**OpenVoting: Making E2E-V voting transparent and recoverable**

Prashant Agrawal  
Kabir Tomer  
Computer Science and Engineering  
Indian Institute of Technology Delhi  
New Delhi, India

Abhinav Nakarmi  
Mahabir Prasad Jhanwar  
Computer Science  
Ashoka University  
Sonepat, India

Subodh Sharma  
Subhashis Banerjee  
Computer Science and Engineering  
Indian Institute of Technology Delhi  
New Delhi, India

**ABSTRACT**

End-to-end verifiable (E2E-V) voting systems have been around for some time. However, their adoption in large elections is poor, seemingly because of the inaccessibility of their underlying complex cryptography to the general electorate. Meanwhile, risk-limiting audits based on voter-verified paper records (VVPR) have been effective in bringing easy-to-understand verifiability and recoverability to electronic voting, but they generally require the electorate to trust the post-election custody chain of the paper trail.

In this paper, we propose **OpenVoting**, a novel polling booth voting protocol that publicly demonstrates a one-to-one correspondence between the cryptographically-secured electronic vote records and the easily understandable paper records, while protecting individual voter secrecy and polling booth-level voting statistics. This one-to-one correspondence helps each provide a check for the other, improves overall transparency, and also enables efficient and principled recovery in case of tally mismatches, by pinpointing mismatching votes and their associated polling booths. We propose a novel distributed zero-knowledge proof (ZKP) that facilitates the above.

To an ordinary voter **OpenVoting** looks just like an old fashioned paper based voting system, with minimal additional cognitive overload.

**KEYWORDS**

Voting; end-to-end verifiable; VVPR

**1 INTRODUCTION**

As recent events have demonstrated, conducting large scale public elections in a dispute-free manner is not an easy task. On the one hand, although end-to-end verifiable (E2E-V) cryptographic voting systems can theoretically guarantee correctness of elections, they are not yet very popular. Not only do they shift significant parts of the responsibilities of audit from the authorities to individual voters, but also observations such as the one made by the German Constitutional Court [35] makes depending entirely on cryptographic guarantees somewhat untenable:

“The use of voting machines which electronically record the voters’ votes and electronically ascertain the election result only meets the constitutional requirements if the essential steps of the voting and of the ascertainment of the result can be examined reliably and without any specialist knowledge of the subject … The legislature is not prevented from using electronic voting machines in elections if the possibility of a reliable examination of correctness, which is constitutionally prescribed, is safeguarded. A complementary examination by the voter, by the electoral bodies or the general public is possible for example with electronic voting machines in which the votes are recorded in another way beside electronic storage.”

Moreover, in case audits fail and elections do not verify, the E2E-V systems do not usually provide easy methods of recovery without necessitating complete re-election [13].

On the other hand, there are systems that mainly rely on paper-audit trails for correctness of elections [12, 32, 36, 43, 44]. In such systems, reliable records of cleartext voter-marked paper ballots or voter-verified paper records (VVPRs) are maintained in addition to the electronic vote records. These systems use electronic counting primarily for efficiency and perform rigorous statistical audits, called risk-limiting audits (RLA) [13], to establish that the winners declared by the electronic system match those that would have been declared by the full paper count. In case of serious mismatches, the electronic outcome is suggested to be replaced by the paper one.

However, paper records, which may be trustworthy at the time of voting, may not remain trustworthy at the time of counting or auditing. Hence RLA-based systems require the electorate to trust that a strict compliance audit [13, 44] of the custody chain of paper records – using traditional methods – is performed correctly.

It is thus natural to ask whether the strong cryptographic guarantees provided by E2E-V systems can be effectively combined with the understandability of paper records, in a hybrid dual voting system [11]. However, to design such a scheme, one must be careful not to end up running two parallel and independent elections, only coupled loosely through simultaneous voting for both in the polling...
we achieve through a novel threshold zero-knowledge proof (ZKP).

The encrypted votes on \( BB_1 \), forms the encrypted electronic vote. These encrypted votes voter receipt from the ballot. The VVPR is cast in the traditional voting stage, while maintaining strong vote secrecy guarantees. At

threshold case. Representing an encrypted vote as a commitment can be clearly identified.

receipts not correctly included in the tally and all spurious votes corresponding cleartext votes (property \( P1 \)) and all cleartext votes without corresponding encrypted votes (property \( P2 \)). Thus, all voter receipts not correctly included in the tally and all spurious votes can be clearly identified.

We obtain property \( P1 \) by extending a ZKP of set-membership by Camenisch et al. \cite{camenisch2003} — which was for the single prover case — to the threshold case. Representing an encrypted vote as a commitment

booth. If the electronic and paper record systems are not tightly coupled, then it begs the question that which ought be the legal definition of the vote, and, in case of a mismatch in counts, which is the one that should be trusted and why? This question can be resolved if a one-to-one correspondence between the electronic and paper records can be publicly demonstrated. Then, in case of tally mismatches, instead of overriding one tally with the other, specific paper records without a corresponding electronic record (or vice-versa) can be identified and errors may be localised to their associated polling booths. Re-elections — if at all required — may then be localised only to such booths. Such transparent recoverability may be crucial for the credibility of election processes in case elections do not completely verify.

Our contribution. In this paper, we propose OpenVoting, a novel dual voting protocol that demonstrates a one-to-one correspondence between VVPRs and electronic vote representations at each voting stage, while maintaining strong vote secrecy guarantees. At a high level, OpenVoting proceeds as follows. A set of independent election authorities (EAs) jointly generate paper ballots. No EA singly holds the ballot secrets. During polling, an electronic ballot marking device (BMD) generates both the VVPR and an encrypted voter receipt from the ballot. The VVPR is cast in the traditional way and a cryptographic commitment of the cast vote extracted from the voter receipt, signed and uploaded by a polling officer (PO), forms the encrypted electronic vote. These encrypted votes are displayed on a public bulletin board \( BB_1 \) along with the voter id \( (V_{id}) \). Post polling, the encrypted votes on \( BB_1 \) are threshold decrypted and secretly permuted by the EAs and the thus obtained cleartext votes are published on a second public bulletin board \( BB_2 \). The encrypted votes on \( BB_1 \) are simply electronic representations of voter receipts, and the cleartext votes on \( BB_2 \) are electronic representations of VVPRs. A one-to-one correspondence between all these vote representations can therefore be established by demonstrating a one-to-one correspondence between \( BB_1 \) and \( BB_2 \), which we achieve through a novel threshold zero-knowledge proof (ZKP). See Figure 1 for a summary.

The threshold ZKP provides a novel E2E-V backend with better transparency and recoverability properties compared to existing E2E-V backends (see Figure 2). Specifically, in case of verification failures, it allows identifying all encrypted votes without corresponding cleartext votes (property \( P1 \)) and all cleartext votes without corresponding encrypted votes (property \( P2 \)). Thus, all voter receipts not correctly included in the tally and all spurious votes can be clearly identified.

We obtain property \( P1 \) by extending a ZKP of set-membership by Camenisch et al. \cite{camenisch2003} — which was for the single prover case — to the threshold case. Representing an encrypted vote as a commitment

[38] to the vote, it allows us to prove that the committed vote was included correctly in the set of cleartext votes on \( BB_2 \). Encrypted votes on \( BB_1 \) that do not verify can thus be easily identified.

For property \( P2 \), we propose a novel primitive — which we call a ZKP of reverse set-membership — that allows proving that a given message is committed by at least one of a set of commitments. We extend this primitive too to the threshold case, and use it to prove that each cleartext vote on \( BB_2 \) is a decryption of one of the encrypted votes on \( BB_1 \). As before, cleartext votes on \( BB_2 \) that do not verify can be easily identified.

Both ZKP of set membership and reverse set membership have an amortised complexity of \( O(n) \), where \( n \) represents the number of commitments or messages verified.

The OpenVoting protocol maintains strong secrecy guarantees. Unless a threshold number of EAs collude, they do not learn any information (beyond the final tally) about how anyone voted. The BMD does not see the cleartext votes, thereby preventing possible information leak to any hacker or coercer. Voter receipts cannot be linked to VVPRs or the published cleartext electronic votes. Also, cleartext votes or VVPRs are revealed only after aggregation from all polling booths. Thus, voting statistics at any given polling booth cannot be determined, ensuring community vote secrecy.

2 RELATED WORK

2.1 E2E-V frontends

E2E-V voting frontends generally consist of either optical scanning of hand-marked paper ballots \cite{lu2004, tegesa2006, chen2007, jhanwar2013, jhanwar2014} or direct recording electronic (DRE) machines \cite{hu2008, jhanwar2014}. Our optical scan ballot scheme is a variant of Scratch & Vote \cite{lu2004}. We use ballot marking devices and retain the simple voter experience provided by existing frontends.

2.2 E2E-V backends

Most E2E-V systems use either homomorphic encryption or mixnet based backends for tallying \cite{calandrino2017}. In homomorphic tallying \cite{lu2004, jhanwar2014}, an encrypted tally is computed by homomorphically adding the published encrypted votes. Election authorities can then decrypt the encrypted tally and provide a ZKP of correct decryption. In mixnet backends \cite{lu2004, tegesa2006, jhanwar2013, jhanwar2014} the published encrypted votes are decrypted by a verifiable procedure that shuffles and decrypts cast ballots through a set of mix servers, and then proves that it did so correctly.

Multiple techniques for verifying correctness of mixnet decryption exist (see \cite{calandrino2017}), of which the randomised partial checking technique \cite{calandrino2011} seems to be the most popular in voting. In this technique, each encrypted vote’s path through the mixnet is partially opened and verified. However, verification failures cannot be localised to individual failing items. If \( n \) votes are manipulated, this fact can be detected with probability at least 1 - \( (1/2^m) \), but the procedure cannot identify all \( n \) failures. This makes recovery from universal verification failures difficult.

Some works have explored the notion of sender-verifiable mixnets \cite{calandrino2017}, where each sender can verify that its message was correctly included in the mixnet output. This is closer to our requirement of localisability, but these mixnets require the sender to know the
randomness used to create its encrypted messages. Thus, they are not suitable for barehanded voting protocols.

### 2.3 VVPR support

| Protocol                | VVPR to electronic vote correspondence | Encrypted to cleartext vote correspondence | Backend |
|-------------------------|----------------------------------------|--------------------------------------------|---------|
| Scantegrity II [18]     | Localisable, non-public                | Aggregate                                  | Mixnet  |
| Sigma ballots [39]      | Aggregate, non-public                  | Aggregate                                  | Mixnet  |
| Benaloh’s VOpScan [11]  | Aggregate                               | Aggregate                                  | Mixnet  |
| Wombat [25]             | Aggregate                               | N/A\(^1\)                                  | Homomorphic tallying |
| Pret-a-Voter HRPAT [33] | Localisable, public (see \$2.3\)       | Aggregate                                  | Mixnet  |
| STAR-vote [9]           | Localisable\(^1\), public (see \$2.3\) | N/A\(^1\)                                  | Homomorphic tallying |
| Essex et al. [24]       | Localisable, public (see \$2.3\)       | Aggregate                                  | Mixnet  |
| OpenVoting              | Localisable, public                    | Localisable                                | ZKPs of set membership and reverse set membership |

\(^1\)Individual encrypted votes are never decrypted.

\(^2\)Although STAR-vote is based on homomorphic tallying, it shows a one-to-one correspondence with paper records by injecting the encrypted records to an independent shuffler and decryptor subsystem.

Figure 2: Comparison with some existing E2E-V protocols that support paper audit trails.

Most hand-marked ballot schemes such as Scratch & Vote [2], Punchscan [23] and Prêt à voter [41] do not support VVPR. Some systems such as Scantegrity I [19] and II [18] do maintain a VVPR audit trail but the VVPR can be trivially linked with the voter receipt to reveal their vote. Hence the VVPR audit cannot be performed publicly.

Some protocols [11, 25, 39] maintain VVPR audit trails, but they are independent and decoupled. Thus, although VVPR counts can be compared with the electronic count, mismatches in aggregate counts cannot be localised to individual problematic votes.

Localising VVPR count mismatches requires establishing a one-to-one correspondence between VVPVs and electronic votes. Homomorphic tallying backends [9, 25] are fundamentally incompatible with this requirement, because individual encrypted votes are never decrypted. Mixnet-based backends can support this if unique identifiers can be attached to cleartext votes at the mixnet output that can be matched with the VVPR but not the voter’s receipt.

A few systems [9, 24, 33] do provide such one-to-one correspondence but they have their limitations. In [33] the VVPR identifier needs to be decrypted during audit to find the corresponding electronic record. In [9] the encrypted votes need to be fully decrypted thereby defeating the purpose of homomorphic encryption. [24] depends on specially designed invisible inks and pens which has its own verifiability and practicality concerns. We summarise this discussion in Table 2.

### 2.4 Risk-limiting audits

Risk-limiting audits (RLAs) verify that the winners declared by the electronic voting system match those that would be declared by a full hand count of the paper audit trail. The audit consists of repeated checking of paper ballots and continuing until either this fact is established beyond reasonable doubt or a full hand-count is performed to correct the winners reported by the electronic count.

All RLAs assume that the paper audit trail is secure. They have traditionally provided verifiability to non-E2E-V systems [12, 32, 36, 43]. For E2E-V systems, replacing the publicly verifiable electronic count with the paper count is questionable.

The traditional definition of recoverability is called strong software independence which demands that a detected change or error in an election outcome (due to a change or error in the software) can be corrected without re-running the election [40]. However, “correcting” errors without re-running the election requires a ground truth, which is usually assumed to be the paper audit trail. In contrast, we do not wish to declare some election output as a ground truth, but rather identify inconsistencies between election outputs and narrow them down to individual votes, so that case-specific recovery procedures can be triggered.

### 3 DESIGN OBJECTIVES AND THREAT MODEL

The overall correctness of voting is verified by the correctness of three steps: cast-as-intended indicating that the vote has been registered correctly, recorded-as-cast indicating the cast vote is correctly included in the final tally, and counted-as-recorded indicating that the final tally is correctly computed [13]. Recorded-as-intended is a composition of the first two and counted-as-intended is a composition of all three. Spurious votes are fake votes not certified by POs.

We consider a typical polling booth dual voting protocol executed by a set of voters using BMDs, a set of EAs, and a set of POs. The voting protocol consists of the following standard sub-protocols: a ballot generation protocol executed by the EAs; a public ballot audit protocol before polling starts; a vote casting protocol executed by the voter (using a BMD) and a PO, which produces a receipt for the voter, a VVPR for a physical ballot box and an electronic version of the cast vote for tallying; a vote processing protocol executed by the EAs to obtain the final tally; a receipt verification protocol for verifying voter receipts; a VVPR verification protocol for verifying VVPVs; and a universal verification protocol for establishing the overall correctness of the election. We require that the verifiability and secrecy properties defined below should hold.

#### 3.1 Verifiability

**Definition 1 (Verifiability).** Suppose that a polynomially bounded adversary can manipulate the BMDs and controls all EAs, all POs and an arbitrary set of voters. Also suppose that a statistically significant number of random ballots pass the ballot audit.

1. Receipt verifiability: The event that a voter’s receipt passes the receipt verification protocol but her electronic vote is not recorded as intended [13] can happen with only a bounded small probability.
(2) VVPR verifiability: The event that a VVPR passes the VVPR verification protocol but was not cast as intended [13] can happen with only a bounded small probability.

(3) Universal verifiability: The event that a statistically significant number of randomly chosen receipts pass the receipt verification protocol and a statistically significant number of randomly chosen VVPRs (as per RLA [13]) pass the VVPR verification protocol, yet an electronic vote is not counted as intended or some VVPR was not cast as intended can happen only with a bounded small probability.

(4) No spurious vote injection: We also require that no entity other than the PO should be able to introduce a spurious electronic vote without failing universal verification. (Trust on POS for identity and eligibility verification—which is usually carried out in public in the presence of candidates’ representatives—is generally unavoidable.)

3.2 Secrecy
Definition 2 (Vote secrecy). Suppose that a polynomially bounded adversary controls less than a threshold number of EAs, all POS, and can try to coerce an arbitrary set of voters. A voting protocol is said to protect vote secrecy if the adversary does not learn any information beyond the final tally, under the following assumptions:

1. ballot printing is done by a trusted printer controlled by a ballot printing authority; each EA has an independent secure channel with the printer and the printed ballots are covered under tamper-evident seals;
2. ballot boxes are perfectly sealed and are opened only in aggregate after mixing ballot boxes from all polling booths (this is required to protect the secrecy of polling booth level vote statistics);
3. voters are bare-handed and cannot read numbers encoded in QR codes;
4. the EAs do not deliberately cause verification failures of electronic records to extract voting information because then they can be detected.

3.3 Recoverability
In case verification fails, we require a well defined recovery protocol that can narrow down the failures to individual failing items and polling booths. In addition, it should be possible to identify the parties responsible for the failures, wherever possible, without compromising individual voting privacy. In case the number of receipts or VVPRs that fail to verify are more than the margin of winning votes, and it is decided to have re-election only in the offending polling booths, then the voting statistics of those subset of polling booths cannot be kept secret and can be easily computed using a differential analysis.

4 PRELIMINARIES
In this section, we discuss notation and some key cryptographic primitives we need in the OpenVoting protocol.

4.1 Notation
We let $m$ denote the number of candidates, $n$ denote the number of voters and $\alpha$ denote the number of EAs. $G_1, G_2, G_T$ denote cyclic groups of prime order $q (q \gg m, n)$ such that they admit an efficiently computable bilinear map $e : G_1 \times G_2 \rightarrow G_T$, i.e., for all $a, b \in \mathbb{Z}_q$ and generators $g_1, g_2$ of $G_1$ and $G_2$ respectively, $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$ and $e(g_1, g_2) \neq 1_{G_T}$, where $1_{G_T}$ denotes the identity element of $G_T$. We assume that the n-Strong Diffie Hellman assumption [14] holds in $(G_1, G_2)$ and that the discrete logarithm problem is hard in $G_1$. We let $f_1, g_1, h_1 \in G_1, f_2, g_2 \in G_2$ and $f_T \in G_T$ denote randomly chosen generators of groups $G_1, G_2$ and $G_T$ respectively, $[x]$ denote the set $\{0, 1, ..., x - 1\}$, $x \triangleleft S$ denote that $x$ is drawn randomly from set $S$, and $\text{Perm}(n)$ denote the set of permutations $\pi : [n] \rightarrow [n]$. We use $\ell' \triangleq \text{PermuteReEnc}(\ell, \pi)$ to denote that an input list $\ell$ of ciphertexts (under a homomorphic encryption scheme) is permuted by $\pi$ and each ciphertext is re-encrypted to obtain $\ell'$ (e.g., for ElGamal encryption scheme, re-encryption of a ciphertext $c$ can be done as $c' \leftarrow c \cdot \text{Enc}(pk, 1)$).

4.2 Pedersen commitments
Given a message $m \in \mathbb{Z}_q, C = g_1^m h_T^r$ is called a Pedersen commitment [38] to $m$ with secret randomness $r \in \mathbb{Z}_q$. Pedersen commitments are a) perfectly hiding: C information-theoretically hides $m$; b) computationally binding: it is hard to compute $(m_1, r_1) \neq (m_2, r_2)$ such that $C = g_1^{m_1} h_T^{r_1} = g_1^{m_2} h_T^{r_2}$; and c) additively homomorphic: given $C_1 = g_1^{m_1} h_T^{r_1}$ and $C_2 = g_1^{m_2} h_T^{r_2}$, $C_1 C_2 = g_1^{m_1 + m_2} h_T^{r_1 + r_2}$ is a commitment to $m_1 + m_2$ with randomness $r_1 + r_2$.

4.3 Signature schemes
In the following, let $x$ denote a signer’s secret key, $y$ denote its public key, and $m$ denote a message to be signed.

Definition 3 (Boneh-Boyen (BB) signatures - basic [14]). A Boneh-Boyen signature scheme is given by algorithms (KeyGen, Sign, Verify):

\[ \begin{align*}
\text{KeyGen:} & \quad x \triangleleft \mathbb{Z}_q^n; y = g_2^x. \\
\text{Sign}(m, x): & \quad \sigma \leftarrow g_1^m x^r. \\
\text{Verify}(\sigma, m, y): & \quad \text{accept if } e(e(\sigma, y), g_2^m) = e(g_1, g_2)^{mr}. 
\end{align*} \]

Definition 4 (BBS+ signatures [5]). A BBS+ signature scheme is given by algorithms (KeyGen, Sign, Verify):

\[ \begin{align*}
\text{KeyGen:} & \quad x \triangleleft \mathbb{Z}_q^n; y \leftarrow f_2^x. \\
\text{Sign}(m, x), c, r_r \leftarrow \mathbb{Z}_q; S = (f_1 g_1^m h_T^r)^{t_{Z_T}}. \text{Output } \sigma \leftarrow (S, c, r). \\
\text{Verify}(\sigma = (S, c, r), m, y): & \quad \text{accept if } e(S, y f_2^r) = e(f_1 g_1^m h_T^r, f_2). 
\end{align*} \]

4.3.1 Obtaining signatures on committed values. An interesting property about BBS+ signatures is that given a Pedersen commitment $C = g_1^m h_T^r$, a signature on the committed value $m$ can be obtained by sending only commitment $C$ and a zero-knowledge proof of knowledge (ZKPoK) of the committed value to the signer:

(1) The committer sends $C$ and a ZKPoK of $m, r$ s.t. $C = g_1^m h_T^r$ to the signer. The signer verifies the ZKPoK.

(2) The signer computes a quasi signature by choosing $c, r' \triangleleft \mathbb{Z}_q$ and computing $S \leftarrow (f_1 h_T^r C)^{t_{Z_T}}$. It sends $\sigma' := (S, c, r')$ to the committer.

(3) The committer obtains $\sigma' \leftarrow (S, c, r' + r)$. Note that $\sigma' = ((f_1 h_T^{r'} g_1^m)^{t_{Z_T}}, c, r' + r)$ is a valid BBS+ signature on message $m$ under the signer’s public key $y = f_2^x$. 

Prashant Agrawal, Kabir Tomer, Abhinav Nakarmi, Mahabir Prasad Jhanwar, Subodh Sharma, and Subhashis Banerjee
4.4 Threshold primitives

4.4.1 Multiplicative secret sharing. A multiplicative secret sharing scheme [20] is a secret sharing scheme with the following property: if $x^{(a)}$ denotes the share of a secret $x$ held by party $a$, then 1) $x^{(a)} + y^{(a)} = (x + y)^{(a)}$, and 2) there exists an algorithm Mult such that $(xy)^{(a)} = \text{Mult}(x^{(a)}, y^{(a)})$. A standard trick due to Beaver [8] then converts any additive secret sharing scheme to a multiplicative secret sharing scheme using an offline input-independent precomputation step to share special multiplication triples.

4.4.2 Threshold decryption. Threshold decryption allows multiple parties to hold shares of a decryption key and jointly decrypt a ciphertext such that each party learns the decrypted message but no one learns any additional information. Given that each party $a \in \{x\}$ holds a share $sk^{(a)}$ of secret key $sk$, we use Beaver’s trick (see Section 4.4.1) such that the multiplicative protocol are with respect to EA public keys $p_k$.

These proofs can be easily given using standard $\Sigma$-protocol techniques, as long as each prover knows a share of each of the $G_1, G_2$ or $G_T$. These proofs can be easily given using standard $\Sigma$-protocol techniques, as long as each prover knows a share of each of the group generators with respect to each other.

5 THE OPENVOTING PROTOCOL

5.1 Setup

Before the election begins, some public election parameters need to be initialised during the setup phase in a public ceremony. During this setup, group generators $f_1, f_2, f_1$ are generated such that nobody knows the discrete logarithms of the generators with respect to each other. The number of voters $n$, the number of candidates $m$ and the number of EAs $\alpha$ are decided. An official candidate list is created such that for $i \in \{0, \ldots, m - 1\}$, $cand_i$ denotes the name of the $i$th candidate.

A multiplicatively homomorphic encryption scheme $E_{\mathbb{G}, \mathbb{G}} := (\text{KeyGen}_{\mathbb{G}, \mathbb{G}}, \text{Enc}_{\mathbb{G}, \mathbb{G}}, \text{Dec}_{\mathbb{G}, \mathbb{G}})$ in group $G_1$ and an additively homomorphic encryption scheme $E_{\mathbb{G}, \mathbb{G}} := (\text{KeyGen}_{\mathbb{G}, \mathbb{G}}, \text{Enc}_{\mathbb{G}, \mathbb{G}}, \text{Dec}_{\mathbb{G}, \mathbb{G}})$ in group $G_q$ are decided. For $E_{\mathbb{G}, \mathbb{G}}$, we choose the ElGamal encryption scheme [22] in group $G_1$. However, instantiating $E_{\mathbb{G}, \mathbb{G}}$ in a prime order group $G_q$ is non-trivial. Nevertheless, in Appendix A, we suggest a restricted instantiation of $E_{\mathbb{G}, \mathbb{G}}$ that serves our purpose. All encryptions in our protocol are with respect to EA public keys $pk_{\mathbb{G}, \mathbb{G}}$, for $\mathbb{G} \in \{\mathbb{G}, \mathbb{G}\}$. Thus, we abbreviate $\text{Enc}_{\mathbb{G}, \mathbb{G}}(pk_{\mathbb{G}, \mathbb{G}}, m)$ by $\text{Enc}_{\mathbb{G}}(m)$. The corresponding secret keys $sk_{\mathbb{G}, \mathbb{G}}$ are shared among the EAs.

We also need to instantiate a multiplicative secret sharing scheme, for which we use Beaver’s trick (see Section 4.4.1) such that the multiplicative triples are shared between the EAs before the election.

5.2 Ballot design

Our ballot is a variant of a Scratch & Vote ballot [2], adapted for VVPR and BMD support (see Figure 3 - left). It has two halves attached to each other through a perforated line. The left half is designed to be the VVPR and the right half is designed to be the voter receipt such that the two halves are not linkable to each other post vote casting.

The left half, i.e., the VVPR, contains a ballot identifier $u$ randomly drawn from $Z_q$. The candidate list printed on this half in a circular rotation of the official candidate list. Specifically, the candidate name printed on row $u$ is $\text{cand}_{ud}$, where $u \in [302]$, the candidate printed on row $u = 1$ is $\text{cand}_{d102+1}$.

The right half, i.e., the voter receipt, contains, for $w = 0$ to $m - 1$, Pedersen commitments $C_w = g^{u_w}h^w$ of values $u_w = u + w$ with randomnesses $r_w \in Z_q$. We call values $v_w \in Z_q$ the extended votes and values $\tilde{v}_w := v_w \mod m$ as the raw votes. The commitment randomnesses $r_w$’s are secret and they are put under a detachable scratch surface in the ballot. A separate identifier, denoted as $b_{id}$, for the voter receipt is computed as follows: $b_{id} = \text{Hash}(C_u || \cdots || C_{m-1})$.

Both left and right halves contain a designated gray area at the top. The voter’s chosen $w$ is printed on both halves in this area by the BMD during voting (see Section 5.5). In addition, the right half also contains a polling booth identifier $k$ for the ballot’s designated polling booth, whereas the left half contains its commitment $C_k = g^{k}h^k$ (which is stored along with $r_w$’s). This is used for identifying polling booths during the recovery protocol (see Section 5.8).

Both left and right halves also need to contain signatures of the ballot printing authority for authenticity, but we do not explicitly show them. All ballots must arrive at the polling booth covered under a tamper-evident seal, to protect ballot secrecy. The publicly visible ballot cover has a copy of the $b_{id}$.

5.3 Ballot generation

Ballots are generated by EAs $A^{(0)}, \ldots, A^{(\alpha-1)}$ in a distributed fashion. Each EA only holds shares of ballot secrets $\{u, \{v_w\}_{w=0}^{m-1}, \{r_w\}_{w=0}^{m-1}\}$. For ballot printing, we assume that a trusted printer controlled by a ballot printing authority prints the left and right halves securely. We assume that each EA has a secure communication channel with the printer, and we trust the printer not to leak information. The ballot printing and automatic sealing must happen in public to ensure that no ballot information leaks during printing. Figure 4 shows the ballot generation steps in detail.

The ballot id $(b_{id})$ generated by the printer is signed with the ballot printing authority’s signature and also printed on top of sealed ballot.
We require audits of some randomly picked ballots in each polling public. The remaining sealed ballots must then remain in public ceremonies involving representatives of candidates and general public. The remaining sealed ballots must then remain in public view during polling. During a ballot audit, \( n_a \) random ballots are sampled and for each ballot, its sealed cover is opened, secrets hidden under the scratch surface are opened and the following checks are performed:

1. For \( w = 0 \) to \( m - 1 \), \( C_w \equiv g_0^{k+w} r_w^a \mod m \).
2. For \( w = 0 \) to \( m - 1 \), the candidate name at row \( w \) is \( \text{cand}_u \equiv w \mod m \).
3. \( C_k \equiv g_1^a t_k \).
4. \( b_{id} \) outside the ballot cover and on the ballot right half match.
5. All signatures on the ballot are valid.

Since the secrets of audited ballots are revealed, audited ballots cannot be used for vote casting and must be spoiled.

Assuming that at least \( n_f \) out of \( n \) ballots are incorrectly formed and \( \epsilon \) is the maximum acceptable probability of not detecting an incorrect ballot, \( n_a \) should be chosen such that \( \left( \frac{n_f}{n} \right)^{n_a} \leq \epsilon \).

5.5 Vote casting

Vote casting happens as follows (see Figure 3 - center):

5.5.1 Ballot pick-up and eligibility verification: The voter picks up a random covered ballot from a set of ballots kept at the polling booth (see Section 7.4 for a caveat). The PO verifies voter’s eligibility, scans \( b_{id} \) from the ballot cover, and allows the voter to proceed to a private enclosure containing a BMD.

5.5.2 Vote casting: Inside the vote casting booth, the voter feeds the top gray region of the ballot to the BMD and presses a button on the onscreen display to select \( w \) corresponding to her preferred candidate. The BMD can only access the top gray region for printing and cannot read any part of the ballot (it should not have any attached scanner or camera). The BMD prints the voter’s chosen \( w \), which we denote by \( \hat{w} \), on both the left and right halves of this gray region.

The voter needs to verify that indeed her intended choice is printed on both the halves. If satisfied, the voter feeds the ballot into a ballot box which separates the two halves, gives the right
Common input: \(((b_{\text{id}}, w_0), \ldots, (b_{\text{id}}, w_{m-1})\) from BB₁

\(A^{(a)}\) input: \(((b_{\text{id}}, w_{\text{voter}}), \ldots, (b_{\text{id}}, w_{\text{voter}}), (b_{\text{EAs}}, w_{\text{EAs}})\)

**Stage 1 (Obtain encrypted votes):**
- Retrieve \(e_i^{(a)}\) for \(i \in [0, n-1]\) using \((b_{\text{id}}, w_0), \ldots, (b_{\text{id}}, w_{n-1})\)
- \(e_i^{(a)} \leftarrow \text{Enc}_{\text{EAs}}(e^{(a)})\)
- Broadcast \(e_0^{(a)}, \ldots, e_n^{(a)}\)

**Stage 2 (Permute and re-encrypt):**
- \(A^{(0)}\) receive \(E^{(a)}\) from \(A^{(a)}\)
- \(E^{(a)} \leftarrow \text{PermuteReEnc}(E, \pi^{(a)})\)
- Send \(E^{(a)}\) to \(A^{(1)}\)
- \(A^{(0)}\) receive \(E^{(a)}\) from \(A^{(a)}\)
- \(E^{(a)} \leftarrow \text{PermuteReEnc}(E^{(a)}), \pi^{(a)}\)
- Send \(E^{(a)}\) to \(A^{(1)}\)

**Stage 3 (Threshold decryption):**
- \(A^{(a)}\) receive \(E^{(a)}\) from \(A^{(a)}\)
- \(E^{(a)} \leftarrow \text{PermuteReEnc}(E^{(a)}), \pi^{(a)}\)
- Broadcast \(E^{(a)}\)
- Where \(\pi \equiv \pi^{(a)} \oplus \pi^{(a)} \oplus \pi^{(a)}\)

Notation: \(A^{(a)}\) denotes the steps that the right hand are executed one-by-one by EAs \(A^{(a)}\) through \(A^{(a)}\). We use \(f'\) to denote that an input list \(f\) is re-encrypted to obtain \(f'\).

**Figure 5: Vote processing protocol.**

half back to the voter and accepts the left half. The voter takes the right half to the PO for scanning. If not satisfied, the voter shreds the marked ballot and raises a dispute. In this case, the voter is allowed to re-vote.

The ballot box should be a purely mechanical device with no electronic component.

### 5.5.3 Receipt scanning
The PO scans the voter’s receipt and verifies if the ballot identifier \(b_{\text{id}}\) in the receipt matches the one scanned in Section 5.5.1, to ensure that a fresh sealed ballot is used, and verifies the ballot printing authority’s signature, for ballot authenticity. It ensures that the scratch surface is intact and that the ballot secrets are not compromised and shreds the scratch region in front of the voter. From the scanned receipt, the PO extracts \(C_w\) and uploads \((V_{\text{id}}, b_{\text{id}}, w, C_w)\) to BB₂ along with \(\sigma_{\text{id}} = \text{Sign}\_\text{sk}_\text{BBO}(V_{\text{id}} || b_{\text{id}} || w || C_w)\) and the ballot printing authority’s signature on \(b_{\text{id}}\). The PO affixes the \(V_{\text{id}}\) to the receipt part, stamps it, and returns it to the voter.

We denote the voter’s selected \(w\) on a ballot with identifier \(b_{\text{id}}\) by \(\hat{w}\). For notational convenience, we define
\[
\hat{v} = v_{\text{w}}, \quad \hat{r} = r_{\text{w}}, \quad \hat{C} = C_w
\]

**5.6 Vote processing**

Post polling, the EAs read from BB₁ the tuples \(((b_{\text{id}}, \hat{w}_0), \ldots, (b_{\text{id}}, \hat{w}_{m-1})\) of ballot identifiers and voter selections to obtain clearvote texts on BB₂ (see Figure 3 - right). Each \(A^{(a)}\) uses the voter’s choice \(\hat{w}_i\) in ballot \(b_{\text{id}}\) to fetch the share \(\hat{v}_i = v_{\text{w}_i}\) of \(\hat{v}_i = v_{\text{w}_i}\) from its local storage. In stage 1, the EAs combine their shares homomorphically in encrypted form to obtain \((\text{Enc}_{\text{EAs}}(\hat{v}_j))_{j=0}^{n-1}\). Stages 2 and 3 are then identical to a re-encryption mixnet where each EA starting from \(A^{(0)}\) through \(A^{(a-1)}\) permutes its list of input ciphertexts using a secret permutation \(\pi^{(a)}\) and re-encrypts the output ciphertexts, and at the end there is a decryption stage where the permuted encrypted votes are threshold decrypted. The obtained cleartext extended votes \((\hat{v}_1, \ldots, \hat{v}_{n-1})\), where \(\pi \equiv \pi^{(a-1)} \oplus \pi^{(a)}\), are published on BB₂ along with raw votes \((\hat{v}_{n-1} \mod m, \ldots, \hat{v}_1 \mod m)\). Figure 5 shows the detailed steps.

It is guaranteed that up to \(a - 1\) colluding EAs do not learn any information beyond the set of extended cleartext votes \((\hat{v}_0, \ldots, \hat{v}_{n-1})\). In particular, they do not learn which \(\hat{C}_i\) on BB₁ commits which entry on BB₂.

The VVPRs from each polling booth’s ballot box are collected and mixed in a central facility. VVPRs are revealed to the public only after this mixing phase, to avoid leaking polling booth-level voting statistics.

### 5.7 Verification

Our verification protocol depends on our ZKPs of set membership and reverse set membership (see Section 6). The preprocessing steps required for the ZKPs should be set up independently by parties with the biggest stake in the outcome, for example by the political parties that the candidates represent. Our verification protocol consists of the following components.

#### 5.7.1 Receipt verification

1. Identify the voter receipt on BB₁ using the \(V_{\text{id}}\) as index and verify that the \(b_{\text{id}}\) and the commitment \(C_w\) corresponding to the choice \(w\) on the receipt are recorded correctly.
2. Verify ZKP of set membership (see Section 6.1.2): verify, in zero knowledge, that the commitment \(C_w\) on BB₁ commits one of the extended cleartext votes on BB₂.

#### 5.7.2 VVPR verification

1. Identify the VVPR on BB₂ using \(u + w\) as index and verify that the candidate name printed at row \(w\) in the VVPR is \(\text{candidate}_w \mod m\).
2. Verify ZKP of reverse set membership (see Section 6.2.2): verify, in zero knowledge, that there exists a commitment on BB₁ that commits \(u + w\) on BB₂.

#### 5.7.3 Universal verification

1. Verify that the sizes of BB₁ and BB₂ are equal, there are no duplicate entries and all signatures on BB₁ are correct.
2. Verify ZKP of set membership for every entry on BB₁ and ZKP of reverse set membership for every entry on BB₂.
3. Compute the tally on BB₂. Let \(n_w\) be the winning margin. Hence \(n_w\) number of false entries can change the election outcome. Fix \(\epsilon'\) as the maximum probability with which a false outcome may be accepted. Ensure that a minimum of \(n_w\) number of receipts and VVPRs are verified such that \(\frac{n_w}{\pi} \leq \epsilon'\). Once a VVPR is identified on BB₂ set a ’checked’ flag on BB₂.
5.8 Recovery protocol

If verifications fail, the following cases arise:

1. A voter receipt does not match any encrypted vote on $BB_1$: If the receipt is correctly stamped and signed, then the PO is at fault, otherwise the voter is at fault. The polling booth identifier can be noted from the receipt.

2. An encrypted vote on $BB_1$ fails the ZKP of set membership: If the ballot printing authority’s signature can be verified and the PO used an unofficial ballot that the EAs cannot decrypt. Otherwise, it is the EAs’ fault (EAs did not decrypt this vote correctly). All entries on $BB_1$ identify the polling booths.

3. A cleartext vote on $BB_2$ fails the ZKP of reverse set membership: The EAs are at fault (the EAs added a spurious vote). No polling booth is involved.

4. A VVPR does not match with any cleartext vote on $BB_2$: This can happen in several ways - i) a voter may introduce a VVPR from a ballot with incorrect $b_{id}$ into the ballot box, but the PO would not sign and upload the corresponding right half, ii) a spurious VVPR may be introduced anywhere in the custody chain and iii) a PO may not upload the scanned receipt for a genuinely cast vote. It is impossible to distinguish these cases, and this precisely is the reason that putting entire trust on VVPRs and RL without cryptographic checks is risky. The polling booth corresponding to the VVPR (but not necessarily where it has been injected into the ballot box) can be identified by asking the EAs to supply the opening $k, r_k$ of the commitment $C_k$ on the VVPR and verifying it.

If the number of failures is smaller than the winning margin, they may simply be ignored. Otherwise, local re-polling only in the offending booths may be considered for recovery.

6 ZKP OF SET MEMBERSHIP AND REVERSE SET MEMBERSHIP

We now describe our zero-knowledge proofs of set membership and reverse set membership (see Section 5.6). Although our ZKPs are distributed proofs of knowledge, we begin each ZKP by describing the case when there is only a single authority $A$ that acts as the prover. $A$ knows the openings $o, r$ for each commitment on $BB_1$ and also the secret permutation $\pi$ between $BB_1$ and $BB_2$.

6.1 ZKP of set membership

6.1.1 Single prover case. In our context, a ZKP of set membership proves that a given commitment $C$ on $BB_1$ commits a $v \in \phi$, where $\phi$ is the set of extended votes published on $BB_2$ (see Figure 3 - right). Since commitments are binding, this is equivalent to proving knowledge of one opening $o, r$ such that $C = g^o h^r_i \land v \in \phi$, i.e., $PK\{(o, r) : C = g^o h^r_i \land v \in \phi\}$. For the single prover case, a scheme proposed by Camenisch et al. [16] proves exactly this.

The main idea behind this scheme is that the verifier sends to the prover Boneh-Boyen signatures [14] on elements of set $\phi$ under a fresh signing key. Then the prover can prove that $C$ commits a member of the set by proving that it knows a signature on the value committed by $C$. This can be done in zero knowledge by revealing only a blinded signature to the verifier and proving knowledge of appropriate blinding factors from which a valid signature can be obtained. If $C$ does not commit a member of $\phi$ then the proof fails because the prover does not know signatures on non-members of the set. See Figure 11 in Appendix B for the detailed protocol.

The scheme is an honest-verifier ZKP of set membership if $|\phi|$ - Strong Diffie Hellman assumption holds in $(G_1, G_2)$ [16]. A nice property of this scheme is that if proofs for multiple commitments are requested against the same set, the verifier’s signatures can be reused. The blinded verifier signatures can be pre-published next to each commitment on $BB_1$ using a secret permutation, resulting in an $O(1)$ online overhead per commitment and thus an $O(n)$ amortised complexity. In contrast, a naïve scheme based on the generic OR construction $(PK\{(o, r) : C = g^o h^r_i \land (v = \phi_0 v \cdots v = \phi_{n-1})\})$, where $\phi_i \in \phi$, is $O(n^2)$.

6.1.2 Distributed case. As in the single prover case, the distributed ZKP is also split into two stages: a preprocessing stage to obtain signatures (stage 1) and an online stage to prove knowledge of signatures (stage 2). In stage 1, the single prover can obtain verifier signatures on extended votes published on $BB_2$ and use its secret permutation to publish blinded signatures next to each commitment on $BB_1$. However, in the distributed case, no single EA knows the full permutation between $BB_1$ and $BB_2$ and they need to obtain blinded signatures on $BB_1$ without learning this permutation. This is achieved as follows. The verifier publishes its signatures next to each extended vote on $BB_2$. Each Each $A_i^{(a)}$ then computes encryptions/re-encryptions of these signatures and applies the inverse of permutation $\pi^{(a)}$ to them in reverse order from $A^{(a)}$ to $A^{(0)}$, where $\pi^{(a)}$ is the permutation they applied during vote processing (Figure 6 - left). Thus the encrypted signatures output by $A^{(0)}$ come out in the same order as that of the published commitments on $BB_1$. Now the homomorphism of the encryption scheme is used to compute encryptions of blindings of these signatures, with each authority contributing their blinding factors (Figure 6 - center). The blinded signatures are finally decrypted in threshold manner (Figure 6 - right).

In stage 2, the blinded signature corresponding to a commitment $C_i$ on $BB_1$ is looked up in $O(1)$ time. Then, a distributed proof of knowledge of a signature on the value committed by $C_i$ is jointly carried out by each $EA_i^{(a)}$ using shares $\delta^{(a)}_i, r^{(a)}_i$ and blinding factors $b^{(a)}_i$. The amortised complexity of proving set membership for $n$ commitments is retained at $O(n)$. See Figure 7 for the detailed protocol.

6.2 ZKP of reverse set membership

6.2.1 Single prover case. In our protocol we also need a ZKP for the converse of above, i.e., to prove in zero knowledge that an extended vote $v$ on $BB_2$ is committed by a $C \in \Phi$, where $\Phi$ is the set of commitments published on $BB_1$. In what follows we present a novel scheme following the basic idea of signature-based set membership ZKPs.

First note that the Boneh-Boyen signatures used in the ZKP of set membership require messages to be in group $\mathbb{Z}_p$ and therefore cannot be directly used for signing Pedersen commitments. Recall, however, from Section 4.3.1 that the BBS+ signature scheme [6] allows a committer to present a commitment to the signer and
obtain a signature on the committed value. Thus, we let the reverse set membership verifier act as the signer that sends quasi-BBS+ signatures for each $\tilde{C}_i \in \Phi$ after verifying that the prover knows the opening of $\tilde{C}_i$. By the property of BBS+ signatures, the prover can thus obtain valid BBS+ signatures on values committed by each $\tilde{C}_i \in \Phi$. To prove that a given extended vote $v_0$ is committed by some commitment in $\Phi$, it cannot prove successfully if no commitment in $\Phi$ committed $v$. As in the set membership case, the proof of knowledge of the required signature is given by showing knowledge of blinding factors that give a valid BBS+ signature, where each component $(S, c, r)$ of the BBS+ signature is blinded independently. This protocol also enjoys the $O(1)$ online complexity because signatures can be reused. See Figure 12 in Appendix B for the detailed protocol.
6.2.2 Distributed case. In the distributed case, we follow similarly as in Section 6.1.2 above to extend the single prover case. Figure 8 shows a high-level description of the preprocessing stage and Figure 9 shows the detailed protocol.

In stage 1, a distributed proof of knowledge of the opening of each $C_i \in \Phi$ is given using standard threshold techniques (see Section 4.4.3).

In stage 2, the verifier provides quasi BBS+ signatures against commitments $C_i$ as in the single prover case, but now instead of obtaining a valid BBS+ signature on $\tilde{v}_i$ in the clear, which would leak the commitment randomness $\tilde{r}_i$ to the verifier, encryptions of BBS+ signatures are created instead. This is done by the EAs first obtaining encryptions of $\tilde{r}_i$ by homomorphically combining their shares (Figure 8 - top left) and then adding them homomorphically to encryptions of the $r'$ component of the quasi BBS+ signatures. The obtained encrypted BBS+ signatures (Figure 8 - top right) are then permuted and re-encrypted similarly to Section 6.1.2, but in the forward direction this time, to obtain encrypted signatures in the same permuted order as votes published on $BB_2$. Since the BBS+ signature is a tuple of the form $(S, c, r)$, we perform encryption of each component separately using the homomorphic encryption scheme appropriate for that component ($E_d$ for $S$ and $E_\phi$ for $c, r$). The encrypted signatures (each of their components) are then homomorphically blinded (Figure 8 - bottom left) and threshold decrypted (Figure 8 - bottom right) as in Section 6.1.2.

In stage 3, a proof of knowledge of a signature $({\bar{x}}_{r_{1}^{-1}(j)}, {\bar{x}}_{r_{2}^{-1}(j)}, \tilde{r}_{2}^{-1}(j))$ on extended vote $\tilde{v}_{2}^{-1}(j)$ on $BB_2$ is given using the published blinded signature $({\bar{x}}_{r_{1}^{-1}(j)}, {\bar{x}}_{r_{2}^{-1}(j)}, \tilde{r}_{2}^{-1}(j))$. The proof of knowledge here is a distributed proof of knowledge of the form $PK_1(\{x_1, \ldots, x_s\}) : \prod_{j=1}^s \frac{\tilde{g}_{j_{1}^{\tilde{h}_{1}}}}{g_{j_{1}^{\tilde{h}_{1}}}}$, for which efficient techniques are known (see Section 4.4.3). Recall that such distributed proofs of knowledge can be given if each prover has shares of the shared secrets. The shares for $b_{s_{1}}, b_{c_{1}}, b_{s_{2}}, \delta_{s_{2}}$ are directly available to each EA. Shares of $\delta_{s_{1}}$ and $\delta_{s_{2}}$ are obtained using the Mult algorithm of a multiplicative secret sharing scheme (see Section 4.4.1; also see the setup steps for Beaver multiplication triples in Section 5.1).

6.3 From honest-but-curious to general zero-knowledge

Note that the proofs of knowledge $DPK_0$ in Figure 7 and $DPK_1$ and $DPK_2$ in Figure 9 can be directly converted from honest-but-curious to general zero-knowledge in the random oracle model, using the Fiat-Shamir heuristic [27, 31].

Further, honest behaviour from the verifier can be enforced by verifying all its BB signatures and BBS+ quasi-signatures. Honest behavior from the EAs is enforced by making them commit to their commitment shares $C_i^{(a)}$ and permutations, and putting all encryptions produced by them to public bulletin boards. Then, either random audits can ask them to prove in ZK that they used correct encryptions in the preprocessing stage or in case of verification failures, they may be asked to reveal secrets for individual failing items.

7 SECURITY ANALYSIS

7.1 ZKPs of set membership and reverse set membership

Theorem 1. The protocol in Figure 7 is a distributed zero-knowledge proof of set membership, i.e., $DPK(\{\tilde{v}_i, \tilde{r}_i\}) : C_i = g_{1}^{r_i^{h_1}} \land \tilde{v}_i \in \Phi$.

Theorem 2. The protocol in Figure 9 is a distributed zero-knowledge proof of reverse set membership, i.e., $DPK(\{\tilde{r}_i\}) : g_{2}^{\tilde{r}_i^{h_1}} \in \Phi$.

See Appendices C and D for the proofs.

7.2 Verifiability

Theorem 3. The OpenVoting protocol satisfies receipt verifiability, VVPR verifiability and universal verifiability (see Definition 1).

Proof. The ballot audit process ensures that the probability of an incorrect ballot is at most $\epsilon$ (see Section 5.4). Thus, below we only consider the case of correctly formed ballots.

Receipt verifiability. For a receipt that passes verification, $C_v$ exists on $BB_1$. By the soundness of the ZKP of set membership (Theorem 1), the value $v$ committed by $C_v$ is correctly included in $BB_2$. By ballot design and the vote casting protocol, $\phi$ is the voter’s intended vote. Thus, the probability that a voter’s vote is not recorded as intended in $BB_2$ is at most $\epsilon$.

VVPR verifiability. Since the VVPR containing $(u, w)$ passes verification, a cleartext vote $u + w$ exists on $BB_2$. By Theorem 2, a commitment $C_u$ such that $C_u$ commits $u + w$ must exist in $BB_1$. Since all commitments on $BB_1$ are signed by some PO, so is $C_u$. According to the vote casting protocol, PO signs a receipt only when a voter produces the right half of a ballot marked with the voter’s choice. Hence the probability that a cleartext vote is not cast as intended is at most $\epsilon$.

Universal verifiability. According to the protocol and the above claims the probability that some votes are not recorded as intended and some VVPRs are not cast is intended at most $\epsilon’$ (see Section 5.7.3). Let us consider the cases such that the electronic votes are recorded as intended and the VVPRs are cast as intended yet the tally on $BB_2$ is wrong:

1. multiple commitments on $BB_1$ map to the same vote on $BB_2$
2. multiple copies of the same receipt given to different voters map to the same entry on $BB_1$
3. multiple VVPRs map to the same entry on $BB_2$
4. the same commitment on $BB_1$ maps to different entries on $BB_2$
5. the same receipt appears on multiple entries on $BB_1$
6. the same VVPR appears on multiple entries on $BB_2$

The set membership and reverse set membership ZKPs ensure elements on $BB_1$ and $BB_2$ are in one-to-one correspondence. Hence the first case is not possible. Since $BB_1$ contains the $V_{d8}$, they would mismatch and receipt verification would fail for some voters. The one-to-one correspondence check between VVPRs and entries on $BB_2$ would imply that the third case is not possible. A commitment can commit only one value, hence the fourth case is not possible. Duplicate checks on $BB_1$ and $BB_2$ ensures that no receipts or VVPRs can appear twice on them, ruling out the fifth and the sixth cases.

No spurious vote injection. The ZKP of reverse set membership ensures that each vote recorded on $BB_2$ is committed by some
commitment on \(BB_1\). Verification of PO signatures on \(BB_1\) ensures that no one except the POs can inject spurious votes. □

7.3 Secrecy

Lemma 1. A polynomially bounded adversary who controls all but one EA cannot learn anything beyond the (unordered) set of cast votes \(\{v_1, \ldots, v_n\}\) from the vote processing protocol.

See Appendix E for the proof.

Theorem 4. The OpenVoting protocol protects vote secrecy (see Definition 2) in the random oracle model provided there are no verification failures in ZKP of set membership and reverse set membership.

See Appendix F for the proof.

7.4 Randomisation attacks

Although our scheme hides the voter’s vote, like many paper ballot-based schemes [2, 41], it does not prevent a coercer from launching a randomisation attack where voters are asked to choose a fixed \(w\), thereby randomising their votes. This can be prevented by adding a detachable slip showing the candidate order on the ballot cover and allowing voters to choose ballots in a private room. Coerced voters could then choose a ballot so that they can vote for their preferred candidate while producing the required \(w\) for the coercer, and then detach the slip before proceeding to the PO (step 5.5.1 of Section 5.5).

8 PRACTICALITIES OF IMPLEMENTATION

We now present some illustrative performance benchmarks for the vote processing and verification protocols of OpenVoting. We implemented these protocols using the Charm cryptographic library [3], with a PBC library [34] backend. We chose the BN254 elliptic curve [7] to instantiate the pairing groups \((G_1, G_2)\). We instantiated \(E_D\) with the ElGamal encryption scheme in group \(G_1\) of the BN254 curve, whose order is of 254 bits. For \(E_D\), we used the additively homomorphic scheme described in Appendix A, and initialised its underlying Paillier encryption scheme with an unoptimised implementation for a 2048 bit Paillier modulus. We ran our benchmarks on an Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz with 46 GB of RAM and 104 cores. Table 10 shows space/time benchmarks for a dummy election with \(n = 10^6\) voters, \(m = 20\) candidates and \(\alpha = 4\) EAs.

We exploit the embarrassingly parallel nature of our ZKPs to distribute proof generation and verification among 100 cores. However, the signature-based nature of our ZKPs implies that they have to be given separately for each verifier. As the total verification takes about a few hours for the EAs per verifier, it may be feasible to support independent universal verification for only some major stakeholders (such as the political parties). Nevertheless, against the signatures supplied by these trusted verifiers in the preprocessing stage, anyone can verify a small subset of entries (e.g., for individual receipt/VVPR verification or for a statistical sampling-based verification of the one-to-one correspondence between \(BB_1\) and \(BB_2\)) rather efficiently (in less than a second per entry).

Ballot sizes are reasonable (modern QR codes can fit around 3 KB of data) and the sizes of \(BB_1\), \(BB_2\) and the ZKPs are moderate.

9 CONCLUSION

We have presented a voting protocol that is not only E2E-V but also recoverable from verification failures. The protocol’s verifiability, secrecy and recoverability properties have been formally established. The protocol is scalable, efficient and easy to implement.

REFERENCES

[1] Ben Adida and C. Andrew Neff. 2009. Efficient Receipt-free Ballot Casting Resistant to Covert Channels. In Proceedings of the 2009 Conference on Electronic Voting Technology/Workshop on Trustworthy Elections (Montreal, Canada) (EVT/WOTE ’09). USENIX Association, Berkeley, CA, USA, 11–11. http://dl.acm.org/citation.cfm?id=1855491.1855502

[2] Ben Adida and Ronald L. Rivest. 2006. Scratch & Vote: Self-contained Paper-based Cryptographic Voting. In Proceedings of the 5th ACM Workshop on Privacy in Electronic Society (Alexandria, Virginia, USA) (PWPES ’06). ACM, New York, NY, USA, 29–40. https://doi.org/10.1145/1179601.1179607
2.1: Obtain verifier quasi-signatures on votes committed by commitments

\[ \text{Stage 1 (for each } i \in \{0, \ldots, n-1\}: \]
\[ (A^{(a) \text{'}}, V) \]
\[ (\text{via standard techniques, with } A^{(a)}) \text{ using shares } \hat{e}_i^{(a)}, i^{(a)}(\text{a})) \]

2.2: Obtain encrypted (full) signatures from quasi-signatures

\[ (A^{(a) \text{'}}, V) \]
\[ \text{for } i = 0 \text{ to } n-1: \]
\[ c_i, r_i^\prime \leftarrow \mathbb{Z}_q \]
\[ S_i = (f_i h_i^{r_i^\prime}, y_i^{r_i^\prime}) \]
\[ \text{send } y_i, (c_i, r_i^\prime) \text{ to } A^{(1)} \]

2.3: Permute and re-encrypt signatures

\[ A^{(1)} \]
\[ \Sigma = (c_1, \ldots, c_n) \]
\[ \text{broadcast } \Sigma \text{ to } A^{(1)} \]
\[ (A^{(2)}, V) \]

3.1: Common calculation of public keys

\[ \text{Stage 3 (for given } j): \]
\[ (A^{(a)}, V) \]
\[ \text{for } i = 0 \text{ to } n-1: \]
\[ S_x^{(2)} (j) = \text{Dec}^{(a)}(\hat{e}_i^{(a) \text{'}}, r_i^\prime, (a_1^{(a)}), \text{Dec}(a_2^{(a)})) \]
\[ \text{broadcast } (\hat{e}_i^{(a)}, r_i^\prime) \text{ to } A^{(2)} \]

3.2: Calculate multiplicative shares

\[ \text{broadcast } \hat{s}_j^{(a)}, d_j^{(a)} \]

3.3: Proof of knowledge

\[ \hat{s} \leftarrow \Pi^{(a)}(\hat{a}, s_i^{(a)}) \]

Figure 9: Distributed proof of reverse set membership DPK{\{r \in \hat{a}^{(a)}, h_i^{r_i^\prime} \in \hat{a}\}} between verifier ‘V’ and provers (A^{(a)}) \text{ for } 0 \leq a \leq (n-1)

[3] Joseph A. Akinwale, Christina Garman, Ian Miers, Matthew W. Pagano, Michael Rushanan, Matthew Green, and Aviel D. Rubin. 2013. Charm: a framework for rapidly prototyping cryptosystems. Journal of Cryptographic Engineering 3, 2 (2013), 111–128. https://doi.org/10.1007/s13389-013-0057-3

[4] Nuttapong Attrapadung, Goichiro Hanaoka, Shigeo Mitsuani, Yasuaki Sakai, Kana Shimizu, and Tadanori Teruya. 2018. Efficient Two-Level Homomorphic Encryption in Prime-Order Bilinear Groups and A Fast Implementation in WebAssembly. In Proceedings of the 2018 on Asia Conference on Computer Security and Communications Security (Incheon, Republic of Korea) (ASIACCS ’18). Association for Computing Machinery, New York, NY, USA, 685–697. https://doi.org/10.1145/3196494.3196552

[5] Man Ho Au, Willy Susilo, and Yi Mu. 2006. Constant-Size Dynamic k-TAA. In Security and Cryptography for Networks, Roberto De Prisco and Moti Yung (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 111–125.

[6] Man Ho Au, Willy Susilo, and Yi Mu. 2006. Constant-Size Dynamic k-TAA. In Security and Cryptography for Networks, Roberto De Prisco and Moti Yung (Eds.).
Figure 10: Benchmarks for $n = 10^6$ voters, $m = 20$ candidates and $\alpha = 4$ EAs. Run on a single machine with 100 cores.
A RESTRICTED ADDITIVELY HOMOMORPHIC ENCRYPTION SCHEME IN A PRIME ORDER GROUP

Our scheme requires an additively homomorphic encryption scheme in a prime order group \( \mathbb{Z}_q \) (see Section 5.1). However, there appear to be only few initializations of such a scheme [4, 10, 17, 28], and these instantiations generally have specific constraints on message domains or are too inefficient for our use. Also, unlike existing homomorphic voting systems [2, 9], the messages we need to add homomorphically are not small numbers, so we cannot use the standard exponential El Gamal trick. Instead, we develop a simple solution that works for our use-case.

Recall that the Paillier encryption scheme [37] is additively homomorphic in a composite order group, with message space \( \mathbb{Z}_N \), where \( N \) is the composite Paillier modulus. We note that we require only a constant number of homomorphic additions (say \( \leq \mu \)). If we choose \( q \) to be such that \( \mu q < N \), then \( \text{Enc}_p(m_1) \cdot \text{Enc}_p(m_2) = \text{Enc}_p(m_1 + m_2 \mod N) = \text{Enc}_p(m_1 + m_2) \), where \( \text{Enc}_p \) denotes Paillier encryption. During decryption, \( m_1 + m_2 \mod q \) can be obtained simply by applying modulus \( q \) to the message \( m_1 + m_2 \) obtained by Paillier decryption. However, this scheme leaks \( m_1 + m_2 \) to the decryptors in addition to \( m_1 + m_2 \mod q \), which may have catastrophic consequences as, e.g., it could allow ruling out certain ranges of \( m_1, m_2 \) given \( m_1 + m_2 \). However, before decryption, if we homomorphically blind the encryption of \( m_1 + m_2 \) using a blinding factor \( b \ specificity. See the original document for the correct formatting and presentation of the mathematical expressions.
Common Input: \( \Phi = (C_0, \ldots, C_n) \)
A’s Input: \((v_i, r_i)\) for \(i = 0, \ldots, n\)

Stage 1 (Preprocessing):
1.1: Obtain verifier signatures on published votes

\[ V \]
\[ y = g_2^v \]
\[ (\sigma_{x_i}(0), \ldots, \sigma_{x_i}(n-1)) \rightarrow (g_1^{\sigma_{x_i}(0) \cdot \pi}, \ldots, g_1^{\sigma_{x_i}(n-1) \cdot \pi}) \]

\[ y \rightarrow \sigma_{x_i}(j) \rightarrow A \]

Stage 1 (for each \(i = 0, \ldots, n-1\))

\[ PK((v_i, r_i) : C_i = g_1^{h_i^1}) \]
A, V

Stage 2 (Preprocessing):
2.1: Obtain verifier quasi-signatures on votes committed by commitments

\[ x \rightarrow Z_q \]
\[ y \rightarrow f_{ij}^* \]

\[ (\pi, \gamma, \delta, \alpha, \beta_j, \beta_i) \]

Stage 2 (for each \(i = 0, \ldots, n-1\))

\[ PK((\beta_i, \beta_j) : \gamma_i = g_1^{h_i^1}) \]
A, V

Stage 3 (for each \(i = 0, \ldots, n-1\))

\[ PK((\beta_i, \beta_j) : \gamma_i = g_1^{h_i^1}) \]
A, V

Figure 11: Proof of set membership (PK\((\beta_i, \beta_j) : \gamma_i = g_1^{h_i^1} \land \delta_i \in \Phi\)) between verifier V and single prover A (due to [16])

Till \(\mu\) homomorphic additions, ciphertext \(c'\) encrypts a number less than \(m^* \prec \mu q + \alpha q\). Thus, \(m = (m^* \mod N) \mod q = m^* \mod q\), as expected.

Lemma 3 finally implies Theorem 5.

\[ \square \]

B ZKP OF SET MEMBERSHIP AND REVERSE SET MEMBERSHIP FOR THE SINGLE PROVER CASE
See Figures 11 and 12.

C PROOF FOR THEOREM 1
Proof. Note that in the distributed version, the verifier obtains the same transcript as that obtained by the single prover in Figure 11. Completeness and special soundness therefore follow directly from the proof in [16].

For special honest-verifier-combiner zero-knowledge with threshold \(t = \alpha - 1\) [31], our proof proceeds with a special case of the definition given in [31] where the role of the combiner is subsumed by the verifier.

We create a simulator \(S\) that works exactly like \(A^{(a^*)}\) except for the following changes:

1. C1: In stage 1.2, \(S\) uses \(\pi(a^*) \leftarrow \text{Perm}(n)\) instead of \(A^{(a^*)}\)’s given \(\pi(a^*)\).

2. C2: In stage 1.4, \(S\) sends \(\tilde{\sigma}_i(a^*) \leftarrow G_1\).
(3) C3: In stage 2, \(S\) simulates DPK\(\tilde{0}\) by running the simulator \(S_0\) of DPK\(\tilde{0}\).

We argue that the view of \((A^{(a)})_{a \in \{0,1\}}\) and \(V\) when interacting with \(S\) is indistinguishable from their view when interacting with \((A^{(a)})\). We sketch this in the following sequence of games:

- **G0**: Real protocol between \((A^{(a)})_{a \in \{0,1\}}\) and \(V\) and \((A^{(a)})\).
- **G1**: Apply change C3. This is indistinguishable from \(G0\) by the zero-knowledge property of DPK\(\tilde{0}\).
- **G2**: Apply change C1. This is indistinguishable from \(G1\) because of semantic security of \(E_0\).
- **G3**: Apply change C2. This is indistinguishable from \(G2\) because of the randomly chosen binding factor \(b_i\), which makes \(\sigma_i^{(a)}\) indistinguishable from random elements drawn from \(G_1\).

\(\square\)

### D PROOF FOR THEOREM 2

**Proof. Completeness:** If all provers \((A^{(a)})_{a \in \{0,1\}}\) are honest, then they can successfully create an encryption of the full BBS+ signature \(\sigma_i := (\langle d_1 \rangle \tilde{\sigma}_i, c_i, r_i) = ((f g_1^{d_1^{(a)}} r_i^{\tilde{\sigma}_i}) \tilde{\sigma}_i, c_i, r_i)\) as \(e_{\sigma_i} := \langle \text{Enc}_{\tilde{\sigma}_i}(f g_1^{d_1^{(a)}} r_i^{\tilde{\sigma}_i}), \text{Enc}_{\tilde{\sigma}_i}(c_i), \text{Enc}_{\tilde{\sigma}_i}(r_i) \rangle\) (stages 2.1 and 2.2). These encryptions are re-encrypted and permuted by \(\pi\) in stage 2.3. After stages 2.4 and 2.5, for each \(j \in \{0, \ldots, n-1\}\), blinded signatures \((\tilde{\sigma}_i, r_{\pi^{-1}(j)}) := (\langle d_1 \rangle g_1^{d_1^{(a)}}, c_x^{\pi^{-1}(j)} + b_{c_j}, r_x^{\pi^{-1}(j)} + b_{r_j})\) are obtained. It can be inspected that the blinded signatures satisfy the conditions of DPK\(2\).

**Special soundness:** We show that if the verifier \(V\) accepts then an extractor \(E\) can extract an \(\tilde{r}\) such that \(g_1^{\tilde{r}_1} \in \Phi\). \(E\) runs the extractor \(E_1\) for DPK\(1\) to obtain \(\langle \tilde{c}_i, \tilde{t}_i \rangle\) for each \(i \in \{0, \ldots, n-1\}\).

Case 1 (\(\forall i : \tilde{c}_i = \tilde{t}_i\)): In this case, \(E\) simply outputs \(\tilde{t}_i\) as the desired value (commitment \(\tilde{c}_i = g_1^{\tilde{t}_i} \in \Phi\) is the desired commitment).

Case 2 (\(\exists i : \tilde{c}_i \neq \tilde{t}_i\)): We show that this case is not possible because of the computational binding of Pedersen commitments and the unforgeability of BBS+ signatures under the \(\Phi\)-Strong Diffie Hellman assumption (see Section 4.3).

We show this by constructing a forger \(\mathcal{F}\) interacting with a BBS+ signing oracle \(\mathcal{O}_{\text{BBS+}}\). The forger interacts internally with provers \((A^{(a)})_{a \in \{0,1\}}\) as a verifier of the reverse set membership protocol. The BBS+ signature unforgeability game between \(\mathcal{F}\) and \(\mathcal{O}_{\text{BBS+}}\) goes as follows:

1. \(\mathcal{O}_{\text{BBS+}}\) generates random secret key \(x \in \mathbb{Z}_q\) and sends public key \(y = g_1^{x}\) to \(\mathcal{F}\).
2. \(\mathcal{F}\) runs extractor \(E_1\) for each \(i \in \{0, \ldots, n-1\}\) to extract \(\langle \tilde{c}_i, \tilde{t}_i \rangle\).
3. \(\mathcal{F}\) sends signature queries \(\tilde{c}_i \in \mathcal{O}_{\text{BBS+}}\) and obtains signatures \(\sigma_i := (\tilde{c}_i, c_i, r_i)\). It constructs \(\sigma_i' := (\tilde{c}_i, c_i, r_i - \tilde{t}_i)\) and sends \(y, \sigma_i'\) to \((A^{(a)})_{a \in \{0,1\}}\).
4. On obtaining \((\tilde{\sigma}_i^{\pi^{-1}(j)}, r_{\pi^{-1}(j)} - \tilde{r}_1)\) from \((A^{(a)})_{a \in \{0,1\}}\), \(\mathcal{F}\) runs the extractor \(E_2\) for DPK\(2\) and obtains \((b_{c_j}, b_{r_j}, r_{\pi^{-1}(j)}, \delta_{1}, \delta_{2})\).
5. \(\mathcal{F}\) computes \(m^* = b_{\pi^{-1}(j)}, \sigma^* = \langle d_1 \rangle g_1^{d_1^{(a)}}, c_{\pi^{-1}(j)} - b_{c_j}, r_{\pi^{-1}(j)} - b_{r_j}\) and outputs \((m^*, \sigma^*)\) as its forgery.

Note that \(\mathcal{F}\) produces a view for provers \((A^{(a)})_{a \in \{0,1\}}\) that is indistinguishable from the view produced by the real verifier in Figure 9.

In particular, \(\mathcal{F}\) sends quasi-signatures \(\sigma_i' = ((f g_1^{d_1^{(a)}} r_i^{\tilde{\sigma}_i}), c_i, r_i - \tilde{t}_i)\) for randomly drawn \(c_i\) and \(r_i\), which are identically distributed to quasi-signatures \(\sigma_i' = ((f g_1^{d_1^{(a)}} r_i^{\tilde{\sigma}_i}), c_i, r_i)\) for randomly drawn \(c_i\) and \(r_i\), sent as by the real verifier in stage 2.1.

Thus, \(\mathcal{F}\) obtains \((\tilde{\sigma}_i^{\pi^{-1}(j)}, c_{\pi^{-1}(j)} - b_{c_j}, r_{\pi^{-1}(j)} - b_{r_j})\) identically distributed to what the real verifier obtains in stage 3. Since the verifier accepts, DPK\(2\) must have succeeded. Thus, by Lemma 4, \(\sigma^*\) is a valid BBS+ signature on message \(m^*\) under public key \(y\).

Since \(\forall i, \tilde{c}_i \neq \tilde{t}_i\), no signature on message \(\tilde{c}_i\) was queried from \(O_{\text{BBS+}}\) in step 3 and this is a forgery. Since the forger queries \(\Phi\) signatures in total, this is not possible under the \(\Phi\)-Strong Diffie Hellman assumption [5].

**Lemma 4.** Let \(\delta_1, \delta_2, d_1, d_2, \gamma, b_1, b_2, \delta_1, \delta_2\) be defined as in Figure 9 and \(b_1, b_2, \delta_1, \delta_2\) be as output by extractor \(E_2\). If DPK\(2\) succeeds, then \((\tilde{\sigma}_i^{\pi^{-1}(j)}, c_{\pi^{-1}(j)} - b_{c_j}, r_{\pi^{-1}(j)} - b_{r_j})\) is a valid BBS+ signature on message \(\tilde{c}_i^{\pi^{-1}(j)}\) under the verifier’s public key \(y\).

**Proof.** Since DPK\(2\) succeeds, by its soundness property, we must have:

\[
\begin{align*}
\mathcal{O} & = \mathcal{B}_1 \mathcal{B}_2 \mathcal{B}_3 \mathcal{B}_4 \mathcal{B}_5, \\
\delta_1 & = \mathcal{B}_1 \mathcal{B}_2 \mathcal{B}_3 \mathcal{B}_4, \\
\delta_2 & = \mathcal{B}_1 \mathcal{B}_2 \mathcal{B}_3, \\
\gamma & = \mathcal{B}_1 \mathcal{B}_2, \\
b_1 & = \mathcal{B}_1, \\
b_2 & = \mathcal{B}_2.
\end{align*}
\]

From Equations 3 and 4, we have:

\[
\begin{align*}
\mathcal{O} & = \mathcal{B}_1 \mathcal{B}_2 \mathcal{B}_3 \mathcal{B}_4 \mathcal{B}_5, \\
\delta_1 & = \mathcal{B}_1 \mathcal{B}_2 \mathcal{B}_3, \\
\delta_2 & = \mathcal{B}_1 \mathcal{B}_2, \\
b_1 & = \mathcal{B}_1, \\
b_2 & = \mathcal{B}_2.
\end{align*}
\]

It must be that \(\delta_1 = b_1, b_2,\) otherwise we would have broken the computational binding property of Pedersen commitments. Substituting this in Equation 2 and abbreviating \(\pi^{-1}(j)\) we get:

\[
\begin{align*}
\frac{e(\tilde{\sigma}_i, y_f^{\tilde{c}_i})}{e(f g_1^{d_1^{(a)}} h_1, f_2)} & = e(\tilde{\sigma}_i, f_2) e(g_1 y_f^{\tilde{c}_i}) e(h_1, f_2) e(f_1, f_2) e(b_{c_j}), \\
\frac{e(\tilde{\sigma}_i, y_f^{\tilde{c}_i})}{e(f_1 g_1^{d_1^{(a)}} h_1, f_2)} & = e(f_1 g_1^{d_1^{(a)}} h_1, f_2) e(b_{c_j}).
\end{align*}
\]

Special honest-verifier-combiner zero-knowledge with threshold \(t = a - 1\) [31]:
We show the construction for a simulator $S$ that does not obtain the input $A^{(a)}$ but simulates its view. $S$ simulates the view produced by $A^{(a)}$ by working exactly like $A^{(a)}$ except for the following changes:

- **C1:** $S$ simulates DPK$_1$ by simply running the simulator $S_1$ for DPK$_1$.

- **C2:** In contrast to $A^{(a)}$ sending $e_{r_i^{(a)}} := \text{Enc}_\theta(r_i^{(a)})$ in the beginning of stage 2.2, $S$ sends $e_{r_i^{(a)}} := \text{Enc}_\theta(0)$ instead.

- **C3:** $S$ randomly chooses a permutation $\pi^{(a)}$ $\xleftarrow{\$}$ Perm$(n)$ instead of the given permutation $\pi^{(a)}$ in stage 2.3.

- **C4:** $S$ simulates DPK$_2$ in stage 3.3 by running the simulator $S_2$ for DPK$_2$.

We argue that the views produced by $A^{(a)}$ and $S$ indistinguishable, via the following sequence of games:

- **G0:** Real protocol between $A^{(a)}$ and $(A^{(a)})_{x \xrightarrow{\$} \mathcal{V}}$.

- **G1:** Apply change C1. This is indistinguishable from G0 by the zero-knowledge property of DPK$_1$.

- **G2:** Apply change C4. This is indistinguishable from G1 by the zero-knowledge property of DPK$_2$.

- **G3:** Apply change C2. By the semantic security of E$_\theta$ and E$_\vartheta$, the encryptions produced by the simulator are indistinguishable with those produced by the real $A^{(a)}$ as long as the outputs post decryption are indistinguishable. Post decryption, for each $j \in \{1, \ldots, n\}$, the simulated world outputs
  
  $$(S_{\pi(j)}g)^{-\sum_{a \in [a]} b_{r_j}^{(a)}}(c_{\pi(j)} + \sum_{a \in [a]} b_{r_j}^{(a)}) \mod q.$$  

  whereas the real world outputs
  
  $$(S_{\pi(j)}g)^{-\sum_{a \in [a]} b_{r_j}^{(a)}}(c_{\pi(j)} + \sum_{a \in [a]} b_{r_j}^{(a)}) \mod q.$$  

  Note that the first component is blinded by a perfectly random element $g_1^{-b_{r_j}^{(a)}}$ in group G1. Thus, the second and third components become independent of each other and can be analysed in isolation. The second component is clearly identically distributed in both the worlds. The third component is identically distributed because $b_{r_j}^{(a)}$ is uniformly random in $\mathbb{Z}_q$.

**F PROOF FOR THEOREM 4**

Proof. We prove this by constructing a simulator $S$ that obtains the final tally but does not obtain input of the honest EA, $A^{(a)}$. It also does not obtain the voters’ vote. Yet, $S$ can simulate the view produced by the honest EA and the voters. $S$ proceeds as follows:

1. Given the final tally $(n_0, \ldots, n_{m-1})$, $n = \sum n_j$, where $n_j$ denotes the number of votes obtained by candidate $j$, $S$ constructs a fake vote list as $\bar{v}^S \leftarrow (0, \ldots, 0, m - 1, \ldots, m - 1)$, $n_j$ times.

2. When vote casting, $S$ sends $\bar{x}_i := \bar{v}^S_i$ to the BMD.

3. $S$ simulates the vote processing protocol by running its simulator $S_1$ on input $\bar{v}^S := \{\bar{x}_i + \bar{w}_j_i^S \mid i \in [n]\}$. It simulates the ZKPs of set membership and reverse set membership by running their simulators $S_2$ and $S_3$ on $B_1$ commitments $(g_{i,j}^{r_1}, g_{1}^{r_1 h_{i,j}^{r_1 - 1}}, \ldots, g_{i,j}^{r_{m-1} h_{i,j}^{r_{m-1} - 1}})$ and $B_{2}$ votes $\pi^{(a)}(v^S)$, where $\pi^{(a)}$ $\xleftarrow{\$}$ Perm$(n)$ and $r$ $\xleftarrow{\$}$ $\mathbb{Z}_q$ are chosen by $S$.

4. When VVPRs need to be produced, $S$ produces fake VVPRs computed using $\bar{w}_j^S$ and $\bar{w}_j^S$.

We show that the simulated world is indistinguishable from the real world through the following sequence of games:

1. **G0:** The real protocol executed by the honest EA, the voters and the adversary controlling other EAs, POs and the BMDs.

2. **G1:** Simulate the vote processing protocol by $S_1$. Indistinguishable from G0 by Lemma 1.

3. **G2:** Simulate ZKPs of set membership and reverse membership by $S_2$ and $S_3$. Since there can be no verification failures, this is indistinguishable from G1 by Theorems 1 and 2.

4. **G3:** Use ballot identifier $u_i^S$ for the $i$th voter, send input $\{u_i^S + \bar{w}_i \mid i \in [n]\}$ to $S_2$, where $\bar{w}_i \leftarrow (\bar{u}_i - u_i^S) \mod m$ and $\bar{u}_i$ is the voter’s real choice, and reveal VVPRs with $u_i^S$ and $\bar{w}_i$. Since $u_i$ and $u_i^S$ are both drawn uniformly from $\mathbb{Z}_q$, $G_3$ is indistinguishable from G2.
(5) **G4**: Send $\tilde{v}_i^S$ to the BMD instead of $\tilde{w}_i$, send input $\{u_i^S + \tilde{v}_i^S \mid i \in [n]\}$ to $S_2$ and reveal VVPRs with $u_i^S$ and $\tilde{w}_i$.

In **G3**, the BMD receives $\tilde{w}_i \leftarrow (\bar{v}_i - u_i^S) \mod m$ whereas in **G4** it receives $\tilde{w}_i \leftarrow (\bar{v}_i^S - u_i^S) \mod m$. By Lemma 2, the two $\tilde{w}_i$’s are indistinguishable. In **G3**, $S_2$ receives the set $\{u_i^S + (\bar{v}_i - u_i^S) \mod m \mid i \in [n]\}$ whereas in **G4** it receives $\{u_{\pi(i)}^S + (\bar{v}_i - u_{\pi(i)}^S) \mod m \mid i \in [n]\}$, where $\pi$ is some permutation, because the total tally is the same. Since $u_i^S, u_{\pi(i)}^S$ are randomly drawn from $\mathbb{Z}_q$ independently of the votes, both these sets are indistinguishable. Similarly, in **G3**, the set of revealed VVPRs can be characterised by $\{(u_i^S, (\bar{v}_i - u_i^S) \mod m) \mid i \in [n]\}$ whereas in **G4** it is characterised by $\{(u_{\pi(i)}^S, (\bar{v}_i - u_{\pi(i)}^S) \mod m) \mid i \in [n]\}$. Since $u_i^S, u_{\pi(i)}^S \leftarrow \mathbb{Z}_q$ both sets of VVPRs are indistinguishable. Thus, **G4** is indistinguishable from **G3**.

\[\square\]