



A Simple Protocol for Tracker Based Voting

Rosario Giustolisi¹(✉), Peter Y. A. Ryan², and Maryam Sheikhi Garjan³

¹ IT University of Copenhagen, Copenhagen, Denmark
rosg@itu.dk

² University of Luxembourg, Esch-sur-Alzette, Luxembourg
peter.ryan@uni.lu

³ University of Hamburg, Hamburg, Germany
maryam.sheikhi.garjan@uni-hamburg.de

Abstract. We present the simplest tracker based voting protocol to date that combines receipt-free and transparent verification, and immediate ballot-casting assurance. Existing schemes achieve these goals only partially or with additional assumptions: Selene and Hyperion rely on a private channel to deliver tracker information after tallying, while Surtr removes that assumption at the cost of more complex voter interaction. Our protocol avoids these trade-offs. Each voter encrypts the chosen candidate and all remaining candidates, each paired with a tracker. The system then immediately shuffles and decrypts these candidate-tracker pairs and publishes them in the voter's bulletin-board row. This allows the voter to verify in cleartext that the chosen vote is associated with the tracker, while a coercer observes only indistinguishable pairs. Tallying uses one re-encrypted real-vote ciphertext per voter, enabling direct tally extraction.

1 Introduction

A common way to let voters check that their ballot is counted is to give each voter a tracker that eventually appears on the bulletin board. State-of-the-art tracker-based systems [3, 11, 13, 15, 19] make this verification transparent: voters can identify their vote in cleartext. Some of these schemes [11, 13, 19] also provide receipt-free verification by allowing a coerced voter to equivocate about trackers.

However, the current designs involve important trade-offs. Selene [19] and Hyperion [11] require a private channel so that tracker information can be delivered privately (i.e. hidden from the coercer) to the voter after tallying. Surtr [13] removes this private-channel assumption, but requires more complex voter interaction to preserve coercion resistance, as voters are required to keep track of any coercer-chosen tracker until tallying.

In this paper, we present a simple tracker-based protocol that provides transparent and receipt-free verification without a private channel and without additional voter burden. Compared with prior systems, we avoid: (i) verified generation, encryption, and mixing of trackers as in Selene; (ii) dual keys and exponentiation mixnets as in Hyperion; and (iii) extra dual keys and assumptions

on coerced voters as in Surtr. In addition to privacy, verifiability, and receipt-free verification, our protocol also gives immediate ballot-casting assurance [1], whereas prior tracker-based systems focus on post-tally verification.

The core idea is simple: the voter encrypts the chosen candidate and associates it with an encrypted commitment to the tracker, and does the same for each of the other $N - 1$ candidates, each with a distinct tracker commitment. Importantly, the chosen candidate-tracker commitment encryption pair is positioned first in the ballot. The tally servers then immediately mix and decrypt all N candidate-tracker commitment pairs and publish the shuffled plaintext pairs in that voter’s bulletin-board row. This gives immediate voter confirmation: the voter can find the chosen candidate next to the corresponding tracker commitment by using the tracker associated with that commitment. At the same time, it still hides the real choice from a coercer, who sees only N candidate-tracker commitment pairs. After voting closes, the first encrypted pairs from all voters’ bulletin-board rows are included in tallying, enabling direct tally extraction.

Contributions. We present a simple tracker-based voting protocol that achieves transparent and receipt-free verification, and ballot-casting assurance. The protocol is simple both cryptographically and operationally: it relies only on standard mixnet constructions and can be instantiated with any threshold encryption scheme and any verifiable parallel shuffle. From the voter’s perspective, the vote-tracker commitment association is revealed immediately on the bulletin board, publicly and in the clear, so the voter does not have to remember any coercer-chosen tracker until after the election closes. We also give a formal security analysis proving that the protocol satisfies ballot privacy and verifiability under standard cryptographic assumptions.

2 Related Work

End-to-end verifiable voting systems have been a major focus of research over the past decade. Pioneering work on optical-scan voting includes Scantegrity [7], which demonstrated end-to-end voter-verifiable elections in a real-world setting. Remotegrity [22] extended this approach to remote voting, providing end-to-end verifiable Internet voting. Both systems rely on voter-verifiable paper records and cryptographic proofs, setting the foundation for transparent verification.

Selections [8] tackled the challenge of combining Internet voting with coercion-resistance through over-the-shoulder coercion-resistance mechanisms. SureVote [6] introduced a code-voting method allowing voters to use pre-distributed codes for response verification without cryptographic operations. Pretty Good Democracy [20] improved this by distributing acknowledgment code management among a group of trustees, decreasing the reliance on a single trusted server. More recently, post-quantum secure end-to-end verifiable schemes like Epoque [4] have extended these ideas to resist quantum adversaries. These works collectively demonstrate that achieving transparent, verifiable voting with user-friendly mechanisms remains an active research challenge. Our protocol contributes to this line of work by combining immediate ballot-casting assurance

with transparent and receipt-free verification, while maintaining the simplicity and directness that voters expect.

Many voting protocols offer strong coercion resistance [2, 9, 10, 12, 14, 16, 21], but without transparent verification. Only a small set of systems combine receipt-free verification with plaintext-style vote tracking. Early work includes the boardroom scheme of [3], where trackers are generated by voters, and sElect [15], which uses partially voter-generated trackers for accountability. However, these designs do not provide receipt-free verification, because a voter can reveal the tracker to a coercer before tallying.

Selene [19] provides receipt-free verification by disclosing tracker information only after tally publication, enabling a coerced voter to equivocate. Its notification mechanism, however, relies on a private channel to preserve that equivocation in front of a coercer. Hyperion [11] replaces trackers with Diffie–Hellman terms that let voters recover their vote in plaintext, but it still requires a private notification channel. Surtr [13] achieves receipt-free verification without the need for a private notification channel. However, as in the previous schemes, verification is post-tally, and a coerced voter needs to write down a coercer-chosen tracker until the tally is published in order to equivocate.

Our scheme achieves transparent voter verification immediately for *recorded-as-intended*, since the vote-tracker association is revealed on the bulletin board after ballot casting, allowing the voter to verify in clear that their vote is correctly associated with the tracker. In Selene and Hyperion, transparent voter verification also covers *tallied-as-recorded*, but no transparent verification is possible before tallying and without a private notification channel.

3 A Simple Protocol for Tracker Based Voting

We first give an informal overview of the protocol. Its full specification is described in Sect. 3.3.

3.1 Overview

Ballot Casting. Let $\mathbb{V} = \{v_1, \dots, v_N\}$ be the candidate set and $\mathbb{I} = \{i_1, \dots, i_M\}$ the electoral roll. The voter i generates N random trackers $x_0, x_1, \dots, x_{N-1}$ and computes the corresponding tracker commitments $g^{x_0}, g^{x_1}, \dots, g^{x_{N-1}}$. The voter chooses their candidate $v_o \in \mathbb{V}$ which is associated with the tracker x_0 . The voter's ballot β_i then contains N ciphertext pairs, one per candidate, each encrypting a candidate alongside an encryption of the tracker commitment, with the real vote pair $(\llbracket v_o \rrbracket, \llbracket g^{x_0} \rrbracket)$ placed first in the ballot. The ballot is posted on the voter's row $\mathcal{BB}_i$ of the bulletin board $\mathcal{BB}$.

Ballot Recording. Right after β_i is posted on the bulletin board, the tally servers retain the first ciphertext $\llbracket v_o \rrbracket$ as the ciphertext that will later be tallied. Then, the servers run a verifiable pairwise re-encryption mix over all N candidate-tracker commitment ciphertext pairs from β_i . This shuffles the pairs into a

random order, publishing a proof of correct re-encryption. Then, they apply a decryption mix to reveal the N plaintext pairs, publishing in the voter's row $\mathcal{BB}_i$ the list $(v_{k_1}, g^{x_{k_1}}), \dots, (v_{k_N}, g^{x_{k_N}})$ in shuffled order, together with a proof of correct decryption.

Voter Verification. The voter can now check their vote on the bulletin board immediately after casting, without waiting for tallying. Knowing v_o and x_o , the voter device computes g^{x_o} and scans the published plaintext pairs to find the entry (v_o, g^{x_o}) . Finding it confirms that the correct candidate is associated with the correct tracker: the vote was cast as intended and is recorded on the board. Because decrypted candidate-tracker pairs appear after ballot casting, this constitutes immediate ballot-casting assurance.

Receipt-Freeness. All N candidates appear in the decrypted list, each associated with a distinct commitment, so the coercer cannot determine which pair corresponds to the real vote by inspection. If the voter is coerced into revealing their vote, the voter can point to any entry (v_k, g^{x_k}) in the published list and produce the corresponding tracker x_k as a convincing demonstration that v_k was their real choice. Since the commitment g^{x_k} is on the board and x_k opens it, a coercer has no way to refute this equivocation. Thus, a coerced voter can always claim to have voted for the coercer-preferred candidate regardless of their true choice. Note that all trackers, tracker commitments, and their associations with candidates can be shared with the coercer right after ballot casting, as they do not reveal the voter's choice.

Our protocol provides receipt-freeness for the verification mechanism, but not full receipt-freeness for ballot casting. In particular, we assume that the coercer neither provides the voter with a precomputed ballot nor learns the randomness used to generate the ballot. Otherwise, the coerced voter may be unable to equivocate, and the coercer could verify the cast vote on the bulletin board. One possible mitigation is to require the server to challenge the voter to re-randomize the ballot before final casting. A potential direction is to use interactive proofs and malleable signatures, as in BeleniosRF [5]. However, the malleable signature construction used in BeleniosRF cannot be directly applied to our protocol, because it does not support conjunctions of zero-knowledge proofs of knowledge, which our construction relies on. Achieving full receipt-freeness would significantly increase protocol and voter-interaction complexity.

Tallying. After voting closes, the tally servers collect the first vote ciphertexts $\{\llbracket v_{oi} \rrbracket\}_{i=1}^M$ from the voter rows. These M ciphertexts are processed by a further re-encryption and decryption mix, yielding the multiset of plaintext votes $\{v_{oi}\}_{i=1}^M$ in shuffled order. The election result can be computed and verified universally from this list of plaintext votes. No additional private channels, dual keys, or post-tallying delivery steps are required.

3.2 Cryptographic Primitives

Our construction uses standard building blocks. A public-key encryption scheme ($\text{EKeyGen}, \text{Enc}, \text{Dec}$) is used for candidate and tracker ciphertexts. A digital signature scheme ($\text{SKeyGen}, \text{Sign}, \text{SVerify}$) authenticates the ballots submitted by voters. Trackers are generated by the randomized algorithm $\mathbf{x} \xleftarrow{\$} \text{TKeyGen}(pp)$, where $\mathbf{x} = (x_0, \dots, x_{N-1})$. We assume a distributed (t, n) -threshold decryption setting with no trusted authority: any set of at least t tallying parties can decrypt, and the resulting decryption is publicly verifiable.

Privacy-preserving shuffle of ciphertexts is performed using verifiable re-encryption and decryption mixnets. Integrity of the mixnet operations is ensured by the corresponding proofs of correct re-encryption and decryption.

Finally, verifiability is guaranteed through non-interactive zero-knowledge proofs of knowledge (NIZKPoK). For each relation R used by the protocol we write $(\text{Proof}_R, \text{Verify}_R)$ for the prover and verifier algorithms. These relations cover the statements needed later in the protocol description, such as well-formed encryptions, correct re-encryption/shuffling, and correct threshold decryption.

3.3 Formal Description

The algorithms that define the protocol are as follows. For each voter identity $id \in \mathbb{I}$, let $\mathcal{BB}_{id}$ denote the row of voter id in the bulletin board $\mathcal{BB}$.

- $\text{EASetup}(1^\lambda, \mathbb{I}, \mathbb{V}) \rightarrow (pk_{EA}, sk_{EA})$: on input the security parameter 1^λ , the electoral roll $\mathbb{I}$, and the candidate list $\mathbb{V}$, generates public parameters pp and computes $(pk_{EA}, sk_{EA}) \xleftarrow{\$} \text{EKeyGen}(pp)$.
- $\text{Setup}(id, pk_{EA}) \rightarrow (id, (\mathbf{x}, \mathbf{g}^\mathbf{x}, sk), (\mathbf{ctr}, \pi_{\mathbf{ctr}}, pk))$: on input of the voter identity id and pk_{EA} , which contains pp , computes $(sk, pk) \xleftarrow{\$} \text{SKeyGen}(pp)$, where pk contains a proof of knowledge π_{sk} , and generates random trackers $\mathbf{x} = (x_0, x_1, \dots, x_{N-1})$ and tracker commitments $\mathbf{g}^\mathbf{x} = (g^{x_0}, g^{x_1}, \dots, g^{x_{N-1}})$ by running $(\mathbf{x}, \mathbf{g}^\mathbf{x}) \xleftarrow{\$} \text{TKeyGen}(pp)$. Then, it computes $ctr_i \xleftarrow{\$} \text{Enc}(pk_{EA}, g^{x_i}; r_{x_i})$ for $i \in [0, N-1]$, and sets $\mathbf{ctr} = (ctr_0, ctr_1, \dots, ctr_{N-1})$. It runs $\pi_{\mathbf{ctr}} \xleftarrow{\$} \text{Proof}_{ctr}(\chi, \omega)$, where $\chi = (\mathbf{ctr}, pk_{EA})$, $\omega = (\mathbf{x}, \mathbf{r}_\mathbf{x})$ s.t. $(\chi, \omega) \in R^{\mathbf{ctr}}$ iff

$$ctr_0 = \text{Enc}(pk_{EA}, g^{x_0}; r_{x_0}) \wedge \dots \wedge ctr_{N-1} = \text{Enc}(pk_{EA}, g^{x_{N-1}}; r_{x_{N-1}}).$$

Note that $\pi_{\mathbf{ctr}}$ retains the order of the tracker commitments and it is bound to the voter's identity id .

- $\text{Vote}(id, sk, pk_{EA}, (v_0, \mathbf{ctr}, \pi_{\mathbf{ctr}})) \rightarrow \beta$: on input voter identity $id \in \mathbb{I}$, voter signing key sk , tallying servers' public key pk_{EA} , vote option $v_0 \in \mathbb{V}$, encryptions $\mathbf{ctr}$, proof $\pi_{\mathbf{ctr}}$, and implicit input $(\mathbb{V}, \mathbb{I})$ where $N = |\mathbb{V}|$,
 1. Compute $ct_0 \xleftarrow{\$} \text{Enc}(pk_{EA}, v_0; r_{v_0})$
 2. Set $\mathbb{W} = \mathbb{V} \setminus \{v_0\}$

3. For $i \in [1, N - 1]$, do:

$$\begin{aligned} v_i &\xleftarrow{\$} \mathbb{W} \\ ct_i &\xleftarrow{\$} \text{Enc}(pk_{EA}, v_i; r_{v_i}) \\ \mathbb{W} &= \mathbb{W} \setminus \{v_i\} \end{aligned}$$

4. Set $\mathbf{ct} = (ct_0, ct_1, \dots, ct_{N-1})$, $\mathbf{v} = (v_0, v_1, \dots, v_{N-1})$, and randomness $\mathbf{r_v} = (\mathbf{r_{v_0}}, \mathbf{r_{v_1}}, \dots, \mathbf{r_{v_{N-1}}})$.

5. Run $\pi_v \xleftarrow{\$} \text{Proof}_{\text{vote}}(\chi, \omega)$, where $\chi = (\mathbf{ct}, pk_{EA}, \mathbb{V})$, $\omega = (\mathbf{v}, \mathbf{r_v})$ s.t. $(\chi, \omega) \in R^{\text{vote}}$ iff

$$\begin{aligned} ct_0 &= \text{Enc}(pk_{EA}, v_0; r_{v_0}) \wedge \dots \wedge ct_{N-1} = \text{Enc}(pk_{EA}, v_{N-1}; r_{v_{N-1}}) \wedge \\ v_0 &\in \mathbb{V} \wedge \dots \wedge v_{N-1} \in \mathbb{V} \wedge \mathbf{v} \equiv \mathbb{V} \end{aligned}$$

6. Set $\pi = (\pi_v, \pi_{\mathbf{ctr}})$ and compute $\sigma \xleftarrow{\$} \text{Sign}(sk, (id, \mathbf{ct}, \mathbf{ctr}, \pi))$, and return $\beta = (id, pk, \mathbf{ct}, \mathbf{ctr}, \pi, \sigma)$.

- $\text{Validate}(\mathcal{BB}, \beta) \rightarrow \top/\perp$: on input ballot $\beta = (id, pk, \mathbf{ct}, \mathbf{ctr}, \pi, \sigma)$ and implicit input $(pk_{EA}, \mathbb{I}, \mathbb{V})$ checks that (i) $id \in \mathbb{I}$, (ii) no terms in β already appear in $\mathcal{BB}$, (iii) $\top \leftarrow \text{SVerify}(pk, (id, \mathbf{ct}, \mathbf{ctr}, \pi), \sigma)$, and (iv) $\top \leftarrow \text{Verify}(\chi, \pi)$, where $\chi = (pk_{EA}, \mathbf{ct}, \mathbf{ctr}, \mathbb{V})$ and $\pi = (\pi_v, \pi_{\mathbf{ctr}})$. If any of the checks fail, it returns $\perp$; otherwise it returns $\top$.
- $\text{Append}(\mathcal{BB}, \beta) \rightarrow \mathcal{BB}$: on input a ballot $\beta = (id, pk, \mathbf{ct}, \mathbf{ctr}, \pi, \sigma)$, checks that $\text{Validate}(\mathcal{BB}, \beta) = \top$. If so, it appends β to $\mathcal{BB}_{id}$.
- $\text{ImmediateMix}(\mathcal{BB}, sk_{EA}, id) \rightarrow \mathcal{BB}$: on input $\mathcal{BB}$, sk_{EA} , and id , obtains the ballot $\beta = (id, pk, \mathbf{ct}, \mathbf{ctr}, \pi, \sigma)$ from $\mathcal{BB}_{id}$ and then does the following:
 1. On input $(ct_0, ctr_0), (ct_1, ctr_1), \dots, (ct_{N-1}, ctr_{N-1})$, run a re-encryption mix to shuffle in parallel the vote ciphertexts and the tracker ciphertexts producing $\{(ct'_j, ctr'_j)\}_{j \in [N]}$ and proof $\pi_{\text{r-enc}}$.
 2. Run a decryption mix on $\{(ct'_j, ctr'_j)\}_{j \in [N]}$, which produces the shuffled $\{(v_j, g^{x_j})\}_{j \in [N]}$ and proof π_d .
 3. Append $\{(v_j, g^{x_j})\}_{j \in [N]}, \pi_{\text{r-enc}}, \pi_d$ to $\mathcal{BB}_{id}$.
- $\text{VerifyVote}(\mathcal{BB}, id, v, x_v) \rightarrow \perp/\top$: on input $\mathcal{BB}$, voter identity id , vote option v and tracker x_v , checks that (i) there is a ballot β in $\mathcal{BB}_{id}$ and that $\text{Validate}(\mathcal{BB}, \beta) = \top$; (ii) $\pi_{\text{r-enc}}$ and π_d in $\mathcal{BB}_{id}$ are valid; (iii) compute g^{x_v} to recover (v_i, g^{x_v}) from $\mathcal{BB}_{id}$ and check that $v_i = v$. If any of the checks fail, it returns $\perp$; otherwise it returns $\top$.
- $\text{Tally}(\mathcal{BB}, sk_{EA}) \rightarrow (R, \Pi)$: on input $\mathcal{BB}$, let $\{ct_{0i}\}_{i=1}^M$ be the first vote ciphertexts extracted from all valid non-empty voter rows of $\{\mathcal{BB}_{id}\}_{id \in \mathbb{I}}$:
 1. Run a re-encryption mix on $\{ct_{0i}\}_{i=1}^M$, producing shuffled $\{ct'_{0i}\}_{i=1}^M$ and proof $\pi_{\text{r-enc-tally}}$.
 2. Run a decryption mix on $\{ct'_{0i}\}_{i=1}^M$, producing shuffled $\{v_i\}_{i=1}^M$ and proof $\pi_{\text{d-tally}}$.
 3. Set $\Pi = (\pi_{\text{r-enc-tally}}, \pi_{\text{d-tally}})$ and compute the result $R = (R_v)_{v \in \mathbb{V}}$ by counting occurrences of each candidate in $\{v_i\}_{i=1}^M$.
- $\text{VerifyTally}(\mathcal{BB}, (R, \Pi)) \rightarrow \perp/\top$: on input $\mathcal{BB}$, the tally result and corresponding proofs (R, Π) , verifies all proofs on $\mathcal{BB}$ and the correctness of (R, Π) . If any of the checks fail, it returns $\perp$; otherwise it returns $\top$.

The protocol is organised into three phases.

Setup Phase: The tallying servers run $\text{EASetup}(1^\lambda, \mathbb{I}, \mathbb{V})$ to generate their key pair (pk_{EA}, sk_{EA}) . Each voter then runs $\text{Setup}(id, pk_{EA})$ to register and obtain a signing key pair (sk, pk) , together with trackers $\mathbf{x}$, the tracker commitments $\mathbf{g}^{\mathbf{x}}$, the tracker commitment encryptions $\mathbf{ctr}$, and the corresponding proof $\pi_{\mathbf{ctr}}$. If two devices are used (one for tracker generation and one for ballot casting), the casting device generates the signing key pair, while the tracker device generates the tracker-related material. This split supports cast-as-intended verifiability and, consequently, end-to-end verifiability (see Sect. 3.4). The bulletin board $\mathcal{BB}$ is initialised with the voter's id , pk , and π_{sk} .

Voting Phase: The voter chooses a candidate $v_0 \in \mathbb{V}$ and creates a ballot β using $\text{Vote}(id, sk, pk_{EA}, (v_0, \mathbf{ctr}, \pi_{\mathbf{ctr}}))$. The ballot is submitted to $\mathcal{BB}$ via $\text{Append}(\mathcal{BB}, \beta)$. The tallying servers then execute $\text{ImmediateMix}(\mathcal{BB}, sk_{EA}, id)$, which mixes and decrypts the candidate-tracker commitment pairs and appends the resulting plaintext pairs to the voter's row $\mathcal{BB}_{id}$. The voter checks inclusion and consistency by running $\perp/\top \leftarrow \text{VerifyVote}(\mathcal{BB}, id, v, x_v)$.

Tallying Phase: After voting closes, the tallying servers run the tallying algorithm $(R, \Pi) \leftarrow \text{Tally}(\mathcal{BB}, sk_{EA})$ to compute the final tally R . Any observer can verify its correctness by running $\perp/\top \leftarrow \text{VerifyTally}(\mathcal{BB}, (R, \Pi))$.

3.4 End-to-End Verifiability

We discuss informally here how our protocol achieves end-to-end verifiability. We provide a formal proof in Sect. 5.

End-to-end verifiability ensures that the system provides cast-as-intended, recorded-as-cast, and tallied-as-recorded assurances to voters and observers. In our protocol, the voter can directly verify cast-as-intended and recorded-as-cast assurances after ballot casting. Tallied-as-recorded assurance is guaranteed by the verifiable mixnet operations, which are publicly verifiable.

Cast-as-intended verifiability concerns auditing the ballot-creation process. This is typically achieved either by using *return codes* [6] or by giving the voter two devices, with at least one of them trusted. As in other tracker-based voting schemes, our protocol can achieve cast-as-intended verifiability with two devices. Specifically, if the voter has two separate devices (e.g., one for tracker generation and one for ballot casting), **Setup** can be split as follows: the tracker device runs **TKeyGen** to generate trackers and tracker commitments, and the casting device runs **SKeyGen** to generate the signing key pair. The first generated tracker, x_0 (the one associated with the voter's choice), is shown to the voter, and the encrypted commitments $\mathbf{ctr}$ with the corresponding proof $\pi_{\mathbf{ctr}}$ are transferred to the casting device (e.g., via QR code) before the ballot is cast. The voter inputs their vote choice to the casting device which generates a list of the encryptions of the vote options with the voter's choice first in the list. The proof $\pi_{\mathbf{ctr}}$ on encrypted commitments ensures that anyone can detect whether a malicious

casting device has altered their order. Therefore, the voter does not need to verify that the first ballot term is the encryption of the chosen candidate. As we show later formally, it is sufficient for cast-as-intended that either the tracker device or the casting device is trusted.

Recorded-as-cast verifiability is achieved by the immediate mixnet operations that reveal the vote-tracker commitment associations on the bulletin board right after ballot casting. The mixnets provide publicly verifiable proofs of their operations. However, the voter can directly check that their vote is correctly recorded by verifying that tracker x_0 corresponds to the tracker commitment g^{x_0} associated with the chosen candidate. This also provides immediate assurance to voters, as they do not need to wait until tallying to verify that their vote is recorded as intended.

Tallied-as-recorded verifiability is guaranteed by the verifiable mixnet operations that are publicly verifiable. The tallying mixnet processes the first vote ciphertexts, and the corresponding proofs ensure that the votes are correctly included in the tally and that the decryption is correct. Therefore, any observer can verify that the final tally corresponds to the recorded votes on the bulletin board.

3.5 Performance

The protocol relies on standard cryptographic primitives that are already present in state-of-the-art voting systems. In fact, it uses fewer components than prior tracker-based systems, as it does not require exponentiation mixnets or dual keys. The most expensive operations are the mixnet operations, which are performed once per voter for the immediate mix and once for the final tally. This has two advantages over prior tracker-based systems: (i) immediate mixing enables faster verification by voters, who do not need to scan the entire bulletin board to find their vote-tracker association; and (ii) the tallying phase depends only on the number of voters, not also on the number of candidates as in Surtr, because only the first ciphertexts of the ballots are included in the tallying mixnet.

4 Ballot Privacy

In this section, we prove that our protocol defined by $\mathcal{VS} = (\text{EASetup}, \text{Setup}, \text{Vote}, \text{Validate}, \text{Append}, \text{ImmediateMix}, \text{VerifyVote}, \text{Tally}, \text{VerifyTally})$ satisfies ballot privacy using the ballot privacy game Ballot-Priv^b described in Fig. 1. We adapt the game-based security definition of ballot privacy from [11]. In Hyperion, there is a post-tally verification that relies on the secret value sent to the voter through a private channel. However, since our scheme is not based on post-tally verification and does not require a secure channel, we modify the oracle VerifyVote in the definition by eliminating the GetSecret function. Furthermore, we modify the oracle Board by adding the function ProcessBallot , which appends the output of the decryption mixnet for each voter id to the corresponding row of the bulletin board. A voter ballot together with the output of the decryption

mixnet is referred to as **Processed**. In our scheme, the oracle **VerifyVote** calls the oracle **Board** to verify the voter ballot. For ballot privacy, we assume that the bulletin board and the tallying server are trusted, each voter is registered with a unique id , and that the voter devices are trusted, that is, these devices do not leak sensitive information, such as the secret signing key, trackers, or encryption randomness. In the **Ballot-Priv^b** game, the PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ interacts with the challenger, which executes the algorithms of the voting system through the various oracles. **SK** and **PK** are used to store the voter's secret key sk and public key pk . The left and right vote inputs to the oracle **LoR**, are stored in V_0 and V_1 , respectively, while **ST** stores the state st associated with the voters' ballots. The adversary $\mathcal{A}_1$ can register honest voters HV using the oracle **HSetup** or register dishonest voters DV with chosen public keys using the oracle **DSetup**. **ValidCred** outputs $\top$ if π_{sk} for voter id is valid, and $\perp$ otherwise. The adversary $\mathcal{A}_2$ has access to the following oracles:

- **LoR**(id, v_0, v_1), which allows the adversary to have an honest voter id cast one of two chosen votes (v_0 or v_1) based on a hidden bit b .
- **Tally**(), which initiates the tallying process but only if the result of votes cast in both scenarios (when $b = 0$ and $b = 1$) is identical.
- **VerifyVote**(id), which models a voter verifying their vote after casting.
- **Board**(), which allows the adversary $\mathcal{A}_2$ to obtain a published version of the Bulletin Board $\mathcal{BB}$.
- **Cast**(id, β), which enables $\mathcal{A}_2$ to directly submit a ballot β to $\mathcal{BB}$ for a voter id .
- **Verify**(), which allows $\mathcal{A}_2$ to check the validity of all ballots and the tally on public $\mathcal{BB}$.

The adversary $\mathcal{A}$ wins the game if it can determine the hidden value of b where b means that the challenger is processing v_b for an honest voter id . We define the advantage of the adversary $\mathcal{A}$ in the ballot privacy experiment as follows:

Theorem 1 (Ballot Privacy). *Assume that the encryption scheme with zero-knowledge proof of knowledge is IND-1-CCA secure, the signature scheme is EUF-CMA secure, the zero-knowledge proof systems are zero-knowledge, and the re-encryption mix is permutation hiding. Then the voting scheme $\mathcal{VS}$ achieves ballot privacy. In particular, for every PPT adversary $\mathcal{A}$,*

$$|\Pr[\text{Ballot-Priv}_{\mathcal{A}, \mathcal{VS}}^1(\lambda) = 1] - \Pr[\text{Ballot-Priv}_{\mathcal{A}, \mathcal{VS}}^0(\lambda) = 1]| \leq \text{Adv}_{\mathcal{B}}^{\text{IND-1-CCA}} + \epsilon,$$

where ϵ is a negligible function in security parameter λ and $\text{Adv}_{\mathcal{B}}^{\text{IND-1-CCA}}$ denotes the advantage of an adversary $\mathcal{B}$ against IND-1-CCA security.

Proof. **Game G_0 .** Let G_0 be the ballot privacy game $\text{Ballot-Priv}_{\mathcal{A}, \mathcal{VS}}^b$ in Fig. 1. We denote the adversary advantage in game G_0 by $\text{Adv}_{\mathcal{A}}^{G_0}$. In G_0 , upon a query **LoR**(id, v_0, v_1), the challenger samples two random trackers and computes their commitments (x_0, g^{x_0}) and (x_1, g^{x_1}) corresponding to the vote options v_0, v_1 , respectively. It then defines two possible ballots for the two inputs of the

Game $\text{Ballot} - \text{Priv}_{\mathcal{A}, \mathcal{VS}}^b$ $\text{SK}, \text{PK}, \text{ST}, \text{V}_0, \text{V}_1 \leftarrow [] \perp$ $\text{HV}, \text{DV} \leftarrow \emptyset$ $(\text{pk}_{EA}, \text{sk}_{EA}) \leftarrow \text{EASetup}()$ $\text{st}_A \leftarrow \mathcal{A}_1(\text{pk}_{EA}, \text{PK})$ $b' \leftarrow \mathcal{A}_2(\text{st}_A)$ return $(b' = b)$	Oracle $\text{HSetup}(id)$ $(sk, pk) \leftarrow \text{Setup}(id, \text{pk}_{EA})$ if $\text{ValidCred}(id, pk, \text{pk}_{EA}) = \perp$ return $\perp$ $\text{SK}[id] \leftarrow sk, \text{PK}[id] \leftarrow pk$ $\text{HV} \leftarrow \text{HV} \cup \{id\}$
Oracle $\text{LoR}(id, v_0, v_1)$ if $id \notin \text{HV}$ return $\perp$ $sk \leftarrow \text{SK}[id]$ $pk \leftarrow \text{PK}[id]$ $(\beta, \text{st}) \leftarrow \text{Vote}(id, sk, \text{pk}_{EA}, v_b)$ if $\text{Validate}(\mathcal{BB}, \beta) = \perp$ return $\perp$ $\text{V}_0[id] \leftarrow v_0, \text{V}_1[id] \leftarrow v_1,$ $\text{ST}[id] \leftarrow \text{st}, \mathcal{BB}[id] \leftarrow \beta$	Oracle $\text{DSetup}(id, pk)$ if $id \in \text{HV}$ or $\text{ValidCred}(id, pk, \text{pk}_{EA}) = \perp$ return $\perp$ $\text{PK}[id] \leftarrow pk$ $\text{DV} \leftarrow \text{DV} \cup \{id\}$
Oracle $\text{Board}()$ for $id \in \text{HV} \cup \text{DV}$ if $\mathcal{BB}[id] \neq \emptyset$ and $\text{Processed}[id] = 0$ $\mathcal{BB}[id] \leftarrow \text{ProcessBallot}(\mathcal{BB}[id], \text{sk}_{EA})$ $\text{Processed}[id] = 1$ return $\mathcal{BB}$	Oracle $\text{Cast}(id, \beta)$ if $id \notin \text{HV} \cup \text{DV}$ or $\text{Validate}(\mathcal{BB}, \beta) = \perp$ return $\perp$ $\mathcal{BB}[id] \leftarrow \beta$
Oracle $\text{VerifyVote}(id)$ if $id \notin \text{HV} \cup \text{DV}$ return $\perp$ $\mathcal{BB} \leftarrow \text{Board}()$ $pk \leftarrow \text{PK}[id], sk \leftarrow \text{SK}[id], \text{st} \leftarrow \text{ST}[id]$ return $\text{VerifyVote}(\mathcal{BB}, (id, pk), \text{st})$	Oracle $\text{Tally}()$ if $\rho(\text{V}_0) = \rho(\text{V}_1)$ return $\text{Tally}(\mathcal{BB}, \text{sk}_{EA})$
	Oracle $\text{Verify}()$ return $\text{Verify}(\mathcal{BB}, (R, \Pi))$

Fig. 1. Ballot privacy game. The adversary $\mathcal{A}$ wins if $b' = b$.

oracle $\text{LoR}(id, v_0, v_1)$ as follows: $\beta_0 = (id, pk, (ct_0, ctr_0), (ct_1, ctr_1), \pi_0, \sigma_0)$ and $\beta_1 = (id, pk, (ct_1, ctr_1), (ct_0, ctr_0), \pi_1, \sigma_1)$. When the board is published, β_b is processed by the decryption mixnet. The mixnet samples a random permutation τ and publishes the shuffled plaintext pairs $(v_{\tau(b)}, g^{x_{\tau(b)}}), (v_{\tau(1-b)}, g^{x_{\tau(1-b)}})$.

Game G_1 . In game G_1 , all proofs (vote, tracker, re-encryption, decryption, and mix proofs) are simulated. By the zero-knowledge property, the adversary $\mathcal{A}$ cannot distinguish simulated proofs from real proofs. Therefore, we have $\text{Adv}_{\mathcal{A}}^{G_0} \leq \text{Adv}_{\mathcal{A}}^{G_1} + \epsilon_{zk}$, where ϵ_{zk} denotes the advantage of an adversary in a zero-knowledge proof system in distinguishing simulated proofs from real proofs.

Game G_2 . In G_2 , we modify the oracle $\text{Cast}(id, \beta)$ in game G_1 so that the adversary $\mathcal{A}$ can only cast a ballot β on behalf of a dishonest voter $id \in \text{DV}$. This is because we assume that the signature scheme is existentially unforgeable. Hence,

we have $\text{Adv}_{\mathcal{A}}^{G_1} \leq \text{Adv}_{\mathcal{A}}^{G_2} + \epsilon_{\text{EUF-CMA}}$, where $\epsilon_{\text{EUF-CMA}}$ denotes the advantage of an adversary against the EUF-CMA property of the signature scheme.

Game G_3 . In G_3 we modify the oracle `VerifyVote` in game G_2 so that it always returns $\top$. By the correctness of the scheme, honestly generated ballots are accepted by `VerifyVote`. Moreover, the adversary cannot modify or cast on behalf of honest voters due to the unforgeability of the signature scheme. Therefore, $\text{Adv}_{\mathcal{A}}^{G_2} = \text{Adv}_{\mathcal{A}}^{G_3}$.

Game G_4 . This game is identical to the previous game G_3 except that, in the mixnet, instead of computing re-encryptions of the input ciphertext pairs, the challenger computes fresh encryptions of the same plaintext pairs. That is, for each input pair encrypting (v_j, x_j) , the challenger samples fresh randomness r'_j, s'_j and outputs $(ct'_j, ctr'_j) = (\text{Enc}(pk_{EA}, v_j; r'_j), \text{Enc}(pk_{EA}, g^{x_j}; s'_j))$. For re-randomizable encryption, this distribution is identically distributed, since $\text{ReEnc}(\text{Enc}(m; r); r') = \text{Enc}(m; r + r')$, and $r + r'$ is uniformly distributed whenever r' is sampled uniformly at random. Hence we have: $\text{Adv}_{\mathcal{A}}^{G_4} = \text{Adv}_{\mathcal{A}}^{G_3}$.

Game G_5 . Game G_5 differs from G_4 in that in oracle `Board`, we replace the real re-encryption/decryption mixnet execution by an ideal execution that first samples a uniform random permutation $\tau \leftarrow S_2$, applies it to $((v_b, g^{x_b}), (v_{1-b}, g^{x_{1-b}}))$ corresponding to the output ciphertext pairs of the oracle `LoR`(id, v_0, v_1) namely $((ct_b, ctr_b), (ct_{1-b}, ctr_{1-b}))$ and outputs $((v_{\tau(b)}, g^{x_{\tau(b)}}), (v_{\tau(1-b)}, g^{x_{\tau(1-b)}}))$. By the permutation-hiding property of the mixnet, the adversary $\mathcal{A}$ cannot distinguish the real mixnet execution from this ideal permuted execution except with negligible probability ϵ_{mix} . Hence, $\text{Adv}_{\mathcal{A}}^{G_5} \leq \text{Adv}_{\mathcal{A}}^{G_4} + \epsilon_{\text{mix}}$.

Game G_6 . In G_6 , analogously to game G_5 , we modify the oracle `Tally`(), where the ideal mixnet computes the result R based on the output oracles `LoR` and `Cast` provided that $\rho(\mathbf{V}_0) = \rho(\mathbf{V}_1)$. In this case the result R is the mix of $\rho(\mathbf{V}_b)$ and the decryption of the all valid ballots cast by the adversary $\mathcal{A}$. By the correctness of decryption and indistinguishability of the real and ideal executions of the mixnet, we have that $\text{Adv}_{\mathcal{A}}^{G_6} = \text{Adv}_{\mathcal{A}}^{G_5}$ (Fig. 2).

In our scheme, the vote encryption scheme with zero-knowledge proof of knowledge with label id for each voter satisfies IND-1-CCA security. Finally, we argue that the advantage of the adversary in game G_6 , namely $\text{Adv}_{\mathcal{A}}^{G_6}$, is equal to the advantage of the adversary $\mathcal{B}$ in the IND-1-CCA security game, namely $\text{Adv}_{\mathcal{A}}^{G_6} = \text{Adv}_{\mathcal{B}}^{\text{IND-1-CCA}}$. By combining the games, we obtain

$$\text{Adv}_{\mathcal{A}}^{\text{Ballot-Priv}^b} \leq \text{Adv}_{\mathcal{B}}^{\text{IND-1-CCA}} + \epsilon_{zk} + \epsilon_{\text{EUF-CMA}} + \epsilon_{\text{mix}},$$

where $\epsilon = \epsilon_{zk} + \epsilon_{\text{EUF-CMA}} + \epsilon_{\text{mix}}$ and $\text{Adv}_{\mathcal{A}}^{\text{Ballot-Priv}^b}$ is defined as

$$|\Pr[\text{Ballot-Priv}_{\mathcal{A}, \mathcal{V}_S}^1(\lambda) = 1] - \Pr[\text{Ballot-Priv}_{\mathcal{A}, \mathcal{V}_S}^0(\lambda) = 1]|.$$

<p>Game Verif($\mathcal{A}$)</p> <p>$SK_c, PK_c \leftarrow [\perp]$</p> <p>$SK_v, PK_v, ST, V \leftarrow [\perp]$</p> <p>$H_c, D_c, H_v, D_v, V_i \leftarrow \emptyset$</p> <p>$(pk_{EA}, sk_{EA}) \leftarrow \text{EASetup}()$</p> <p>$(\mathcal{BB}, st_A) \leftarrow \mathcal{A}_1(pk_{EA}, PK)$</p> <p>for $id \in V_{\text{Chkd}} \cap V_i$ do</p> <p> if $\text{VerifyVoted}(id, \mathcal{BB}) \neq \top$</p> <p> return $\perp$</p> <p> if $id \in H_v$ then</p> <p> $sk \leftarrow SK_v[id], v \leftarrow V[id]$</p> <p> if $\text{VerifyVote}(id, sk, v, \mathcal{BB}) \neq \top$</p> <p> return $\perp$</p> <p> $(r, \pi, st_A) \leftarrow \mathcal{A}_2(st_A, \mathcal{BB})$</p> <p> if $\text{Verify}(\mathcal{BB}, r, \pi) \neq \top$ or</p> <p> $r = \perp$ return $\perp$</p> <p> $M \leftarrow V_{\text{Chkd}} \cap (H_c \cup H_v)$</p> <p> $a \leftarrow V_{\text{Chkd}} \cap D_v$</p> <p> $b \leftarrow D_c \setminus (V_{\text{Chkd}} \cap H_v)$</p> <p> if $\exists n \in \{a, \dots, b\}$,</p> <p> $\exists v_1, \dots, v_n \in \mathcal{V}$,</p> <p> $\exists S \subseteq (V_i \setminus V_{\text{Chkd}}) \cap H_c$ s.t.</p> <p> $r = \rho([V[j]]_{j \in S}) \star \rho([v_j]_{j=1}^n) \star \rho([V[j]]_{j \in M})$</p> <p> return $\perp$</p>	<p>Oracle HCastSetup(id)</p> <p>for $id \notin D_c$</p> <p> $(sk, pk) \leftarrow \text{Setup}_c(id, pk_{EA})$</p> <p> if $\text{ValidCred}_c(id, pk, pk_{EA}) = \top$</p> <p> then $SK_c[id] \leftarrow sk; PK_c[id] \leftarrow pk,$</p> <p> $H_c \leftarrow H_c \cup \{id\}$</p> <p>Oracle DCastSetup(id, pk)</p> <p>for $id \notin H_c$</p> <p> if $\text{ValidCred}_c(id, pk, pk_{EA}) = \top$</p> <p> then $PK_c[id] \leftarrow pk,$</p> <p> $D_c \leftarrow D_c \cup \{id\}$</p> <p>Oracle HVerSetup(id)</p> <p>if $id \notin D_v$</p> <p> then return $\perp$</p> <p> $(sk, pk) \leftarrow \text{Setup}_v(id, pk_{EA})$</p> <p> if $\text{ValidCred}_v(id, pk, pk_{EA}) = \top$</p> <p> $SK_v[id] \leftarrow sk; PK_v[id] \leftarrow pk$</p> <p> $H_v \leftarrow H_v \cup \{id\}$</p> <p>Oracle DVerSetup(id, pk)</p> <p>if $id \in H_v$ return $\perp$</p> <p> if $\text{ValidCred}_v(id, pk, pk_{EA}) = \perp$</p> <p> return $\perp$</p> <p> $PK_v[id] \leftarrow pk, D_v \leftarrow D_v \cup \{id\}$</p> <p>Oracle Vote(id, v)</p> <p>if $id \notin H_c \cup D_c$, Return $\perp$</p> <p> $V_i \leftarrow V_i \cup \{id\}; V[id] \leftarrow v$</p> <p> if $id \in H_c$ then</p> <p> $sk \leftarrow SK_c[id]; pk \leftarrow PK_c[id]$</p> <p> $(\beta, st) \leftarrow \text{Vote}(pk_{EA}, id, sk, pk, v)$</p> <p> $ST[id] \leftarrow st, \text{Return } \beta$</p>
---	--

Fig. 2. Game-based definition of verifiability against a malicious voting board and a compromised device. The indices v and c denote verification and vote casting, respectively. The index i in V_i denotes voters with intended votes.

5 Verifiability

In this section, we prove that the proposed protocol satisfies verifiability against malicious tallying authorities and bulletin board¹, even in the presence of compromised voting or tracker generator devices. In particular, we consider the case

¹ If individual ballot verification is performed once and before tallying, then the bulletin board is assumed to be append-only. This is necessary since otherwise the bulletin board could remove the ballot of the voter immediately after the voter verifies their ballot. If verification is instead performed (also) after tallying, this assumption is not required.

where either the secret signing key or the secret tracker is leaked through a compromised device. We adopt the verifiability definition of Hyperion [11] to capture the cast-as-intended and recorded-as-cast guarantees of the voting scheme. We assume that each voter is uniquely registered with credentials (id, pk) and that the bulletin board provides a consistent public view during the entire election. Each voter uses two separate devices: a vote-casting device holding the signing key sk , and a tracker generator device holding the secret trackers $\mathbf{x}$. The definition models leakage from compromised voting devices by distinguishing between honest and corrupted casting and tracker generator devices. A corrupted casting device D_c leaks the voter's signing key sk , while a corrupted tracker generator device D_v leaks the secret trackers $\mathbf{x}$. We denote by V_{Chkd} the set of voters who successfully complete the verification procedure after casting their vote. This procedure consists of checking that a valid ballot associated with their identity id appears on the bulletin board $\mathcal{BB}$, and executing `VerifyVote`. We use `VerifyVoted` as shorthand for this combined verification procedure. Voters whose casting and tracker generator devices are both corrupted are considered fully corrupted voters. The algorithms `ValidCredc` and `ValidCredv` are used for the signing key and the verification key, respectively. We apply the same distinction to the `Setup` algorithm.

The definition captures the adversary's ability to output a valid bulletin board and tally result while achieving at least one of the following malicious goals:

- (i) changing the vote of a voter who possesses at least one honest device and successfully verifies their vote, namely voters in $V_{\text{Chkd}} \cap H_v$ or $V_{\text{Chkd}} \cap H_c$;
- (ii) for voters equipped with honest casting devices, forging, stuffing, or modifying ballots in any manner other than simply dropping the ballot, namely voters in $V_i \setminus (V_{\text{Chkd}} \cap H_c)$;
- (iii) for the remaining eligible voters, namely voters in $V_{\text{Chkd}} \cap D_v$ and $D_c \setminus (V_{\text{Chkd}} \cap H_v)$, cast more than one vote per voter.

Correctness. We first prove the correctness of the proposed scheme. Assume that all parties execute the protocol honestly and that the underlying cryptographic primitives satisfy their correctness properties, including the encryption scheme, signature scheme, mixnet, and zero-knowledge proof systems. Under these assumptions, for every honest voter the generated public key and the ballot generated by the voting algorithm are accepted by the bulletin board and verification procedures, the recorded ballot encodes the voter's intended vote, and the tallying algorithm correctly computes the election result from the set of recorded honest ballots. Hence, the published tally corresponds to the voters' intended votes.

Theorem 2 (Verifiability). *Assume the signature scheme is EUF-CMA secure, the NIZK proof systems satisfy simulation-sound extractability, the soundness of the mixnet, and hardness of the discrete logarithm problem. Then the voting scheme $\mathcal{VS}$ achieves verifiability.*

Proof. We first prove that for voters with honest vote-casting devices who have verified their ballot, $id \in (V_{\text{Chkd}} \cap H_c)$, the adversary cannot change the voter's ballot on $\mathcal{BB}$. We also show that for $id \in H_c$, the adversary can at most drop ballots, which proves that $\exists S \subseteq (V_i \setminus V_{\text{Chkd}}) \cap H_c$. Let $id \in H_c$, where the honest casting device generates a ballot containing a valid signature. Since the adversary $\mathcal{A}$ does not know the secret signing keys of honest voters, the EUF-CMA security of the signature scheme guarantees that $\mathcal{A}$ cannot forge a valid ballot, alter an honestly cast ballot, or cast ballots on behalf of any voter $id \in H_c$. Therefore, for all $id \in H_c$, the adversary can only drop the cast ballot. However, for all voters $id \in (V_{\text{Chkd}} \cap H_c)$, the ballots appear unchanged on $\mathcal{BB}$.

Now we consider all voters with $id \in (V_{\text{Chkd}} \cap H_v)$, namely voters with an honest tracker generator device who check their vote. Let $\{v_0, v_1\}$ be the candidate set and let β be a ballot of voter id with intended vote v_0 . Then we have $\beta = (id, pk, ((ct_0, ctr_0), (ct_1, ctr_1)), \pi, \sigma)$ as a ballot of voter id , where $(ct_i, ctr_i) = (\text{Enc}(pk_{EA}, v_i; r_{v_i}), \text{Enc}(pk_{EA}, g^{x_i}; r_{x_i}))$ for $i = 0, 1$. The adversary $\mathcal{A}$ (tally server) may have access to (g^{x_0}, g^{x_1}) by decrypting ctr_i and to the voter's secret signing key sk , assuming a dishonest vote-casting device. If the adversary $\mathcal{A}$ generates a modified ballot by altering the vote ciphertexts $\{ct_i\}$ or their ordering, this modification results in $\{(v_0, g^{x_1}), (v_1, g^{x_0})\}$, which does not pass verification. Assume that the adversary $\mathcal{A}$ modifies the ballot β to β' with intended vote v_1 such that voter verification $\text{VerifyVote}(\mathcal{BB}, (id, pk), \text{st}) = \top$, which means that the ballot β' on $\mathcal{BB}$ is a valid ballot for the voter with (id, pk) and the decryption mixnet returns a mix of $\{(v_0, g^{x_0}), (v_1, g^{x_1})\}$. Since the honest tracker generator device generates a proof π_{ctr} proving knowledge of witnesses (x_0, x_1, r_0, r_1) such that $ctr_i = \text{Enc}(pk_{EA}, g^{x_i}; r_i)$ for $i \in \{0, 1\}$, any adversary producing a fresh accepting proof for a modified ordered tuple must know valid witnesses corresponding to the new statement. Since the adversary $\mathcal{A}$ outputs a modified ballot $\beta' = (id, pk, ((ct'_1, ctr'_1), (ct'_0, ctr'_0)), \pi', \sigma')$ with valid proof $\pi' = (\pi'_v, \pi'_{\text{ctr}})$, where $(ctr'_1, ctr'_0, \pi'_{\text{ctr}}) \neq (ctr_0, ctr_1, \pi_{\text{ctr}})$. By the knowledge soundness of the proof system, there exists an extractor $\mathcal{E}$ that outputs witnesses (x'_1, x'_0, r'_0, r'_1) for the modified statement (ctr'_1, ctr'_0) , where $x'_1 = x_1$ and $x'_0 = x_0$. Therefore, the existence of $\mathcal{A}$ contradicts the discrete logarithm assumption underlying the binding and knowledge soundness properties of the commitments.

We now determine the bound on the number of malicious ballots that may be altered by the adversary. Consider voters with $id \in D_c$, namely voters with a dishonest vote-casting device. For voters satisfying $id \in D_c$ and $id \notin V_{\text{Chkd}}$, the adversary $\mathcal{A}$ may arbitrarily modify the vote or drop the ballot entirely, since no successful verification is performed. Therefore, the upper bound is $b = |D_c \setminus (V_{\text{Chkd}} \cap H_v)|$. Next, we consider voters such that $id \in D_c$ and $id \in V_{\text{Chkd}} \cap D_v$. For these voters, $\text{VerifyVoted}(id, \mathcal{BB}) = \top$, which guarantees that a valid ballot has been recorded on the bulletin board. However, since the tracker generator device is also dishonest, successful verification does not guarantee that the recorded vote corresponds to the voter's intended vote. Therefore, the minimum number

of ballots whose plaintext votes may be maliciously chosen by the adversary is lower-bounded by $a = |V_{\text{Chkd}} \cap D_v|$.

Let $M = V_{\text{Chkd}} \cap (H_v \cup H_c)$, $S \subseteq V_i \setminus (V_{\text{Chkd}} \cap H_c)$, and $n \in [a, b]$, which denotes the number of votes defined by $\mathcal{A}$. Therefore we have

$$R = \rho([V[j]]_{j \in S}) \star \rho([v_j]_{j=1}^n) \star \rho([V[j]]_{j \in \mathcal{M}}),$$

where $V[j]$ denotes the intended vote of voter j and $[v_j]_{j=1}^n$ are the votes chosen by the adversary $\mathcal{A}$.

6 Conclusion

We have presented a tracker-based voting protocol that simultaneously achieves receipt-free and transparent verification, and immediate ballot-casting assurance, while remaining simpler than prior related schemes. The key insight is for the tally servers to immediately mix and decrypt all candidate-tracker commitment pairs upon ballot receipt, then publish the result in the voter's bulletin-board row. This step provides clear vote-tracker confirmation right after casting, without requiring a private notification channel, post-tally interaction, or memorization of any coercer-chosen tracker until tallying.

From a cryptographic standpoint, the scheme relies only on standard building blocks: threshold public-key encryption, digital signatures, and verifiable re-encryption and decryption mixnets, all of which are already present in state-of-the-art tracker-based voting systems. In particular, it uses a strict subset of the primitives required by Selene, Hyperion, and Surtr: there are no exponentiation mixnets, no dual keys, and no verified tracker generation. We formally prove that the scheme satisfies ballot privacy under IND-1-CCA encryption, EUF-CMA signatures, and a permutation-hiding mixnet, and we give a game-based verifiability definition that covers malicious boards and device malware.

An interesting direction for future work is to investigate everlasting privacy variants of the scheme, following the approach of [17] for Hyperion. It would also be worth studying whether the two-device assumption for cast-as-intended can be relaxed, perhaps achieving universally cast-as-intended verifiability [18] and simplifying the voter experience even further.

References

1. Adida, B., Neff, C.A.: Ballot casting assurance. In: Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop. EVT 2006, USENIX Association (2006)
2. Araújo, R., Foulle, S., Traoré, J.: A practical and secure coercion-resistant scheme for remote elections. In: Dagstuhl Seminar Proceedings (2008)
3. Arnaud, M., Cortier, V., Wiedling, C.: Analysis of an electronic boardroom voting system. In: Heather, J., Schneider, S., Teague, V. (eds.) International Conference on E-Voting and Identity, Springer (2013). https://doi.org/10.1007/978-3-642-39185-9_7

4. Boyen, X., Haines, T., Müller, J.: Epoque: practical end-to-end verifiable post-quantum-secure e-voting. In: 2021 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 272–291. IEEE (2021)
5. Chaidos, P., Cortier, V., Fuchsbaauer, G., Galindo, D.: BELENIOSRF: a non-interactive receipt-free electronic voting scheme. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1614–1625 (2016)
6. Chaum, D.: Surevote. Int. Patent WO 1(55940), A1 (2001)
7. Chaum, D., et al.: Scantegrity: end-to-end voter-verifiable optical-scan voting. *IEEE Secur. Priv.* **6**(3), 40–46 (2008)
8. Clark, J., Hengartner, U.: Selections: internet voting with over-the-shoulder coercion-resistance. In: Danezis, G. (eds.) *International Conference on Financial Cryptography and Data Security*, pp. 47–61. Springer (2011). https://doi.org/10.1007/978-3-642-27576-0_4
9. Clark, J., Hengartner, U.: Selections: internet voting with over-the-shoulder coercion-resistance. In: Danezis, G. (ed.) *Financial Cryptography and Data Security*, pp. 47–61. Springer, Berlin Heidelberg, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27576-0_4
10. Cortier, V., Gaudry, P., Yang, Q.: Is the JCJ voting system really coercion-resistant? *Cryptology ePrint Archive* (2022)
11. Damodaran, A., Rastikian, S., Rønne, P.B., Ryan, P.Y.A.: Hyperion: transparent end-to-end verifiable voting with coercion mitigation. In: Nicomette, V., Benzekri, A., Boulahia-Cuppens, N., Vaidya, J. (eds.) *European Symposium on Research in Computer Security*, pp. 124–143. Springer (2025). https://doi.org/10.1007/978-3-032-07891-9_7
12. Giustolisi, R., Garjan, M.S., Schuermann, C.: Thwarting last-minute voter coercion. In: 2024 IEEE Symposium on Security and Privacy (SP) (2024)
13. Giustolisi, R., Garjan, M.S., Rønne, P.B.: Surtr: transparent verification with simple yet strong coercion mitigation. In: *E-VOTE-ID* (2025)
14. Juels, A., Catalano, D., Jakobsson, M.: Coercion-resistant electronic elections. In: Chaum, D., et al. (eds.) *Towards Trustworthy Elections*, pp. 37–63. Springer (2010). https://doi.org/10.1007/978-3-642-12980-3_2
15. Küsters, R., Müller, J., Scapin, E., Truderung, T.: sElect: a lightweight verifiable remote voting system. In: *CSF*, pp. 341–354. IEEE (2016)
16. Locher, P., Haenni, R., Koenig, R.E.: Coercion-resistant internet voting with everlasting privacy. In: Clark, J., Meiklejohn, S., Ryan, P.Y.A., Wallach, D., Brenner, M., Rohloff, K. (eds.) *FC 2016. LNCS*, vol. 9604, pp. 161–175. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53357-4_11
17. Mosaheb, R., Rønne, P.B., Ryan, P.Y., Sarfaraz, S.: Direct and transparent voter verification with everlasting receipt-freeness (2024)
18. Ryan, P.Y.A., Rønne, P.B., Arriaga, A., Pereira, O.: End-to-end verifiable elections with casting of publicly audited (2025). *cleartext ballots. E-VOTE-ID*
19. Ryan, P.Y.A., Rønne, P.B., Iovino, V.: Selene: voting with transparent verifiability and coercion-mitigation. In: Clark, J., Meiklejohn, S., Ryan, P., Wallach, D., Brenner, M., Rohloff, K. (eds.) *International Conference on Financial Cryptography and Data Security*, pp. 176–192. Springer (2016). https://doi.org/10.1007/978-3-662-53357-4_12
20. Ryan, P.Y.A., Teague, V.: Pretty good democracy. In: Christianson, B., Malcolm, J.A., Matyáš, V., Roe, M. (eds.) *International Workshop on Security Protocols*, pp. 111–122. Springer (2009). https://doi.org/10.1007/978-3-642-36213-2_15

21. Spycher, O., Koenig, R., Haenni, R., Schl pfer, M.: A new approach towards coercion-resistant remote e-voting in linear time. In: Danezis, G. (eds.) *Financial Cryptography and Data Security*, pp. 182–189. Springer (2012). https://doi.org/10.1007/978-3-642-27576-0_15
22. Zag rski, F., Carback, R., Chaum, D., Clark, J., Essex, A., Vora, P.L.: Remotegrity: design and use of an end-to-end verifiable remote voting system. In: Jacobson, M., Locasto, M., Mohassel, P., Safavi-Naini, R. (eds.) *International Conference on Applied Cryptography and Network Security*, pp. 441–457. Springer (2013). https://doi.org/10.1007/978-3-642-38980-1_28

Open Access This chapter is licensed under the terms of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License (<http://creativecommons.org/licenses/by-nc-nd/4.0/>), which permits any noncommercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if you modified the licensed material. You do not have permission under this license to share adapted material derived from this chapter or parts of it.

The images or other third party material in this chapter are included in the chapter’s Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter’s Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

