Universal Cast-as-Intended Verifiability

Alex Escala[1], Sandra Guasch[2], Javier Herranz[1], Paz Morillo[1]

Universitat Politècnica de Catalunya[1], Scytl Secure Electronic Voting[2]
{alex.escala,javier.herranz,paz.morillo}@upc.edu,
sandra.guasch@scytl.com

Abstract. In electronic voting, we say that a protocol has cast-as-intended verifiability if the contents of each encrypted vote can be audited in order to ensure that they match the voter's selections. It is traditionally thought that this verification can only be performed by the voter who casts the vote, since only she knows the content of her vote. In this work, we show that this is not the case: we present the first cast-as-intended verification mechanism which is universally verifiable, i.e., the first protocol in which anyone (the voter herself or another party) can check that the contents of an encrypted vote match the voter's selections. To achieve this goal, we assume the existence of a trusted registrar. We formally define universal cast-as-intended verifiability and we show that our protocol satisfies such property, while also satisfying ballot privacy. We give a general construction of the protocol and an efficient instantiation which is provably secure in the random oracle model. We also present a voting system which can be implemented on top of the voting protocol, which is intended to present a more intuitive process to the voter.

1 Introduction

Traditionally, verifiability properties of electronic voting are divided in two categories, depending on the entities who can make use of them. Individual verifiability involves auditing the processes of vote creation and vote storage by the voter. Universal verifiability consists on auditing that only votes from eligible voters are stored in the ballot box, and that all stored votes are properly tallied, which can be performed by everyone. Systems providing both types of verifiability are known as *end-to-end verifiable* systems [3].

One of the individual verifiability properties is vote casting assurance [3], also known as cast-as-intended verification, which is focused on the audit of the vote creation process. This property is only meaningful when the vote is cast using a voting device, therefore excluding voting systems where selections are done on a physical ballot. Another property is recorded-as-cast verification, aimed at auditing the correct reception and storage of the vote in a remote voting server.

In this paper, we focus on cast-as-intended verification, which we present here using a simplified electronic voting scenario. In this simplified voting system, the voter uses a voting device in order to select her choices. After she confirms her vote, the voting device encrypts the selections using a public key encryption

scheme. The encrypted vote is then stored in a (local or remote) ballot box. When the voting period finishes, the votes in the ballot box are decrypted in order to obtain the election results. By encrypting the vote at the voting device, the secrecy of the vote is ensured during its transmission and storage in the ballot box, until the time of decrypting and tallying the votes.

Naturally, neither the voter or any other entity is able to check if the content of the encrypted vote that is cast really matches what the voter selected. This implies that a malicious voting device would be able to encrypt other options than those selected by the voter. Cast-as-intended verification mechanisms provide the system with means to audit the content of the encrypted vote in order to detect this kind of attacks.

Traditionally, the electronic voting community has considered that the cast-as-intended verification can only be performed by the voter who cast the vote, given that she made her selections in secret. However, leaving the responsibility of this verification to the voter may not be effective at all. On the one hand, we might think of systems which make the cast-as-intended verification mandatory in order to cast a vote. The problem is that current cast-as-intended verification systems present important drawbacks: verification mechanisms are usually not very usable and in most cases voters have to engage in highly interactive protocols and/or be able to perform complex computations. Therefore, mandatory cast-as-intended verification would disenfranchise less skilled voters. On the other hand, we could allow the cast-as-intended verification to be an optional step which the voter can do before casting the vote. This does not solve the problem though, as targeted attacks against non-skilled voters, who will probably not use the verification system, can succeed undetectably.

Our goal is to propose an alternative paradigm which solves the above-mentioned problem, by allowing any party (not only the voter) to publicly verify that an encrypted cast vote really matches the selection of a voter. We refer to this property as universal cast-as-intended verifiability.

1.1 Our contributions

In this paper, we present a universal cast-as-intended verification protocol, i.e., a protocol where cast-as-intended verification is not restricted to the voter anymore. This allows to deploy mechanisms for auditing all the cast votes in an extensive form, and ensure no tampering attacks happen. Once the voter has cast a ballot, our protocol does not require her to take any further step to verify that her cast ballot corresponds to her chosen candidates.

This is a change of paradigm with respect to current voting protocols, as in our new protocol the verification of the content of the encrypted votes does not rely on the willingness or skills of the voters. Indeed, it can now rely on third parties (e.g. election auditors) without compromising voters' privacy.

The main idea of our protocol is the following: a voter registers to vote with a registrar, who generates a pair of public-secret values for each voting option in the election. The secret values are sent to the voter, while the public ones are

published, linked to the voting options they are related to. We need to assume that the registrar is trusted and follows these instructions correctly.

During the voting phase, the voter provides her selected voting options and a subset of the secret values she received during registration to the voting device. The voting device then encrypts the voter's selections and creates a non-interactive zero-knowledge proof of knowledge (NIZKPK), which will be valid only in the case the voting device encrypted what the voter selected.

Thanks to the zero-knowledge property of the proofs, they can be publicly verified while maintaining the voter's privacy. Therefore, we can say that the system provides *universal cast as intended verifiability.*

1.2 Related work

Helios [1] is a well-known electronic voting protocol which has been used in several real-world binding elections. Both Helios and our new voting protocol are ballot private (even though the ballot privacy definition needs to be adapted for considering voter-private information) and non-receipt free. However, our new protocol has universal cast-as-intended verifiability, whereas Helios only considers individual cast-as-intended verifiability.

At a first glance one would say that our voting protocol is similar to any Code Voting protocol [7]: in both systems voters introduce codes in their voting devices. However, there are important differences between our voting protocol and any Code Voting protocol. Unlike Code Voting systems, privacy of our voting protocol does not depend on the secrecy of the voting codes. In addition, not only our voting protocol is (universally) cast-as-intended verifiable but it can be extended to be end-to-end verifiable by using known techniques such as a Bulletin Board [16] and a verifiable Mix-Net [22]. As far as we know, the only end-to-end verifiable Code Voting protocols (with individual cast-as-intended verifiability) are VeryVote [18], in which verifiability is achieved at the cost of making the scheme non-private to the voting server, and Pretty Good Democracy [21], in which verifiability implies knowledge and use of the election private key by a set of trustees during the voting phase. Finally, note that our voting protocol does not attempt to solve the secure platform problem [15] and, in particular, it can be implemented with a user-friendly point and click voting interface (even though voters still need to introduce voting codes).

Other systems which provide cast-as-intended verifiability while trying to reduce the participation of the voter in the process, such as Scratch & Vote [2], MarkPledge [19], Chaum's *Secret-Ballot Receipts* [8] and *Prêt à voter* [20] are focused on poll-site voting systems, and require specific hardware and procedures, such as optical scanners or DREs with printers, which are not available to remote voters.

Finally, the Eperio cryptographic election verification protocol [13] can be considered a precursor of universal cast-as-intended verifiability. In this protocol, voters might delegate their individual verifiability, without any privacy loss, by giving a copy of their voting receipt to any other party. However, Eperio is

designed to be used in traditional elections in which only the tallying process is done electronically.

1.3 Structure of the paper

We give the syntax of a voting system which enables such universal cast-as-intended property in Section 2.1, and in Section 2.2 we informally state the security assumptions we make, and provide a definition for universal cast-as-intended verifiability. The building blocks of our protocol are presented in Section 3. In Section 4 we give a generic construction of our new protocol. We provide the results of our security analysis (included in the full version of this paper) in Section 5. In addition, we give an efficient instantiation of our new protocol which is provably secure in the random oracle model in Section 6. In Section 7 we present an implementation of our new voting protocol which might lead to the design of usable voting systems. Finally, we detail some lines of future research in Section 8.

2 Electronic voting definitions

2.1 Syntactical definition

We now give the syntax of a voting scheme, which we will later use to present our protocol. This definition is based on single-pass voting schemes, as defined in [5]. These kind of schemes are characterized by the fact that voters interact with the system only by submitting their ballots. Voters may have credentials in order to be able to cast such ballots, but how voters get them is outside the scope of this scheme.

As defined in [5], a voting scheme has the following participants: the *Election Authorities* are in charge of setting up the election, computing the tally and publishing the results; the *Voters* participate in the election by choosing their preferred options; and the *Bulletin Board Manager* receives, processes and publishes the ballots received, as well as other public information.

In addition, we also consider the following participants: the *Registrars* are responsible for providing to the voters that chose them all the information they need to vote, in particular the information that will provide the UCIV property. We also explicitly consider the *Voting Device* as a participant. As opposed to *Voters*, who only choose their preferred voting options, the *Voting Device* is in charge of casting a ballot given those voting options. This distinction is important when introducing the concept of universal cast-as-intended verifiability.

We assume that non-cryptographic election specifications such as the set of voting options V or the set of voters are fixed in advance by the *Election Authorities*. Further we assume a counting function $\rho : (V \cup \{\perp\})^* \to R$ is given, where V is the set of voting options, $\perp$ denotes an invalid vote and R is the set of results.

For sake of simplicity, we will assume that there is only one Election Authority and one Registrar. Detailed trust assumptions for Election Authorities and Registrars are further discussed in Section 2.2.

The voting scheme is characterized by the following algorithms:

- $\mathsf{Setup}(1^\lambda)$ is a protocol executed by the Election Authority. On input a security parameter 1^λ, it generates and outputs an election public/private key pair (epk, esk). In addition, it defines the space of secret Universal Cast-As-Intended Verification (UCIV from now on) information SVI, the space of voting option-dependent secret UCIV information $\widetilde{SVI}$, the space of public UCIV information PVI and a family of functions $\sigma_\cdot : SVI \to \widetilde{SVI}$ such that, for each voting option $v \in V$ the function σ_v maps the secret UCIV information to some voting option-dependant secret UCIV information (*The function σ_v will need to be evaluated by the voter, so it should be a relatively simple one. In our scheme the function will consist on selecting some elements from a given set*).
- $\mathsf{Register}(vid, V, epk)$ is run by the Registrar. It takes as input a voter identity vid, the set of voting options V and the election public key epk. It outputs the secret UCIV information $s^{vid}_{uciv} \in SVI$ and the public UCIV information $p^{vid}_{uciv} \in PVI$.

- $\mathsf{Vote}(vid, v, \sigma_v(s^{vid}_{uciv}), p^{vid}_{uciv}, epk)$ is a probabilistic protocol run by voting devices. It receives as input the voter identity vid, a voting option $v \in V$, the function σ_v evaluated on the secret UCIV information $s^{vid}_{uciv} \in SVI$, the public UCIV information $p^{vid}_{uciv} \in PVI$, the election public key epk and outputs a ballot b.
- $\mathsf{ProcessBallot}(BB, b, vid)$ is run by the bulletin board manager. It receives as input a bulletin board BB, a ballot b and a voter identity vid and outputs either success (1) or reject (0).
- $\mathsf{Tally}(BB, esk)$ is run by the Election Authority. It takes as input a bulletin board BB and the election secret key esk and outputs a result $r \in R$ and a correct tabulation proof Π.

Finally, the scheme is executed as follows:

Configuration phase: in this phase, the Election Authority runs the Setup algorithm. The Election Authority publishes the election public key epk on the bulletin board BB and keeps the election secret key esk.

Registration phase: in this phase, a voter registers to vote in the election. The Registrar runs the $\mathsf{Register}$ algorithm, provides the secret UCIV information s^{vid}_{uciv} to the voter, and publishes the public UCIV information p^{vid}_{uciv} in the bulletin board.

Voting phase: in this phase each registered voter vid can vote. To vote, the voter chooses a voting option v, evaluates $\sigma_v(s^{vid}_{uciv})$ and provides $(vid, v, \sigma_v(s^{vid}_{uciv}))$ to the voting device. The voting device takes the election public key epk and the public UCIV information p^{vid}_{uciv} from the bulletin board and runs the Vote algorithm, producing a ballot b. The ballot b and the voter's identity vid are

then sent to the bulletin board. Upon reception of the ballot, the bulletin board manager executes the ProcessBallot algorithm. In case the output is 1, the ballot is published in the bulletin board, otherwise the ballot is discarded and the voter is notified accordingly.

Counting phase: in the counting phase the Election Authority runs the Tally algorithm using the election private key esk and the information in the bulletin board, including the ballots. The output of the Tally algorithm is published on the bulletin board.

A voting system as defined above is correct if, when the four phases are run with all the participants behaving correctly, the result r output by the Tally algorithm is equal to the evaluation of the counting function ρ on the voting options corresponding to the ballots cast by the voters.

2.2 Security definitions

In this section we give the security definitions of privacy and universal cast-as-intended. We will not define recorded-as-cast verifiability nor counted-as-recorded verifiability, since these properties are out of the scope of this paper. Similarly, we focus on the notions of privacy provided by existing schemes such as [1], and leave the consideration of stronger notions of vote privacy, such as receipt freeness or coercion resistance, for a future work.

Assumptions on voting devices, election authorities and registrars: When considering voting devices, we distinguish between privacy and integrity. In order to guarantee privacy, we assume that the voting device behaves properly by correctly encrypting the voter's choice and not leaking any information. This is a common assumption in electronic voting schemes where the voter can use a point and click interface to make her choices, such as Helios [1]. On the other hand, we do not trust the voting device at all when it comes to integrity, since we assume it could try to change the voters' choice prior to encryption. In the same way, the verifiability of the scheme does not rely on the voting device being honest.

We will assume that there is only one election authority, which is trusted for privacy. Secret sharing and multi-party computation techniques can be easily deployed, as done in [11], to overcome this limitation so that privacy is guaranteed as long as a subset of the election authorities are trusted. The electoral authorities do not need to be trusted for our new verifiability property. Indeed, in the UCIV security definition we assume that the election private key is leaked to the adversary.

In addition, we will also assume that any common reference strings are generated by a single, trusted authority. This is a strong assumption, which can be solved by using the multi-string model defined by Groth and Ostrovsky in [17]. In this model, several parties provide their common reference strings and security relies on assuming that a threshold of such parties are honest. We consider that directly using the multi-string model would make our paper utterly complex

and, on the other hand, it would not add much value to the paper. Moreover, when we instantiate our scheme with protocols with ROM-based security, no common reference string is needed at all.

We also assume that there is only one registrar, who is trusted to produce the UCIV information correctly and keep it secret, so that the secret UCIV information is not leaked to the adversary. This assumption is common in other protocols for the information generated by the registrars for providing individual verifiability properties, such as [21] or [2]. There are several well-known techniques that can be used to weaken these assumptions. One is to randomly audit a set of outputs produced by the registrar in order to ensure the UCIV information is produced correctly. This audit involves the publication of the secret UCIV information, and therefore an audited set of UCIV information should not be used by a voter to cast her vote. Another one is to consider a set of registrars and use multi-party computation techniques in order to guarantee the privacy of the UCIV information, as far as a subset of the registrars are honest. The printer which may be in charge of putting the shares of secret UCIV information together and providing them to the voter is then trusted not to reveal this information, as in [21]. Finally, another approach is to allow each voter to choose which registrars she registers with: she might register with only one registrar or with all of them. These extensions will be detailed in the full version of the paper.

Ballot privacy Intuitively, a voting system has ballot privacy if an adversary with access to the bulletin board is not able to guess what voting options the voters chose. We adopt the formalization given in [4], where they give a definition correcting the flaws of previous definitions.

Ballot privacy is defined using two experiments between an adversary A and a challenger C. The goal of the adversary is to disinguish between the two experiments. In both experiments, we let the adversary corrupt voters and submit ballots on their behalf. In addition, for each honest voter the adversary can also specify two votes to be used for casting her ballot. The votes which will be used to cast the honest voters' ballots will depend on which experiment is taking place. The goal of the adversary is to distinguish between both experiments i.e., to distinguish which votes were used to cast the honest voters' ballots. As revealing the "true" tally would easily allow the adversary to distinguish between the experiments, the same tally is always shown to the adversary, regardless of which vote was used to cast honest voters' ballots.

For compactness, we present the two experiments as a single experiment depending on a bit β. The experiments are parametrized by the set of voting options V and an algorithm $\mathsf{SimProof}(BB, r)$ such that, given a bulletin board and a result simulates a proof of correct tabulation.

1. **Setup phase**. The challenger sets up two empty bulletin boards L and R. It runs the $\mathsf{Setup}(1^\lambda)$ protocol to obtain the election public key epk and the election private key esk. It then posts epk on both bulletin boards. The

adversary is given read access to either L if $\beta = 0$ or R if $\beta = 1$. In addition, C initializes an empty list ID.

2. **Registration phase**. The adversary may make one type of query.
   - **Register**(vid) query. The adversary provides a voter identity such that $(vid) \notin ID$. The challenger runs $\mathsf{Register}$ on inputs (vid, V, epk) to generate the partial public UCIV information p_{uciv}^{vid} and the partial secret UCIV information s_{uciv}^{vid}. C provides both p_{uciv}^{vid} and s_{uciv}^{vid} to A and p_{uciv}^{vid} is published on both bulletin boards. The identity vid is added to ID.
3. **Voting phase**. The adversary may make two types of queries.
   - **Vote**(vid, v_L, v_R) queries. The adversary provides a voter identity vid such that $vid \in ID$ and two votes $v_L, v_R \in V$. The challenger runs $\mathsf{Vote}(vid, v_L, \sigma_{v_L}(s_{uciv}^{vid}), p_{uciv}^{vid}, epk)$ which outputs b_L and $\mathsf{Vote}(vid, v_R, \sigma_{v_R}(s_{uciv}^{vid}), p_{uciv}^{vid}, epk)$ which outputs b_R. C then obtains new versions of the boards L and R by running $\mathsf{ProcessBallot}(L, b_L, vid)$ and $\mathsf{ProcessBallot}(R, b_R, vid)$ and updating the boards accordingly.
   - **Ballot**(b, vid) queries. These are queries made on behalf of corrupt voters. Here the adversary provides a ballot b and an identity vid such that $vid \in ID$. The challenger runs $\mathsf{ProcessBallot}(L, b, vid)$ and if the process accepts it also runs $\mathsf{ProcessBallot}(R, b, vid)$ and updates the boards accordingly.
4. **Tallying phase**. The challenger evaluates $\mathsf{Tally}(L, esk)$ obtaining the result r and the proof of correct tabulation Π. If $\beta = 0$, the challenger posts (r, Π) on the bulletin board L. If $\beta = 1$, the challenger runs $\mathsf{SimProof}(R, r)$ obtaining a simulated proof Π' and posts (r, Π') on the bulletin board R.
5. **Output**. The adversary A outputs a bit $\alpha^{A,V}$.

We say that a voting protocol as defined in Section 2.1 provides ballot privacy if there exists an algorithm $\mathsf{SimProof}$ such that for any probabilistic polynomial time (p.p.t.) adversary A and any set of voting options V, the following advantage is negligible in the security parameter λ.

$$\mathbf{Adv}_V^{\mathbf{priv}}(A) := |\Pr[\alpha^{A,V} = 1|\beta = 1] - \Pr[\alpha^{A,V} = 1|\beta = 0]|$$

We want to remark that in case of honest voters the ballots are properly encrypted. In other words, this implies that the voting devices used to cast those ballots use *good* randomness and do not leak information about the randomness used. In addition, the ballot privacy does *not* rely on the adversary not having access to the secret UCIV information.

Universal cast-as-intended verifiability: Intuitively, a voting system satisfies the cast-as-intended property if a corrupt voting device is not able to cast a ballot for a voting option different to the one chosen by the voter. This should hold as long as the voter is honest. If the voter is malicious no guarantees can be given besides the fact that the ballot must correspond to at most one voting option (i.e., it could also correspond to an invalid voting option which will not

be counted). We define cast-as-intended on a per-ballot basis, not considering the tallying phase inside the definition.

In this definition, we use an extractor algorithm $\mathsf{Extract}$ which is defined as follows: for any (epk, esk) in the image of Setup, for any voter identity vid, for any correctly generated public and secret UCIV information $p_{uciv}^{vid}, s_{uciv}^{vid}$ and for any $v \in V$, it is satisfied that $\mathsf{Extract}(\mathsf{Vote}(vid, v, \sigma_v(s_{uciv}^{vid}), p_{uciv}^{vid}, epk), esk) = v$.

Universal cast-as-intended verifiability is defined as an experiment between an adversary A and a challenger C. In this experiment, the adversary may corrupt registrars, voters or voting devices. The goal of the adversary is to cast ballots in behalf of a non-corrupt voter so that the ballot does not extract to the voting option chosen by the voter. The *extract* is one with which the voting scheme is strongly consistent, which according to [4] states that the tally of a bulletin board must correspond to the result of applying a counting function to the *contents* of the ballots in the bulletin board. The experiment is parametrized by the set of voting options V and an algorithm $\mathsf{Extract}(b, esk)$ such that, given a ballot and the election private key returns a vote or $\perp$ denoting an invalid vote.

1. **Setup phase**. The challenger sets up an empty bulletin board BB and runs the $\mathsf{Setup}(1^\lambda)$ protocol to obtain the election public key epk and the election private key esk, posts epk on the board and gives (epk, esk) to the adversary. The adversary is given read access to BB. In addition, C initializes three empty lists ID_R, ID_P, ID_F. For convenience, we define $ID = ID_P \cup ID_F$
2. **Registration phase**. The adversary may make one type of query.
   - **Register**(vid) query. The adversary provides a voter identity such that $(vid) \notin ID_R$. The challenger runs $\mathsf{Register}(vid, V, epk)$ to generate the public UCIV information p_{uciv}^{vid} and the secret UCIV information s_{uciv}^{vid}. C provides p_{uciv}^{vid} to A, publishes p_{uciv}^{vid} on the bulletin board and adds vid to ID_R.
3. **Voting phase**. The adversary may make two types of queries.
   - **CorruptVotingDevice**(vid, v_{vid}) queries. The adversary provides a voter identity $vid \notin ID$ such that $vid \in ID_R$ and a voting option v_{vid} corresponding to such identity. Then, C provides $\sigma_{v_{vid}}(s_{uciv}^{vid})$ to A. The challenger adds vid to ID_P.
   - **CorruptVoter**(vid) queries. The adversary provides a voter identity $vid \notin ID$ such that $vid \in ID_R$. Then, C provides s_{uciv}^{vid} to A. The challenger adds vid to ID_F.
4. **Output**. The adversary submits a pair (b^*, vid^*). The output of the experiment is a bit δ^V, which is defined as 1 if (i) $vid^* \in ID_P$, (ii) $\mathsf{ProcessBallot}(BB, b^*, vid^*) = 1$ and (iii) $\mathsf{Extract}(b^*, esk) \neq v_{vid^*}$, where v_{vid^*} is the voting option submitted by the adversary in the CorruptVotingDevice query for vid^*. δ^V, is defined as 0 in any other case.

We say that a voting protocol as defined in Section 2.1 has universal cast-as-intended verifiability with respect to a counting function ρ if there exists an algorithm $\mathsf{Extract}$ such that the following two conditions hold:

(i) for all sets of voting options V the voting protocol is strongly consistent with respect to $\rho, \mathsf{Extract}$.

(ii) for any probabilistic polynomial time (p.p.t.) adversary A and any set of voting options V, the following advantage is negligible as a function of λ.

$$\mathbf{Adv}_V^{\mathbf{uciv}}(A) := \Pr[\delta^V = 1]$$

We want to remark that the universal cast-as-intended verifiability property does not rely on the secrecy of the private election key.

3 Building blocks

Encryption scheme. An encryption scheme consists of three probabilistic polynomial time (PPT) algorithms: KGen, Enc_{pk} and Dec_{sk}. On input a security parameter 1^k, the KGen algorithm outputs a public key pk and a secret key sk, and implicitly defines a message space M_{sp}, a ciphertext space C_{sp} and a randomness space R_{sp}. The Enc_{pk} algorithm takes as input a message $M \in M_{sp}$ and uses the public key pk to output a ciphertext $C \in C_{sp}$. The Dec_{sk} algorithm takes as input a ciphertext $C \in C_{sp}$ and uses the secret key sk to output a message $M \in M_{sp}$, or halts outputting $\perp$.

We use the notion of $\mathsf{NM\text{-}CPA}$ security [5] for the encryption scheme.

One-way functions. Roughly speaking, a one-way function is a function which is easy to compute but very difficult to invert. More formally, a function $F : X \to Y$ between two finite sets is said to be one-way if the following two properties are satisfied, where $k = \log |X|$: (i) For each $x \in X$, the value $F(x)$ can be computed in time polynomial in k; (ii) For any algorithm A running in time polynomial in k, and for $x \in X$ chosen with the uniform distribution, the probability that A, on input $F(x)$, outputs x is negligible.

Non-interactive zero-knowledge proofs of knowledge. Namely, a NIZKPK for the relation R is composed by three PPT algorithms (GenCRS, Prove, Verify): 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. Verify takes as input the 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 following three properties (see for instance [12,23]): completeness, knowledge soundness and zero-knowledge, which implies *witness indistinguishability* [14].

4 Core voting protocol

4.1 Overview

In this Section we present our new voting protocol. We will not take into account usability issues from a voter's point of view; we will take care of these issues on

the voting systems proposed in Section 7. We think that by splitting the core protocol from the voting systems built on top of it the reader can get a clearer picture of our solution.

Our protocol is a mix-net based voting protocol, although the ideas can be also applied to the setting of homomorphic tallying schemes. In mix-net based protocols, the voters encrypt their votes (we will assume that each selection is encrypted individually), then all the ciphertexts submitted by the voters are shuffled and re-encrypted in several nodes and then the shuffled ciphertexts are decrypted. We will work with a simplified scheme where votes are only shuffled and decrypted once.

The goal of our work is to provide a protocol which provides ballot privacy and universal cast-as-intended verifiability. Other requirements of voting schemes such as the assurance of the authorship of the vote (an eligible voter), recorded-as-cast verifiability and counted-as-recorded verifiability are not considered in this work, since other measures can be built around the protocol in order to fulfill them. For instance, digital signatures, public bulletin boards or verifiable mix-nets could be used to satisfy the mentioned properties respectively.

The main idea of the core protocol is the following: during the registration phase, a voter registers to vote. The registrar generates a secret value for each voting option of the election, and provides them to the voter. A one-way function will be applied to each secret value and the resulting images will be made public (maintaining the relation with the voting options).

Once the voter has selected her voting options by using a voting device, she will provide a subset of the secret values (previously unknown) to the voting device. Then the voting device will encrypt the voting options, and will create a non-interactive zero-knowledge proof of knowledge (NIZKPK). By carefully choosing which secret values the voter discloses, the voting device will only be able to create a valid NIZKPK if it really encrypted the voter's selections.

The NIZKPK used in the protocol can be publicly verified, since the voter's privacy is maintained in such case due to the zero-knowledge property of the proofs. Therefore, the system provides *universal cast as intended verifiability* since the proofs can be universally verified, and this validation is successful only in the case the voting device encrypted the voter's selections.

A sketch on how this NIZKPK works is the following: In the voting phase, the voter will provide the secret values whose images are associated to the voting options she did not choose to the voting device; this idea is reminiscent of the Bingo voting scheme [6]. The voting device will then create, for each possible voting option, a NIZKPK of the statement "either this voting option is encrypted in the ciphertext *or* I know the pre-image of the public value which corresponds to it". Without further information, the voting device will be able to create all the proofs as long as it encrypted the voter's selected voting option.

The proofs also guarantee that, as long as the voter behaved honestly, the encrypted vote contains a valid voting option. This might be useful to discern whether a malformed vote was intentionally created by a malicious voter or by a malicious voting device (used by a honest voter). Note that, as we are giving a

mix-net based protocol, if both the voter and the voting device are malicious and submit a vote containing a non-valid option, this vote will be anyway discarded once the votes have been mixed and decrypted, prior to being added to the count. Therefore, this is not an attack against the system.

4.2 2-cnf-proof of knowledge

In Section 3 we defined an encryption scheme $(\mathsf{KGen}, \mathsf{Enc}_{pk}, Dec)$ with an associated message space M_{sp}, ciphertext space C_{sp} and randomness space R_{sp}. Consider the relation $R_{enc} = \{((C, M); r) | M \in M_{sp}, C \in C_{sp}, r \in R_{sp}, C = \mathsf{Enc}_{pk}(M; r)\}$ which consists of the tuples of ciphertexts, messages and randomness such that the ciphertext is an encryption of the message using the specified randomness. Note that, in this relation, the statement is the pair ciphertext - message, while the witness is the randomness.

Also consider that we have a one-way function F with an associated input space X and output space Y. Consider the relation $R_{ow} = \{(y; x) | x \in X, y \in Y, y = F(x)\}$ which consists of the pairs of values such that the first value is the image of the second one under the one-way function F. In this case, the image value (i.e., the first element, y) is the statement while the pre-image (i.e., the second element, x) is the witness.

The relation for which we will make the NIZKPK is the following one:

$$\begin{aligned} R_{2\text{-}cnf} = \{((C, (M_1, \ldots, M_n), (y_1, ..., y_n)); (r, (x_1, \ldots, x_n)))| \\ (((C, M_1); r) \in R_{enc} \vee (x_1, y_1) \in R_{ow}) \wedge \\ (((C, M_2); r) \in R_{enc} \vee (x_2, y_2) \in R_{ow}) \wedge \\ \cdots \qquad\qquad \wedge \\ (((C, M_n); r) \in R_{enc} \vee (x_n, y_n) \in R_{ow})\} \end{aligned}$$

This way of presenting the relation $R_{2\text{-}cnf}$ allows us to analyze it easily: the relation takes a statement which is a ciphertext, a set of messages and a set of image values, and a witness, which is some randomness and a set of pre-images. The tuple of statement and witness belongs to the language iff every predicate in the *AND* clause is satisfied. This is, we require that for every $i \in \{1, \ldots, n\}$ the predicate $(((C, M_i); r) \in R_{enc} \vee (x_i, y_i) \in R_{ow})$ is satisfied. This predicate perfectly captures the intuition given above: either the ciphertext encrypts the message M_i or a pre-image of y_i is known. The description of the language $R_{2\text{-}cnf}$ will also allow us to generate a NIZKPK from the composition of simpler NIZKPKs for the relations R_{enc} and R_{ow}. In Section 6 we present efficient NIZKPK systems for this $R_{2\text{-}cnf}$ relation.

4.3 Detailed protocol

Our protocol implements a voting scheme where the counting function ρ is the multiset function as defined in [4]. This function returns a random permutation of its input, filtering invalid votes. This is the case for all mix-net based protocols.

Here we describe the protocol for single-mark ballots. An extension for multiple-mark ballots is provided in the full version.

The protocol uses as building blocks an encryption scheme $(\mathsf{KGen}, \mathsf{Enc}_{pk}, \mathsf{Dec}_{sk})$, a one-way function F and a NIZKPK $(\mathsf{GenCRS}, \mathsf{Prove}, \mathsf{Verify})$ for the relation $R_{2\text{-}cnf}$ defined in Section 4.2.

In our protocol, the secret UCIV information for a voter is a set of pre-images $(x_1^{vid}, \ldots, x_n^{vid})$ of F, one for each element in V (the set of voting options), and the public UCIV information is $(y_1^{vid}, \ldots, y_n^{vid}) = (F(x_1^{vid}), \ldots, F(x_n^{vid}))$, the images of each element of the secret UCIV information. Each pair (x_i^{vid}, y_i^{vid}) is related to the corresponding voting option v_i, and the relation between y_i^{vid} and v_i is public.

The function σ_{v_i} has as input the set of all pre-images $(x_1^{vid}, \ldots, x_n^{vid})$ and outputs the the same values except for x_i^{vid}. Note that this implicitly defines $SVI, \widetilde{SVI}, PVI$. As all this information is only determined by F, there is no need to consider the specification of $SVI, \widetilde{SVI}, PVI, \sigma$ as part of the Setup algorithm.

The protocol consists of the following algorithms:

$\mathsf{Setup}(1^\lambda)$: the algorithm first runs GenCRS to generate the common reference string crs. The algorithm also runs KGen to generate a pair of public/private encryption keys (pk, sk). This implicitly defines the message space M_{sp}. For the sake of simplicity, we will assume that $V \subset M_{sp}$ so that no specific encoding is needed. The election public key epk is defined as $epk = (\mathsf{crs}, pk)$ and the election secret key is defined as $esk = sk$.

$\mathsf{Register}(vid, V, epk)$: the algorithm generates the secret UCIV information s_{uciv}^{vid} for that voter by generating, for each voting option $v_i \in V$, a random value $x_i^{vid} \in X$. The public UCIV information p_{uciv}^{vid} is generated by computing the image $y_i^{vid} = F(x_i^{vid})$ for each voting option.

$\mathsf{Vote}(vid, v, \sigma_v(s_{uciv}^{vid}), p_{uciv}^{vid}, epk)$: the algorithm parses epk as (crs, pk). It then parses $\sigma_v(s_{uciv}^{vid})$ as $\{\{x_i^{vid}\}_{i \text{ s.t. } v_i \neq v}\}$ and p_{uciv}^{vid} as $\{(y_1^{vid}, \ldots, y_n^{vid})\}$. It then runs the Enc_{pk} algorithm using fresh randomness $r \in R_{sp}$, producing a ciphertext $C = \mathsf{Enc}_{pk}(v, r)$. Then, by using the witness $w = (r, (\xi_1^{vid}, \ldots, \xi^{vid}))$, where ξ_k^{vid} with $k = j$ such that $v_j = v$ is the empty string ε and it is x_k^{vid} otherwise, the algorithm computes a NIZKPK for the relation $R_{2\text{-}cnf}$ as $\pi = \mathsf{Prove}(\mathsf{crs}, C, y_1^{vid}, \ldots, y_n^{vid}, w)$. The resulting ballot b is defined by the pair (C, π).

$\mathsf{ProcessBallot}(BB, b, vid)$: upon reception of a ballot b, which can be parsed as $b = (C, \pi)$, it is checked if in the bulletin board there is another pair (id, b') (i.e., with the same id), or another ballot $b' = (C', \pi')$ such that $C = C'$. If any of such case is found, the algorithm stops and returns 0. Otherwise, p_{uciv}^{vid} is recovered from the bulletin board and parsed as $\{(y_1^{vid}, \ldots, y_n^{vid})\}$. Then, $\mathsf{Verify}(\mathsf{crs}, C, y_1^{vid}, \ldots, y_n^{vid}, \pi)$ is run, where crs is also recovered from the bulletin board. The output of Verify is returned as the output of $\mathsf{ProcessBallot}$.

$\mathsf{Tally}(BB, esk)$: at the end of the election, $\mathsf{ProcessBallot}(BB, b, vid)$ is run for all pairs (b, vid) appearing in the bulletin board. Then, each individual ballot b is decrypted $\tilde{v} = \mathsf{Dec}_{sk}(C, sk)$ and ρ is applied to the resulting decryptions $\{\tilde{v}\}$.

The output of ρ is defined as the result and the proof of correct tabulation is defined to be the empty string ε.

5 Security of the protocol

Here we provide the results of our security analysis for ballot privacy and universal cast-as-intended verifiability. The security proofs will be available in the full version of the paper.

Theorem 1. *Let* $(\mathsf{KGen}, \mathsf{Enc}_{pk}, \mathsf{Dec}_{sk})$ *be a NM-CPA secure encryption scheme and let* $(\mathsf{GenCRS}, \mathsf{Prove}, \mathsf{Verify})$ *be a NIZKPK for the relation* $R_{2\text{-}cnf}$ *defined in Section 4.2. Then, the protocol defined in Section 4.3 satisfies the ballot privacy property.*

Theorem 2. *Let* F *be a one-way function and let* $(\mathsf{GenCRS}, \mathsf{Prove}, \mathsf{Verify})$ *be a NIZKPK. Let the number of voting options be polynomial in the security parameter. Then, the protocol defined in Section 4.3 satisfies the universal cast-as-intended property.*

6 A possible instantiation

The instantiation of our protocol that we propose is quite efficient and its security holds in the ROM. It works over a fixed cyclic group $G = \langle g \rangle$ of prime order q. It uses a modified version of ElGamal with a Schnorr proof (Signed ElGamal [24]), which is $\mathsf{NM\text{-}CPA}$ secure in the ROM; the encryption of a message $m \in G$ becomes $C = (g^r, pk^r \cdot m, h, z)$, where $r, s \in Z_q$ are randomly chosen, $h = H(g^r, g^s, pk^r \cdot m)$ for some suitable hash function H (included in the common reference string) and $z = r + sh \mod q$.

The exponentiation function defined as $F(x) = g^x$, given an $x \in Z_q$, with the same cyclic group and generator used for the encryption scheme, is used for the one-way function

Finally the NIZKPK for the relation $R_{2\text{-}cnf}$ is constructed by combining existing techniques. On the one hand, a σ-protocol for proving knowledge of a discrete logarithm [24] (also known as a Schnorr protocol) and a σ-protocol for proving equality of two discrete logarithms [9] (also known as a Chaum-Pedersen protocol). On the other hand, as shown in [10], given a σ-protocol for a relation R_1 and another σ-protocol for a relation R_2 one can construct both a σ-protocol for the relation R_{or} defined as $(x, w) \in R_{or}$ iff $(x, w) \in R_1$ or $(x, w) \in R_2$ and a σ-protocol for the relation R_{and} defined as $(x, w) \in R_{and}$ iff $(x, w) \in R_1$ and $(x, w) \in R_2$.

Focusing on the election scheme for single-mark ballots, with n candidates, a ballot in this specific instantiation of our scheme will have the form (C, π), where $C = (c_1, c_2, c_3, c_4) = (g^r, pk^r \cdot v, h, z)$, if the chosen option is $v = v_j$, and the non-interactive zero-knowledge proof of knowledge $\pi = (h, \{h_1^{(i)}, s_1^{(i)}, s_2^{(i)}\}_{1 \leq i \leq n})$ for the relation $R_{2\text{-}cnf}$ can be computed by combining the Fiat-Shamir heuristic

and the afore-mentioned techniques, as detailed below (for simplicity, we denote $x_i = x_i^{vid}$ and $y_i = y_i^{vid}$, for each $i = 1, \ldots, n$):

1. For $i = 1, \ldots, n$, $i \neq j$, choose $s_1^{(i)}, h_1^{(i)} \in Z_q$ at random. Define the values $c_1^{(i)} = g^{s_1^{(i)}} \cdot c_1^{-h_1^{(i)}}$ and $c_2^{(i)} = pk^{s_1^{(i)}} \cdot (c_2/v_i)^{-h_1^{(i)}}$.
2. For index j, choose $s_2^{(j)}, h_2^{(j)} \in Z_q$ at random, and define the value $R^{(j)} = g^{s_2^{(j)}} \cdot y_j^{-h_2^{(j)}}$.
3. For $i = 1, \ldots, n$, $i \neq j$, choose $\alpha^{(i)} \in Z_q$ at random and compute $R^{(i)} = g^{\alpha^{(i)}}$.
4. For index j, choose $\beta^{(j)} \in Z_q$ and $z^{(j)} \in G$, at random, and compute $c_1^{(j)} = g^{\beta^{(j)}}$ and $c_2^{(j)} = pk^{\beta^{(j)}}$.
5. Compute the hash function $h = H(c_1, c_2, c_3, c_4, \{c_1^{(i)}, c_2^{(i)}, R^{(i)}\}_{1 \leq i \leq n}) \in Z_q$.
6. For $i = 1, \ldots, n$, $i \neq j$, compute $h_2^{(i)} = h - h_1^{(i)} \mod q$. For index j, compute $h_1^{(j)} = h - h_2^{(j)} \mod q$.
7. For $i = 1, \ldots, n$, $i \neq j$, compute $s_2^{(i)} = \alpha^{(i)} + x_i h_2^{(i)} \mod q$.
8. For index j, compute $s_1^{(j)} = \beta^{(j)} + r h_1^{(j)} \mod q$.

To verify the correctness of the proof π, one has to compute first, for each $i = 1, \ldots, n$, the values $h_2^{(i)} = h - h_1^{(i)} \mod q$, $c_1^{(i)} = g^{s_1^{(i)}} \cdot c_1^{-h_1^{(i)}}$, $c_2^{(i)} = pk^{s_1^{(i)}} \cdot (c_2/v_i)^{-h_1^{(i)}}$ and $R^{(i)} = g^{s_2^{(i)}} \cdot y_i^{-h_2^{(i)}}$. After that, the proof is accepted if and only if $h = H(c_1, c_2, c_3, c_4, \{c_1^{(i)}, c_2^{(i)}, R^{(i)}\}_{1 \leq i \leq n})$.

Efficiency comparison to Helios Each element in Z_q and each element in G can be represented with λ bits, where λ is the length in bits of the prime number q (that is, the security parameter of the scheme). Therefore, the size of each ballot (C, π) in this instantiation is $(3n+5)\lambda$ bits: 4λ bits for C and $(3n+1)\lambda$ bits for π. The number of modular exponentiations that must be computed to generate such a ballot is $3n + 4$.

We can compare these costs with the costs of the basic version of Helios for the case of a single-mark election, with n candidates. Let us remember that a ballot in Helios consists of n ElGamal encryptions $C_1, \ldots, C_n$ of 0 or 1, one encryption for each of the n candidates (the voter encrypts 1 only for the chosen option, and 0 everywhere else), along with a non-interactive zero-knowledge proof of knowledge π of the fact that each ciphertext encrypts 0 or 1, and the product of all the ciphertexts (which is an encryption of the sum of the n plaintenxts, by the homomorphic properties of ElGamal) also encrypts 0 or 1. The size of each ballot in Helios is $(5n + 4)\lambda$ bits: $2n\lambda$ bits for the n ElGamal ciphertexts, and $(3(n + 1) + 1)\lambda$ bits for π. The number of modular exponentiations required to generate a ballot in Helios is $6n$.

The instantiation that we have just presented is more efficient than the basic version of Helios, for single-mark elections, in terms of the size of the ballots and the computational cost to generate them, while adding a new property (universal cast-as-intended verifiability). This efficiency comparison shows that our new protocol has ballots of a reasonable size and that ballot creation is efficient enough to be used in a real election.

7 Towards designing usable UCIV systems

The protocol presented in Section 4 requires the voter to provide one secret value per each of the voting options that she did not choose to the voting device. This may require a considerable effort, since the number of voting options available in an election may be quite large, even if the voter can only make one selection (as in single-marking ballot elections). In addition, these values will be fairly big, for instance 256 bits if elliptic curves are used. Finally, providing secrets for the non-selected options is counter-intuitive (note that it is just inverse to what is required in code-voting schemes, for example), since the voter usually expects to enter information related to what she selects. Clearly, requiring the voter to introduce such information is not very user-friendly.

We present now a voting system built on top of our new voting protocol which provides a more intuitive approach for the voter introducing the codes. We provide this example as a first step towards an implementation in a real scenario: usability tests should be done to assess the real usability of this system.

The voting system that we propose uses high-capacity barcodes such as QR codes. These QR codes will contain the secret values the voter has to enter in the voting device. A paper sheet with printed QR codes provides means for storage for those secrets, as well as a channel which can not be eavesdropped by the voting device. The security of the channel depends on the method for delivering the QR sheets to the voters, for example in sealed envelopes sent by postal mail.

The QR contents can be organized in a way such that the voter is only required to scan the QR codes corresponding to the options she selects in order to retrieve the secrets corresponding to the other voting options.

For each voter, a QR paper sheet will be created as follows. First, one QR code will be assigned to each voting option. Let n be the total number of voting options and let ℓ be the number of voting options which the voter can select. Then, each secret value x_i^{vid} (which corresponds to the voting option v_i) will be divided into $n-1$ shares using a threshold secret sharing scheme [25] with threshold ℓ. This will result in shares $x_{i,k}^{vid}$ for $k \in \{1,\ldots,n\}\backslash i$, this is, $n-1$ shares, each one assigned to each voting option different from v_i. Then, the QR code assigned to the option v_i will be created containing all the shares $x_{j,i}^{vid}$ for $j \in \{1,\ldots,n\}\backslash i$, this is, the i-th share of each secret value except for x_i^{vid}.

It follows that, given the information of ℓ QRs codes (assigned to ℓ voting options), only the secret values for the voting options not assigned to those QR codes can be reconstructed from the shares inside the QR codes. Scrath surfaces or stickers can be used in order to enforce that only the designated QR codes are scanned by the voting device's camera.

Note that the maximum capacity of a QR code is around 23 thousand bits, which should be more than enough, since each QR code will contain $n-1$ secret values of size 256 bits each (if using elliptic curves).

This system can be used either in remote or in poll-site/kiosk-based electronic voting schemes. The QR sheets may be distributed to the voters by postal mail or by hand, depending on the scenario.

8 Future work

We have proposed in this work a generic way to achieve the new notion of universal cast-as-intended verifiability. It would be interesting, and even necessary, to investigate if this new property can be achieved by other means, for instance by modifying existing voting protocols. In particular, the systems Scratch&Vote [2] and Pretty Good Democracy [21] seem very good candidates; taking into account that commitments (or encryptions) are particular cases of one-way functions, our intuition is that a modification of Scratch&Vote or of Pretty Good Democracy would lead to schemes that could be thought as a particular instantiation of our generic construction. The formalization of this intuition, as well as the comparison with the implementation proposed in this work, is left as future work.

Another interesting line of research is to improve the efficiency of the voting protocol. In particular, the size of ballots scales badly with the number of voting options and the number of choices a voter can make. This is in contrast with typical (non-UCIV) mix-net protocols, where the size of ballots increases at most linearly with the number of choices a voter can make.

Although in this work we focused on the notions of privacy provided by existing schemes such as [1], we consider for future works to design a protocol with universal cast-as-intended verifiability, together with stronger notions of privacy such as receipt-freeness or coercion-resistance. Another challenge to solve is to design a voting protocol such that the voting device does not learn the identity of the voter even in front of dishonest registrars.

Finally, we consider there is work to do in the usability field. Although we have proposed a solution for making the voting process more intuitive for the voter, there is still a long way to walk to have a usable voting scheme. Feedback from usability tests and user experience designers may be used in order to find out the best approaches to follow.

References

1. Adida, B.: Helios: Web-based open-audit voting. In: van Oorschot, P.C. (ed.) USENIX Security Symposium. pp. 335–348. USENIX Association (2008)
2. Adida, B., Rivest, R.L.: Scratch & vote: self-contained paper-based cryptographic voting. In: Juels, A., Winslett, M. (eds.) ACM Workshop on Privacy in the Electronic Society, WPES 2006. pp. 29–40. ACM (2006)
3. Benaloh, J.: Simple verifiable elections. In: Electronic Voting Technology Workshop. pp. 5–5. EVT'06, USENIX Association, Berkeley, CA, USA (2006)
4. Bernhard, D., Cortier, V., Galindo, D., Pereira, O., Warinschi, B.: Sok: A comprehensive analysis of game-based ballot privacy definitions. In: IEEE Symposium on Security and Privacy, SP 2015. pp. 499–516. IEEE Computer Society (2015)
5. Bernhard, D., Pereira, O., Warinschi, B.: IACR Cryptology ePrint Archive
6. Bohli, J., Müller-Quade, J., Röhrich, S.: Bingo voting: Secure and coercion-free voting using a trusted random number generator. In: Alkassar, A., Volkamer, M. (eds.) E-Voting and Identity, VOTE-ID 2007. Lecture Notes in Computer Science, vol. 4896, pp. 111–124. Springer (2007)

7. Chaum, D.: Physical and digital secret ballot systems (Aug 2 2001), WO Patent App. PCT/US2001/002,883
8. Chaum, D.: Secret-ballot receipts: True voter-verifiable elections. IEEE Security & Privacy 2(1), 38–47 (2004)
9. Chaum, D., Pedersen, T.: Wallet databases with observers. In: Brickell, E. (ed.) CRYPTO '92, Lecture Notes in Computer Science, vol. 740, pp. 89–105. Springer Berlin Heidelberg (1993)
10. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 839, pp. 174–187. Springer (1994)
11. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. In: Fumy, W. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 1233, pp. 103–118. Springer (1997)
12. Damgård, I.: Commitment schemes and zero-knowledge protocols. In: Damgård, I. (ed.) Lectures on Data Security, Lecture Notes in Computer Science, vol. 1561, pp. 63–86. Springer Berlin Heidelberg (1999)
13. Essex, A., Clark, J., Hengartner, U., Adams, C.: Eperio: Mitigating technical complexity in cryptographic election verification. IACR Cryptology ePrint Archive 2012, 178 (2012)
14. Feige, U., Shamir, A.: Witness indistinguishable and witness hiding protocols. In: 22nd Annual ACM Symposium on Theory of Computing. pp. 416–426. STOC '90, ACM Press (1990)
15. Gerck, E., Neff, C.A., Rivest, R.L., Rubin, A.D., Yung, M.: The business of electronic voting. In: Syverson, P.F. (ed.) 5th Financial Cryptography, FC 2001. Lecture Notes in Computer Science, vol. 2339, pp. 234–259. Springer (2001)
16. Gharadaghy, R., Volkamer, M.: Verifiability in electronic voting - explanations for non security experts. In: Krimmer, R., Grimm, R. (eds.) Electronic Voting. LNI, vol. 167, pp. 151–162. GI (2010)
17. Groth, J., Ostrovsky, R.: Cryptography in the multi-string model. J. Cryptology 27(3), 506–543 (2014)
18. Joaquim, R., Ribeiro, C., Ferreira, P.: Veryvote: A voter verifiable code voting system. In: Ryan, P.Y.A., Schoenmakers, B. (eds.) E-Voting and Identity, VOTE-ID 2009. Lecture Notes in Computer Science, vol. 5767, pp. 106–121. Springer (2009)
19. Neff, C.A.: Practical high certainty intent verification for encrypted votes (2004)
20. Ryan, P.Y.A., Bismark, D., Heather, J., Schneider, S., Xia, Z.: Prêt à voter: a voter-verifiable voting system. IEEE Transactions on Information Forensics and Security 4(4), 662–673 (2009)
21. Ryan, P.Y.A., Teague, V.: Pretty good democracy. In: Christianson, B., Malcolm, J.A., Matyas, V., Roe, M. (eds.) 17th International Workshop on Security Protocols, 2009. Lecture Notes in Computer Science, vol. 7028, pp. 111–130. Springer (2009)
22. Sako, K., Kilian, J.: Receipt-free mix-type voting scheme - a practical solution to the implementation of a voting booth. In: Guillou, L.C., Quisquater, J.J. (eds.) EUROCRYPT. LNCS, vol. 921, pp. 393–403. Springer (1995)
23. Santis, A.D., Persiano, G.: Zero-knowledge proofs of knowledge without interaction (extended abstract). In: FOCS. pp. 427–436. IEEE Computer Society (1992)
24. Schnorr, C.P., Jakobsson, M.: Security of Signed ElGamal encryption. In: Okamoto, T. (ed.) ASIACRYPT. LNCS, vol. 1976, pp. 73–89. Springer (2000)
25. Shamir, A.: How to share a secret. Commun. ACM 22(11), 612–613 (Nov 1979)