldots, \sigma_n, k) \wedge \\ & (C_1, \ldots, C_n) = (\mathsf{Enc}(pk_e, \nu_1, r_1), \ldots, \mathsf{Enc}(pk_e, \nu_n, r_n)) \wedge \\ & (\mathsf{VerifySign}(pk_s, \sigma_1, \nu_1), \ldots, \mathsf{VerifySign}(pk_s, \sigma_n, \nu_n)) = (1, \ldots, 1) \wedge \\ & (p_1, \ldots, p_n) = (F_k(\nu_1), \ldots, F_k(\nu_n))\} \end{aligned}$$

The algorithms are then defined as follows:

$\mathsf{Setup}(1^\lambda)$ starts by running the $\mathsf{ComSetupGen}$ algorithm to generate the common setup information cs, which will be used by GenCRS, $\mathsf{KeyGenEnc}$ and $\mathsf{KeyGenSign}$. Then the algorithm runs GenCRS to generate the common reference string crs, $\mathsf{KeyGenEnc}$ to generate a pair of public/private encryption keys (pk_e, sk_e) and $\mathsf{KeyGenSign}$ to generate a pair of public/private signing keys (pk_s, sk_s), all of which may depend on the common setup information cs. This implicitly defines the message space for the encryption scheme M_e and the message space for the signature scheme M_s. We require that there exist two injective mappings η_1, η_2 such that $\eta_1(\mathcal{V}) \subset M_e$ and $\eta_2(\mathcal{V}) \subset M_s$. For the sake of simplicity we will assume that $\mathcal{V} = M_e = M_s$. Then, for each $\nu \in \mathcal{V}$, the algorithm produces a signature on it, $\sigma_\nu = \mathsf{Sign}(sk_s, \nu)$. The election public key is defined as $pk = (\mathsf{crs}, pk_e, pk_s, \{\sigma_\nu\}_{\nu \in \mathcal{V}})$ and the election secret key is defined as $sk = (pk, sk_e)$.

$\mathsf{Vote}(pk, v)$ parses pk as $(\mathsf{crs}, pk_e, pk_s, \{\sigma_\nu\}_{\nu \in \mathcal{V}})$ and v as $(\nu_1, \ldots, \nu_n)$. It then samples fresh randomness $(r_1, \ldots, r_n)$ and runs the $(\mathsf{Enc}(pk_e, \nu_1, r_1), \ldots, \mathsf{Enc}(pk_e, \nu_n, r_n))$ obtaining ciphertexts $C = (C_1, \ldots, C_n)$. Next, it selects a fresh random PRP key $k \in K_{PRP}$ and computes $(p_1, \ldots, p_n) = (F_k(\nu_1), \ldots, F_k(\nu_n))$ Finally, it computes a NIZK proof π for the statement $x = (C_1, \ldots, C_n,$

$p_1, \ldots, p_n, pk_e, pk_s)$ and witness $w = (\nu_1, \ldots, \nu_n, r_1, \ldots, r_n, \sigma_{\nu_1}, \ldots, \sigma_{\nu_n}, k)$. The ballot is defined as $b = (C, \pi, \{p_i\}_{i=1}^n)$.

$\mathsf{ValidateBallot}(BB, b)$ recovers pk from the bulletin board BB and parses it as $pk = (\mathsf{crs}, pk_e, pk_s, \{\sigma_\nu\}_{\nu \in \mathcal{V}})$. Upon reception of a ballot b, which parses it as $b = (C, \pi, \{p_i\}_{i=1}^n)$, it is checked if in the bulletin board there is another ballot b' such that $C'_j = C_i$ for any $i, j \in \{1, \ldots, n\}$. If any such ballot is found, the algorithm stops and returns 0. Otherwise, the algorithm checks that the values $(p_1, \ldots, p_n)$ are distinct, returning 0 if they are not. If the values are distinct, the algorithm returns the output of $\mathsf{VerifyProof}$ using the statement $x = (C_1, \ldots, C_n, p_1, \ldots, p_n, pk_e, pk_s)$.

$\mathsf{Tally}(sk, BB)$ after individual ballot b ballot has been processed with $\mathsf{ValidateBallot}$, during the tallying algorithm they are *decrypted* and the result function is computed. The *decryption procedure* is defined as follows.

1. $(\tilde{\nu_1}, \ldots, \tilde{\nu_n}) = (\mathsf{Dec}(sk_e, C_1), \ldots, \mathsf{Dec}(sk_e, C_n))$ is computed.
2. It is checked that $\tilde{\nu}_1, \ldots, \tilde{\nu}_n \in \mathcal{V}$.
3. It is checked that $(\tilde{\nu_1}, \ldots, \tilde{\nu_n})$ are pairwise different.
4. If any of such checks fail, v is assigned the value $\perp$. Otherwise, v is assigned the value $(\tilde{\nu_1}, \ldots, \tilde{\nu_n})$.

Then, ρ is applied to the resulting decryptions $\{v\}$. Note that, for each v, either $v \in \mathbb{V}$ or $v = \perp$, so ρ can be applied. The output of ρ is defined as the result and the proof of correct tabulation is defined to be the empty string ϵ

Note that, as the proof of correct tabulation is the empty string ϵ, $\mathsf{VerifyTally}$ can be the algorithm which returns 1 on any input.

Security of Our Scheme. Finally, we give the security properties fulfilled by our scheme. Let $(\mathsf{KeyGenEnc}, \mathsf{Enc}, \mathsf{Dec})$ be a NM-CPA secure encryption scheme, let $F_{(\cdot)}$ be a PRP family, let $(\mathsf{GenCRS}, \mathsf{Prove}, \mathsf{VerifyProof})$ be a NIZK proof system, and let $(\mathsf{KeyGenSign}, \mathsf{Sign}, \mathsf{VerifySign})$ be an EUF-CMA signature scheme. Let ρ be the counting function which outputs its inputs randomly permuted and let $\mathsf{Extract}$ be the *decryption procedure* of the Tally algorithm. Then, the protocol defined above (i) has ballot privacy, and (ii) has vote validatability for any $\mathbb{V}$, with respect to $\rho, \mathsf{Extract}$. These two results are formally stated in Theorems 1 and 3, which are found along with their proof in Appendix A.

5 Concrete Instantiation

We now give a concrete instantiation of the voting protocol given above. In order to give the concrete instantiation, we just need to define which encryption scheme, signature scheme, pseudo-random permutation family and non-interactive zero-knowledge proof of knowledge scheme the protocol will use. With regard to our instantiation, the candidates will be encoded as n *randomly sampled* elements of $\mathbb{G}_1$.

The $\mathsf{ComSetupGen}$ algorithm will output a type-III bilinear group as a common setup cs, i.e., a tuple $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$, where p is a prime, $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$

are groups of order p, G, H generate $\mathbb{G}_1, \mathbb{G}_2$ respectively, e is a non-degenerate bilinear map and there is no efficiently computable homomorphism from $\mathbb{G}_1$ to $\mathbb{G}_2$ or viceversa. Besides, the Decisional Diffie-Hellman assumption [18] holds in $\mathbb{G}_1$ and in $\mathbb{G}_2$.

Encryption Scheme. The protocol will use the Signed ElGamal [17] encryption scheme in $\mathbb{G}_1$, which is NM-CPA secure [5].

Signature Scheme. The signature scheme that we will use is the structure-preserving signature scheme given in [1]. A structure-preserving signature scheme is characterized by having messages, signatures and verification keys to be group element and having a verification procedure that only consists on evaluating product-pairing equations.

The signature works as follows. On a common setup $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$, an extra random element $X \in \mathbb{G}_1$ is added to the public parameters. The secret key is a value $v \in \mathbb{Z}_p$ and the public key is computed as $V = H^v$. The signature on $M \in \mathbb{G}_1$ is then $(R, S, T) = (H^r, M^{\frac{v}{r}}, S^{\frac{v}{r}} G^{\frac{1}{r}})$ for a random r. To verify a signature it is checked if $e(S, R) = e(M, V)e(X, H)$ and $e(T, R) = e(S, V)e(G, H)$.

Pseudo-Random Permutation Family. We will define the set of candidates, $\mathcal{V}$, as a set containing n randomly sampled group elements from $\mathbb{G}_1$. This allows us to define the pseudo-random function $F_k : \mathbb{G}_1 \to \mathbb{G}_1$ where $F_k(g) = g^k$ and $k \in \mathbb{Z}_p^*$. As we assume that the Decisional Diffie-Hellman assumption holds in $\mathbb{G}_1$, this function family is pseudo-random when we restrict the input to $\mathcal{V}$.[3]

Non-Interactive Zero-Knowledge Proof of Knowledge. Finally, we have to give the NIZKPK scheme that we will use. We will use the Groth-Sahai Proof System [14] but we will frame it as a Commit-and-Prove scheme as done in [10]. A Commit-and-Prove scheme is similar to a NIZKPK scheme with the difference that a Commit-and-Prove scheme explicitly splits the process of committing to secret values and proving statements related to such values. In addition, [10] introduces *type-based* commitments, where the type indicates how the commitment should be computed. For example, the type "encryption" indicates that the secret value should be encrypted, as opposed to using the more expensive commitment operation.

We first remark that the encryption and signature schemes must use the same algebraic groups that the NIZKPK scheme. Therefore, at the beginning of the Setup algorithm, the common setup $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$ is generated and will be used to generate the crs and the keys of the encryption and signature schemes. There is no loss of generality since the groups are generated in the same way in ComSetupGen and, respectively, in GenCRS, KeyGenEnc and KeyGenSign.

In addition, the Commit-and-Prove scheme given in [10] allows us to treat the ElGamal encryption of a value as a commitment of type "encryption", where the

[3] Technically, it is a Pseudo-Random Function [12] from $\mathcal{V}$ to $\mathbb{G}_1$ where $F_{(\cdot)}$ is injective for any $k \in \mathbb{Z}_p^*$. Therefore, an adversary restricted to only evaluate the function in points from $\mathcal{V}$ can not distinguish those evaluations from randomly sampled elements, which is sufficient for the security reduction to work.

randomness used for the encryption is the randomness used for the commitment. The encryption scheme is thus embedded into the NIZKPK scheme, instead of being an independent scheme, as assumed in the general construction. However, we can still adapt the security proof to keep it sound as we now describe.

We will consider the conjunction of two proofs. The first one is a zero-knowledge proof for the language defined by the relation

$$\begin{aligned} R_1 = \{(x,w) |\, & x = (C_1,\ldots,\mathcal{C}_n, pk_e, pk_s) \wedge \\ & w = (\nu_1,\ldots,\nu_n, r_1,\ldots,r_n, \sigma_1,\ldots,\sigma_n) \wedge \\ & (C_1,\ldots,C_n) = (\mathsf{Enc}(pk_e,\nu_1,r_1),\ldots,\mathsf{Enc}(pk_e,\nu_n,r_n)) \wedge \\ & (\mathsf{VerifySign}(pk_s,\sigma_1,\nu_1),\ldots,\mathsf{VerifySign}(pk_s,\sigma_n,\nu_n)) = (1,\ldots,1)\} \end{aligned}$$

For this proof, the prover computes a commitment to each value of the signature and builds proofs for satisfiability of the verification equations.

We now need to see that the Commit-and-Prove scheme in [10] is Zero-Knowledge for the language defined by R_1. In other words, we need to see that exists a simulator. As seen in [10], this reduces to check that there are no terms in pairing product equations which prevent simulation. Those terms are pairings where in each side of the pairing there is either a public, non-equivocable value[4] or a value which commitment type is "encryption". Going back to the verification equations of the signature scheme, we see that there are none of these terms. Therefore, there exists a simulator for the statement defined by R_1.

The second zero-knowledge proof is defined by the relation

$$\begin{aligned} R_2 = \{(x,w) |\, & x = (C_1,\ldots,\mathcal{C}_n, p_1,\ldots,p_n, pk_e) \wedge \\ & w = (\nu_1,\ldots,\nu_n, r_1,\ldots,r_n, k) \wedge \\ & (C_1,\ldots,C_n) = (\mathsf{Enc}(pk_e,\nu_1,r_1),\ldots,\mathsf{Enc}(pk_e,\nu_n,r_n)) \wedge \\ & (p_1,\ldots,p_n) = (F_k(\nu_1),\ldots,F_k(\nu_n))\} \end{aligned}$$

For the proof for the relation R_2, we will consider the multi-exponentiation equations $\nu_i^k = p_i$, where ν_i and k are secret values. The prover computes a commitment on k and builds a proof for satisfiability of this equation using C_i as a commitment to ν_i. Both the commitments on the signatures and the proofs will be included in π. As noted in [10], multi-exponentiation equations are always simulatable.

Note that Groth-Sahai proofs are not extractable for exponents such as k. However, the proof of our scheme having vote validatability only needs to extract the values $\nu_1,\ldots,\nu_n$ and their corresponding signatures.

5.1 Efficiency

Each Signed ElGamal encryption consists of 2 elements in $\mathbb{G}_1$ and 2 elements in $\mathbb{Z}_p$. Each value p_i consists of a single element in $\mathbb{G}_1$.

[4] In [10] the authors define equivocable values as the generators of the group. However, it can be seen that values for which the simulator knows the discrete logarithm w.r.t. the generator of the group are also equivocable.

When looking at the proof π, we have to consider both the proof for the language defined by R_1 and the language defined for R_2. For the proof for the language defined by R_1, we have that in the structure-preserving signature of [1] each signature consists of three elements. A Groth-Sahai commitment on a single element consists on 2 elements in $\mathbb{G}_1$. As we have to commit to n signatures, the number of elements is $6n$ elements in $\mathbb{G}_1$. Furthermore, there are two verification equations per signature and a Groth-Sahai proof for a single of such equation consists of 4 elements in $\mathbb{G}_1$ and 4 elements in $\mathbb{G}_2$. Therefore, the proofs for all the verification equations consist on $8n$ elements in $\mathbb{G}_1$ and $8n$ elements in $\mathbb{G}_2$.

When considering the proof for the language defined by R_2, we have to commit to k, which has a cost of 2 elements in $\mathbb{G}_2$, and compute the Groth-Sahai proofs. A Groth-Sahai proof for an equation of the form $\nu^k = p$ consists on 2 elements in $\mathbb{G}_1$ and 4 elements in $\mathbb{G}_2$, and we have to compute n of them.

In total, we get that a ballot consists of $19n$ elements in $\mathbb{G}_1$, $12n+2$ elements in $\mathbb{G}_2$ and $2n$ elements in $\mathbb{Z}_p$. The cost is linear in n (the number of candidates encoded in each vote). Moreover, the constant factor is relatively small.

6 Conclusions

We have formalized the definition of vote validatability in order to give an accurate meaning to *avoid voters from casting invalid votes*, both if done in purpose or as a consequence of a software bug. Besides creating a construction based on generic building blocks and general-purpose zero-knowledge proofs, we have provided a concrete instantiation. We have shown that its efficiency fits into the device's computational capacity of voters in current elections.

There are other alternatives which may improve the performance of our construction achieving the same security properties. First, a cryptographic accumulator could be used to prove that candidates are valid. This approach could reduce the length of the ballot but would make the scheme to rely on the Random Oracle Model. Second, much of the cost of the ballot comes from the NIZKPK proof for the language defined by R_1. A choice of a different structure-preserving signature scheme might improve the efficiency of our system.

A Proofs of Security Theorems

We prove the security for the construction given in Sect. 4.2.

Theorem 1. *Let* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a NM-CPA secure encryption scheme, let* $F_{(\cdot)}$ *be a PRP family and let* $(\mathsf{GenCRS}, \mathsf{Prove}, \mathsf{VerifyProof})$ *be a NIZK proof system. Then, the protocol defined in Sect. 4.2 has ballot privacy.*

Proof. Recall that privacy is defined as the indistinguishability of two experiments which depend on a bit β. We will refer to them as Exp_β for $\beta \in \{0, 1\}$.

Let $\mathsf{SimVote}_1(pk, v)$ be the Vote algorithm of the protocol given in Sect. 4.2 but, instead of using the Prove algorithm to generate π it uses the SimProve algorithm. Moreover, let $\mathsf{SimVote}_2(pk, v)$ to be the $\mathsf{SimVote}_1$ algorithm but, instead of using a PRP it uses a truly random permutation.

Consider experiments $\mathsf{Exp}_{\beta,0} = \mathsf{Exp}_{\beta}$, $\mathsf{Exp}_{\beta,1}$ to be the experiment which are the same as $\mathsf{Exp}_{\beta,0}$ but the challenger runs SimGenCRS instead of GenCRS and it runs SimProve instead of Prove. Finally, let $\mathsf{Exp}_{\beta,2}$ be the experiments which are identical to $\mathsf{Exp}_{\beta,1}$ but in which the challenger uses a truly random function instead of a PRP in order to cast ballots.

Due to the zero-knowledge property of the NIZK proof system, $\mathsf{Exp}_{\beta,0}$ and $\mathsf{Exp}_{\beta,1}$ are indistinguishable for $\beta \in \{0, 1\}$. Besides, $\mathsf{Exp}_{\beta,1}$ and $\mathsf{Exp}_{\beta,2}$ are indistinguishable for $\beta \in \{0, 1\}$ due to the pseudo-randomness of the PRP. Now the only thing left is to prove that $\mathsf{Exp}_{0,2}$ and $\mathsf{Exp}_{1,2}$ are indistinguishable.

Consider the Enc2Vote scheme [5], where the result function ρ is the multiset function. The scheme is defined as follows: the Setup algorithm runs KeyGenEnc to produce a public key pk_e and a secret key sk_e. Then, pk is set to be pk_e and sk is set to be (pk_e, sk_e). The Vote algorithm takes as input a vote v and a public key pk_e and outputs b defined by $b = \mathsf{Enc}(pk_e, v, r)$ for some fresh randomness r. ValidateBallot looks if the ballot b already appears on the bulletin board BB: it returns 1 if it does already appear and 0 otherwise. Tally decrypts all ballots $\boldsymbol{b}$ on the bulletin board obtaining votes $\boldsymbol{v}$ and evaluates $r = \rho(\boldsymbol{v})$, outputting an empty proof of correct tabulation. Observe that Enc2Vote implicitly assumes that $\mathbb{V} = M_e$, the message space of the encryption scheme. As shown in [5], the following is satisfied:

Theorem 2. *Let* $(\mathsf{KeyGenEnc}, \mathsf{Enc}, \mathsf{Dec})$ *be an NM-CPA secure encryption scheme. Then,* Enc2Vote *has ballot privacy.*

Finally, we reduce the privacy of our scheme to the privacy of Enc2Vote.

Lemma 1. *Let* $\mathcal{A}^1$ *be a p.p.t. adversary that interacts which challenger* $\mathcal{C}$ *and outputs a bit* $\alpha^{\mathcal{A}_1}$ *such that* $|\Pr[\alpha^{\mathcal{A}_1} = 1|\mathsf{Exp}_{0,2}] - \Pr[\alpha^{\mathcal{A}_1} = 1|\mathsf{Exp}_{1,2}]|$ *is non-negligible. Then, there exists an adversary* $\mathcal{A}^2$ *that breaks the ballot privacy property of the* Enc2Vote *scheme.*

In our reduction, $\mathcal{A}^1$ will interact with $\mathcal{A}^2$, which will act as the challenger for $\mathcal{A}^1$. At the same time, $\mathcal{A}^2$ will interact with the privacy challenger $\mathcal{C}$. The reduction is as follows:

In the **Setup** phase, $\mathcal{C}$ will run ComSetupGen, outputing cs and posting it to the bulletin board. It will also run KeyGenEnc, keeping the private key for itself and publishing the public key pk_e to the bulletin board. Then, A^2 will run the GenCRS and the KeyGenSign algorithms and will produce signatures on each voting option, posting all the information to the bulletin board.

In the **Voting** phase, when $\mathcal{A}^1$ submits a **Vote** query, A^2 will submit n **Vote** queries to $\mathcal{C}$, one for each pair of candidates. The challenger $\mathcal{C}$ will answer with n pairs of ciphertexts $(C_{0,1}, \ldots, C_{0,n})$ and $(C_{1,1}, \ldots, C_{1,n})$. A^2 will then sample two pairs of random values $(p_{0,1}, \ldots, p_{0,n})$ and $(p_{1,1}, \ldots, p_{1,n})$ of the target space

of the PRP. Finally, it will create ballots $b_0 = (C_{0,1}, \ldots, C_{0,n}, p_{0,1}, \ldots, p_{0,n}, \pi_0)$ and $b_1 = (C_{1,1}, \ldots, C_{1,n}, p_{1,1}, \ldots, p_{1,n}, \pi_1)$ where π_0 and π_1 will be simulated. $\mathcal{A}^2$ will post these ballots to the respective bulletin boards. Finally, when $\mathcal{A}^1$ submits a $\mathbf{Ballot}(b)$ query, $\mathcal{A}^2$ will run the $\mathsf{ValidateBallot}$ algorithm and will create a $\mathbf{Ballot}(b')$ for $\mathcal{C}$ with $b' = (C_1, \ldots, C_n)$ from b.

It is straightforward to see that the output of $\mathcal{A}^2$ in its interaction with $\mathcal{A}^1$ is correctly distributed, which implies that the reduction is sound.

Theorem 3. *Let ρ be the counting function which outputs its inputs randomly permuted. Let* $(\mathsf{GenCRS}, \mathsf{Prove}, \mathsf{VerifyProof})$ *be a NIZKPK proof system and let* $(\mathsf{KeyGenSign}, \mathsf{Sign}, \mathsf{VerifySign})$ *be an EUF-CMA signature scheme. Let* $\mathsf{Extract}$ *be the* decryption procedure *of the* Tally *algorithm of the protocol defined in Sect. 4.2. Then, the protocol defined in Sect. 4.2 has vote validatability for any $\mathbb{V}$, with respect to $\rho, \mathsf{Extract}$.*

Proof. Strong consistency of the protocol follows by construction. Therefore we only need to show that, on correctly generated (pk, sk) no adversary can construct a ballot b such that $\mathsf{ValidateBallot}$ returns 1 but $\mathsf{Extract}$ returns $\perp$.

Let Exp_0 be the vote validatability experiment and let Exp_1 be identical to Exp_0 but instead of using GenCRS the challenger uses $\mathsf{ExtGenCRS}$. These two experiments are indistinguishable by the properties of the NIZKPK. Now assume that an adversary $\mathcal{A}^1$ is able to output a ballot b in the experiment Exp_1 such that $\mathsf{ValidateBallot} = 1$ and $\mathsf{Extract}(sk, b) = \perp$. Then, we build an adversary $\mathcal{A}^2$ which breaks the EUF-CMA of the signature scheme.

The reduction is straightforward: $\mathcal{A}^2$, interacting with an EUF-CMA challenger asks for signatures on $\{\nu\}_{\nu \in \mathcal{V}}$. Then, it interacts with $\mathcal{A}^1$, posing as a vote validatability challenger. It runs all the algorithms as in the protocol but uses $\mathsf{ExtGenCRS}$, keeping the trapdoor key tk for itself, and using the answers from the EUF-CMA challenger as the signatures on the voting options. When $\mathcal{A}^1$ outputs a ballot b, $\mathcal{A}^2$ uses $\mathsf{Extract}$ on π to obtain a witness $w = (\tilde{\nu}_1, \ldots, \tilde{\nu}_n, r_1, \ldots, r_n, \sigma_{\tilde{\nu}_1}, \ldots, \sigma_{\tilde{\nu}_n}, k))$ such that $(x, w) \in R$. This means that $\mathsf{VerifySign}(pk_s, \sigma_{\tilde{\nu}_i}, \tilde{\nu}_i) = 1$ for $i \in \{1, \ldots, n\}$. $\mathsf{Extract}(sk, b)$ might return $\perp$ either because (i) some $\mathsf{Dec}(sk_e, C_i) = \perp$, (ii) some $\tilde{\nu}_i = \tilde{\nu}_j$ for $i \neq j$ or (iii) some $\tilde{\nu}_i \notin \mathcal{V}$. However, (i) and (ii) are ruled out due to w being a valid witness, so the only possibility is (iii). Then, $\mathcal{A}^2$ can submit $(\tilde{\nu}_i, \sigma_{\tilde{\nu}_i})$ as its EUF-CMA forgery.

References

1. Abe, M., Groth, J., Ohkubo, M., Tibouchi, M.: Unified, minimal and selectively randomizable structure-preserving signatures. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 688–712. Springer, Heidelberg (2014)
2. Adida, B.: Encrypting your vote in javascript. Electronic Voting Technology Workshop - EVT/WOTE, August 2011. http://assets.adida.net/presentations/2011-08-08-helios-evt-rump.pdf
3. Belenkiy, M., Chase, M., Kohlweiss, M., Lysyanskaya, A.: Compact E-Cash and simulatable VRFs revisited. In: Shacham, H., Waters, B. (eds.) Pairing 2009. LNCS, vol. 5671, pp. 114–131. Springer, Heidelberg (2009)

4. Bernhard, D., Cortier, V., Galindo, D., Pereira, O., Warinschi, B.: A comprehensive analysis of game-based ballot privacy definitions. IACR Cryptology ePrint Archive 2015, 255 (2015). http://eprint.iacr.org/2015/255
5. Bernhard, D., Pereira, O., Warinschi, B.: On necessary and sufficient conditions for private ballot submission. IACR Cryptology ePrint Archive 2012, 236 (2012). http://dblp.uni-trier.de/db/journals/iacr/iacr2012.html#BernhardPW12. Informal Publication
6. Camenisch, J.L., Chaabouni, R., shelat, a: Efficient protocols for set membership and range proofs. In: Pieprzyk, J. (ed.) ASIACRYPT 2008. LNCS, vol. 5350, pp. 234–252. Springer, Heidelberg (2008)
7. Chaum, D.: Untraceable electronic mail, return addresses and digital pseudonyms. In: Gritzalis, D. (ed.) Secure Electronic Voting, Advances in Information Security, vol. 7, pp. 211–219. Springer, New York (2003). http://dx.doi.org/10.1007/978-1-4615-0239-5_14
8. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. Eur. Trans. Telecommun. **8**(5), 481–490 (1997). http://dx.doi.org/10.1002/ett.4460080506
9. Damgård, I., Jurik, M., Nielsen, J.B.: A generalization of Paillier's public-key system with applications to electronic voting. Int. J. Inf. Secur. **9**(6), 371–385 (2010)
10. Escala, A., Groth, J.: Fine-tuning Groth-Sahai proofs. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 630–649. Springer, Heidelberg (2014)
11. Gjøsteen, K.: Analysis of an internet voting protocol. Cryptology ePrint Archive, Report 2010/380 (2010). http://eprint.iacr.org/
12. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. J. ACM **33**(4), 792–807 (1986). http://doi.acm.org/10.1145/6490.6503
13. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. **17**(2), 281–308 (1988). http://dx.doi.org/10.1137/0217017
14. Groth, J., Sahai, A.: Efficient noninteractive proof systems for bilinear groups. SIAM J. Comput. **41**(5), 1193–1232 (2012)
15. Luby, M., Rackoff, C.: How to construct pseudo-random permutations from Pseudo-random functions. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 447–447. Springer, Heidelberg (1986)
16. Santis, A.D., Persiano, G.: Zero-knowledge proofs of knowledge without interaction (extended abstract). In: FOCS, pp. 427–436. IEEE Computer Society (1992)
17. Schnorr, C.-P., Jakobsson, M.: Security of signed ElGamal encryption. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, p. 73. Springer, Heidelberg (2000)
18. Tsiounis, Y., Yung, M.: On the security of ElGamal based encryption. In: Imai, H., Zheng, Y. (eds.) PKC 1998. LNCS, vol. 1431, p. 117. Springer, Heidelberg (1998)