End-to-End Verifiable Internet Voting with Partially Private Bulletin Boards

Valeh Farzaliyev[0009−0007−9513−9812] and Jan Willemson[0000−0002−6290−2099]

Cybernetica, Narva mnt 20, Tartu 51009, Estonia
valeh.farzaliyev@cyber.ee, jan.willemson@cyber.ee

Abstract. In 2024, Harrison and Haines examined the applicability of STARKs in the context of homomorphically tallied elections. While their work ensures the Recorded-as-Cast and Tallied-as-Recorded properties of a voting system, it lacks Cast-as-Intended verification and does not provide a coercion mitigation mechanism. In this work, we address these challenges and propose an updated voting protocol that achieves all three verification properties, at the same time providing coercion resistance by allowing re-voting. Our approach leverages vector commitment schemes with update mechanisms. We implement our protocol and provide comparative benchmarks to the Harrison and Haines solution. Our approach significantly outperforms the latter, allowing processing a considerably larger number of votes within the same hardware limits.

Keywords: Electronic voting · zero-knowledge proofs · vector commitments

1 Introduction

Voting is the primary method used to delegate power in the democratic societies. The technical solutions to cast and count votes have gone through a long evolution, starting from raising one's hand to filling in paper ballots in the polling booths.

However, in the increasingly mobile world, gathering all the eligible voters in one place during a short period of time has become increasingly challenging. Thus, the need for reliable remote vote casting solutions has become more and more evident.

Practically speaking, there are two main alternatives for remote voting – casting the votes by paper mail, or over the Internet. Postal voting has the benefit of being familiar to the voters (the ballot can be exactly the same as for the in-person voting) and not requiring complex technical apparatus. However, in its classical form, postal voting is susceptible to coercion attacks, and it is hard to give integrity or secrecy guarantees for the postal channel. It has recently been even semi-formally shown by Vakarjuk *et al.* that postal voting has strictly weaker security guarantees than casting the votes over Internet [22].

Of course, Internet voting needs to satisfy all the requirements set for democratic elections, too, and meeting them is far from being trivial. In this paper,

we are going to concentrate on two main classes of such requirements: integrity of the elections, and voting freedom.

By election integrity we mean that the end result adequately represents the combination of individual preferences. In order to ensure this, all the main steps of the elections must be independently verifiable. In particular, the following verifiability properties are important to us.

- **Cast-as-Intended** verifiability ensures that every ballot corresponds to the voter preference.
- **Recorded-as-Cast** verifiability ensures that the way preferences are stored in a (digital or physical) ballot box corresponds to the way the ballots were submitted by the voters (i.e. no ballot has been altered in, deleted from or added to the ballot box illegitimately).
- **Tallied-as-Recorded** verifiability ensures that the contents of the ballot box is converted to the end result without alterations in the process.

The first two properties are sometimes commonly known as *individual verifiability*, and the latter one as *universal verifiability*. The three properties together are often called *end-to-end (E2E) verifiability*. However, we note that there is no common definition of E2E verifiability, and it may vary significantly between different sources; see [7] for an overview and comparison of different verifiability notions.

Voting freedom, on the other hand, means that the voters are able to cast their votes according to their true preferences, i.e. without coercion. There are several possible ways proposed to achieve coercion resistance. Typical requirements include *voter privacy* and *vote secrecy*, with the rationale that if the coercer can not learn the voter's preference, it will be hard for him to actually achieve the goal of coercion.

In case of remote voting, voter privacy may be hard to ensure. The main threat here is over-the-shoulder coercion, where another person (say, a family member) attempts to find out how the voter voted with the aim of influencing her choice and making sure she complied with the influence.

Lack of privacy can, to a significant extent, be compensated by the option of re-casting one's vote later, with the last vote getting tallied. In order for the anti-coercion measures to be effective, the voting scheme should be *receipt-free*, i.e. it should be impossible for the voter to get a strong proof of her tallied vote to show to the coercer.

We note that the requirements of E2E verifiability and receipt-freeness are tricky to achieve at the same time. From the verifiability point of view, we would like to give the voter some evidence that her vote was tallied as intended. At the same time, this evidence should not be strong enough to allow for proving the content of the vote to an external party.

Related works. There are numerous academic works trying to find a good trade-off between E2E verifiability and receipt-freeness. Cramer *et al.* [8] described the first efficient homomorphic tallying based multi-authority election

system with universal verifiability and vote secrecy by using non-interactive zero-knowledge proof of ballot correctness together with homomorphic secret sharing. Adding cast-as-intended verifiability makes the scheme coercible, since all the ballot encryptions are stored on a public bulletin board. To address this problem, the use of secret bulletin boards were introduced in [9] where a new cryptographic primitive, *commitment consistent encryption*, was introduced. The secret bulletin board stores only encrypted ballots, while the public bulletin board appends consistently extracted commitments for each ballot. The authors also proposed efficient constructions for both homomorphically tallied and mix-net based election systems. Universal verifiability is achieved through *Perfectly Private Audit Trail* (PPAT). Furthermore, the authors suggest using re-voting to enhance coercion resistance. Unfortunately, the protocol is not equipped with a cast-as-intended verification mechanism to be considered as end-to-end verifiable.

When the secret bulletin boards and re-voting are simultaneously used, E2E verifiability requires verification that only the last vote by the voter is included in the final tally. However, in practice it is usually guaranteed by trusting election authority. Recently, a new paradigm – deniable, yet verifiable vote updating mechanism – has emerged [12, 18, 20]. Particularly, DeVoS [20] is fully compatible with PPAT by introducing a new trusted party.

Our Contribution. In this work, we demonstrate an online voting protocol that satisfies E2E verifiability, while providing coercion-resistance by allowing verifiable vote updating. To do so, we follow the approach by Harrison and Haines presented in [13] and extend their protocol design. The main idea is to replace the trust in the components of the voting scheme with well-defined zero-knowledge proofs. In the original work, Harrison and Haines utilize *authenticated data structures* and proof of homomorphic tallying. The intent of their work is to show applicability of modern zero-knowledge proofs systems in the context of voting, achieving recorded-as-cast, tallied-as-recorded, vote secrecy and everlasting voter privacy properties.

Our contributions extend the state of the art by adding the following features.

- Adding a cast-as-intended verification mechanism at the auditing phase.
- Allowing deniable yet verifiable re-voting to mitigate coercion.
- Introducing new proof relations for every stage of the voting process.
- Showing how to modify the scheme for mix-net based voting systems.
- Proof of concept implementation for small-scale elections[1].

Organization. The rest of the paper is structured as follows. We start with explaining the mathematical notation, cryptographic primitives and the protocol by Harrison and Haines in Section 2. Next, we formally define our protocol in Section 3. Section 4 discusses possible modifications to the protocol design, implementation, benchmarking results and further optimizations. Finally, conclusions and future work are given in Section 5.

[1] Available at https://github.com/Valeh2012/starkevoteid

2 Preliminaries

2.1 Notation and primitives

Let λ be the security parameter. Let n denote the number of voters, and let each voter be represented by an index $i = 1, \ldots, n$. If χ is a probability distribution over a set D, both $x \xleftarrow{\chi} D$ and $x \leftarrow \chi$ mean that x is sampled from D with respect to χ. When x is sampled uniformly, we us the notation $x \xleftarrow{\$} D$.

Encryption Schemes. A public-key encryption scheme consists of key generation, encryption and decryption algorithms:

- $\mathcal{E}$.KeyGen(1^λ): On the input of security parameter λ, return the secret key sk and public key pk.
- $\mathcal{E}$.Enc(pk, m): Using the public key pk, compute and output the ciphertext c.
- $\mathcal{E}$.Dec(sk, c): Recover the plaintext message m using the secret key.

In this work, we consider only CPA-secure encryption schemes, i.e. there should be no efficient probabilistic polynomial time algorithm that, given a ciphertext ct and two messages m_0, m_1, can decide if $ct = \mathcal{E}.\text{Enc}(pk, m_0)$ or $ct = \mathcal{E}.\text{Enc}(pk, m_1)$. We also require the additive homomorphic property, i.e. that for all sk, pk, m_0, m_1

$$\mathcal{E}.\text{Dec}(sk, \mathcal{E}.\text{Enc}(pk, m_0) + \mathcal{E}.\text{Enc}(pk, m_1)) = m_0 + m_1$$

holds.

Informally, CPA security implies a randomized encryption algorithm. Therefore, we sometimes use the notation $\mathcal{E}.\text{Enc}(pk, m; r)$ to explicitly specify that r is the used randomness, being sampled from an appropriate probability distribution. Moreover, we define re-randommization and special decryption algorithms that do not depend on the secret key.

- $\mathcal{E}$.ReRand(pk, ct, r'): updates the randomness by adding $\mathcal{E}Enc(pk, 0; r')$ to the given ciphertect ct.
- $\mathcal{E}$.SpecDec(ct, r): having access to the encryption randomness r (but not necessarily the private key!) extract the plaintext (e.g. by full inspection on the plaintext space in case it is small).

Digital Signatures. Similarly, a digital signature scheme is a collection of three algorithms:

- $\mathcal{S}$.KeyGen(1^λ): On the input of security parameter λ, return the private signing key sk and public verification key pk.
- $\mathcal{S}$.Sign(sk, m): Return the signature σ of the message m using the signing key sk.
- $\mathcal{S}$.Verify(pk, m, σ): Output 1 if σ is the signature on m being signed using the signing key that corresponds to the public key pk; otherwise output 0.

Vector Commitment Schemes. Vector Commitments ($\mathcal{VC}$) were first introduced in 2013 by Catalano and Fiore [6]. A $\mathcal{VC}$ scheme allows to commit to many elements, represented as a finite-dimensional vector, at once. Moreover, there should be an efficient opening to a message at a specific index, an update mechanism and a verification method for such updates. Formally, a Vector Commitment scheme $\mathcal{VC}$ is a set of five efficient algorithms KeyGen, Commit, Open, Verify, Update and ProofUpdate.

- $\mathcal{VC}$.KeyGen(1^λ, n): Given the security parameter λ and a vector of size n, output the public parameters pp.
- $\mathcal{VC}$.Commit($pp, m_1, m_2, \ldots m_n$): On the input of public parameters pp and a vector of n elements, output the commitment string C. It is equivalent to the notation $\mathcal{VC}$.Commit(pp, S) where S is a vector of n elements.
- $\mathcal{VC}$.Open(pp, C, m_i, i): Produce a proof Λ_i that a given message m_i is the i-th committed message.
- $\mathcal{VC}$.Verify(pp, C, m_i, i, Λ_i): Return 1 if Λ_i is a valid proof that C opens to m_i at position i; otherwise return 0.
- $\mathcal{VC}$.Update(pp, C, m_i, m_i', i): The committer who produced C updates the commitment string by replacing the i-th message m_i with m_i' and returns C'.
- $\mathcal{VC}$.ProofUpdate(pp, C, Λ_j, m', i): Update the commitment string C and opening proof Λ_j for some message m' at position i. The output of this method is a new commitment string C' and a new opening proof Λ_i'.

A vector commitment scheme $\mathcal{VC}$ is said to be *correct* if for all opening proofs Λ_i, either generated by $\mathcal{VC}$.Open or $\mathcal{VC}$.ProofUpdate, $pp \leftarrow \mathcal{VC}$.KeyGen($1^\lambda, n$), and all commitment strings C that commit to a vector message with i-th element being m_i, $\mathcal{VC}$.Verify(pp, C, m_i, i, Λ_i) is 1 (with overwhelming probability).

The scheme is *position binding* if no polynomial time adversary with the knowledge of pp can find a commitment string C that opens to two distinct messages at any position. Moreover, the scheme is *concise* if the commitment and opening proof lengths are independent of n. We also require *hiding* schemes, meaning that an adversary cannot distinguish whether C is a commitment to $m_1, m_2, \ldots, m_n$ or $m_1', m_2', \ldots, m_n'$ even after seeing some openings.

In [13], the authors give a definition of *Authenticated Data Structures* (ADS) which overlaps with the definition of Vector Commitments above, except for the update mechanism.

Zero-Knowledge Protocols. Let $R = \{(x; w) \mid x \in \mathcal{L}\}$ be a relation for an NP-language $\mathcal{L}$. Here, x is called a *statement* and w is the *witness*. A zero-knowledge proof system is a multi-round interactive protocol between a prover and a verifier which satisfies three properties.

- *Completeness*: Honest verifier accepts an honestly generated proof if $x \in \mathcal{L}$.
- *Soundness*: Cheating prover cannot convince the verifier if $x \notin \mathcal{L}$.
- *Zero-knowledge*: Verifier learns nothing other than the fact that $x \in \mathcal{L}$.

These properties have *perfect*, *statistical* and *computational* variations. Simply speaking, perfect and statistical versions say that the requirements are satisfied with probability 1 or statistically close to 1, respectively, while the computational condition restricts the adversary to be computationally efficient. For soundness, a stronger notion *knowledge-soundness* asserts that if the prover can convince the verifier, then it must know the witness. The zero-knowledge property can be proved by constructing a simulator $\mathcal{S}$ (potentially more powerful than the verifier) which can generate an accepting transcript without the knowledge of w. In case the proof system is only computationally sound, it is called an *argument*.

If there is only a single round of communication, then we speak about a non-interactive zero-knowledge (NIZK) proof. All public coin interactive proofs can be turned into NIZK proofs using Fiat-Shamir transformation [10]. Furthermore, when the proof size is at most polylogarithmic in the witness size, we call it *succinct*.

Recently, many interesting succinct non-interactive arguments of knowledge (SNARK [11, 17, 19]) and succinct transparent arguments of knowledge (STARK [2]) have been proposed. STARKs are quantum-resistant by construction; however they require more computational power than SNARKs. There exist constant-size SNARKs, but all the known STARKs have logarithmic proof size. On the other hand, STARK verification outperforms the most efficient SNARK verifiers in their runtime. Unless explicitly said otherwise, we assume all SNARKs (STARKs) are zero-knowledge instead of denoting them zkSNARKs (zkSTARKs) for distinction.

2.2 The protocol by Harrison and Haines

In [13], Harrison and Haines build a simple e-voting protocol upon the ideas from [8]. Protocol participants are the Election Authority (EA), the voters $V_1, V_2, \ldots, V_n$ and the talliers $T_1, T_2, \ldots, T_m$. There are two append-only bulletin boards: the public bulletin board $\mathcal{PB}$ and the secret bulletin board $\mathcal{SB}$. The talliers also have read-only access to $\mathcal{SB}$.

The EA is responsible for organizing the election. It provides public information such as the list of candidates, the public signature verification keys of eligible voters, the public signature verification keys of valid talliers, and the public voting parameters. It is assumed that an authentic copy of the EA's public signature verification key (pk^{EA}) is known to all participants.

The authors of [13] chose ElGamal encryption scheme over elliptic curves to ensure ballot privacy. The encryption secret key sk is jointly generated by the set of talliers using standard secret sharing techniques and the public key is computed as $PK = sk \cdot G$ (where G is the generator point of the elliptic curve group). Moreover, the talliers keep their own shares of the secret key.

During the voting phase, each voter first verifies public voting parameters, and after that encrypts her vote. The ballot message is a pair consisting of the ciphertext and a NIZK proof of ballot correctness. At last, the voter sends the signed ballot message $\sigma = \mathcal{S}.\mathrm{Sign}(sk, (c, \pi))$ to $\mathcal{SB}$ through a private channel

(here sk is the secret signing key, $c = \mathcal{E}.\text{Enc}(PK, v)$ and π is the ballot correctness proof for the vote v).

After the voting phase ends, the talliers start to count the collected votes. They iterate over the submitted ballots, verify signatures using the public signature verification keys of the corresponding voters, and check that the ballot correctness proofs pass. Next, they add ciphertexts to get the encrypted result $C_j = \sum_{i=1}^{n} c_i$. Now, the results can be decrypted by the talliers using their secret key shares. In addition to this, each tallier commits to the list of voters' public signature keys and added ciphertexts. Following the notation of [13], let $r_p = \mathcal{VC}.\text{Commit}(pp, pk_1, pk_2, \ldots, pk_n)$ and $r_c = \mathcal{VC}.\text{Commit}(pp, c_1, c_2, \ldots, c_n)$. The ciphertext commitment is blinded with a randomness factor $\eta \in \mathbb{Z}_q$. At this point, a STARK proof is generated for the relation

$$R = \left\{ \begin{array}{c} (r_p, r'_c, C); \\ (pk_1, \ldots, pk_n, c_1, \ldots, c_n, \\ \sigma_1, \ldots, \sigma_n, \pi_1, \ldots, \pi_n, \eta) \end{array} \middle| \begin{array}{c} r_p = \mathcal{VC}.\text{Commit}(pp, pk_1, \ldots, pk_n) \\ \wedge\, r'_c = \mathcal{VC}.\text{Commit}(pp, c_1, \ldots, c_n) \cdot \eta \\ \wedge\, \mathcal{S}.\text{Verify}(pk_j, \sigma_j, c_j) = 1 \\ \wedge\, Verify(c_j, \pi_j, PK) = 1\ \forall j = 1, \ldots, n \\ \wedge\, C = \prod_{i=1}^{n} c_i \end{array} \right\}$$

with the public statement (r_p, r'_c, C_j) and private witness $(pk_1, \ldots, pk_n, c_1, \ldots, c_n, \sigma_1, \ldots, \sigma_n, \pi_1, \ldots, \pi_n, \eta)$. Finally, each tallier signs and appends the decryption result and the STARK proof on the public bulletin board $\mathcal{PB}$.

The last stage of this e-voting protocol is called EXTRACT, where the EA verifies all the signatures and included proofs written on the $\mathcal{PB}$, and computes the final result as the sum of extracted partial decryptions. Finally, the EA signs the final result and reveals it.

For the implementation, Harrison and Haines choose ElGamal over STARK elliptic curve [21], Pedersen hash function [15, Section 5.4.1.7], Elliptic Curve Digital Signature Algorithm (ECDSA) [16], Merkle Trees as the vector commitment, and ElectionGuard's ballot correctness proof [3].

The authors mention that the contribution is not the protocol definition itself, but presentation and evaluation of the STARK proof as an efficient alternative to verify talliers' work without publicly exposing the submitted ballots. Therefore, they do not prove the informal claim that, assuming the vector commitment is secure, binding and hiding, and the non-interactive proofs are complete, sound and zero-knowledge, the proposed e-voting protocol satisfies everlasting privacy, ballot privacy, recorded-as-cast and tallied-as-recorded properties.

3 Our vector commitment-based Voting Protocol

In this section, we formally define our online voting protocol based on vector commitments, arguing that the protocol is end-to-end verifiable and coercion resistant. For the latter property, the ballots are stored privately on a private bulletin board, and re-voting is allowed. Assuming the coercer does not block the voters from submitting their choices, but only demands a proof that the submitted ballots encode the coercer's wish, the voters can generate such proofs

at the Cast-as-Intended phase, and later vote again. If the coercer does not have access to the submitted ballots, he cannot tell if the voter has voted again, and hence the coercive strategy would fail.

However, having a private bulletin board raises concerns about the correct processing of the recorded ballots. Therefore we propose a second append-only bulletin board which is public and contains auxiliary information about each submitted ballot. This information must be verifiable by anyone for universal verifiability. It also has a purpose of linking two consecutively received ballots. Furthermore, the last entry on the public bulletin board guarantees the order of ballot history and is used at the later stages of the election to guarantee that the ballots are recorded-as-cast.

Before describing the proposed protocol explicitly, we first give a brief list of the used methods.

3.1 Building blocks

Let $\mathcal{E}$ be a CPA-secure additively homomorphic encryption scheme to encrypt the votes, let $\mathcal{S}$ be a digital signature scheme to sign the ballots, and let $\mathcal{VC}$ be a position binding vector commitment scheme to commit to the submitted ballots. Let n be the number of eligible voters and let $PK^{\mathcal{S}} = \{pk_1^{\mathcal{S}}, pk_2^{\mathcal{S}}, \ldots, pk_n^{\mathcal{S}}\}$ be the set of the voters' public keys. Let Υ be the set of possible choices; e.g., for simple yes/no elections the set $\Upsilon = \{0, 1\}$ can be used.

Ballot correctness proof A ballot correctness proof, or simply a ballot proof, is a non-interactive zero-knowledge proof of knowledge of intent, generated by each voter to show that their choice under encryption satisfies election rules. Let $c = \mathcal{E}.\mathrm{Enc}(pk^{\mathcal{E}}, v)$ be an encrypted ballot where $v \in \Upsilon$. A ballot proof Π_{BP} is a NIZK for the following relation:

$$R_{BP} = \left\{ (c, pk^{\mathcal{E}}, \Upsilon); (v, r) \;\middle|\; c = \mathcal{E}.\mathrm{Enc}(pk^{\mathcal{E}}, v; r) \wedge v \in \Upsilon \right\}.$$

Update proof Let $C = \{c_1, c_2, \ldots, c_j, \ldots, c_n\}$ be the set of ballots submitted by the voters at some moments in time, and let the voter j submit a new signed ballot $\sigma = \mathcal{S}.\mathrm{Sign}(sk_j^{\mathcal{S}}, (c'_j, \Pi_{PB}))$. Furthermore, let $\Lambda_j = \mathcal{VC}.\mathrm{Open}(pp, VC, c, j)$ and $(\Lambda'_j, VC') = \mathcal{VC}.\mathrm{ProofUpdate}(pp, VC, \Lambda_j, c', j)$ be opening proofs to the ballots before and after this submission at position j. Update proof Π_U is a NIZK for the following relation:

$$R_U = \left\{ \begin{array}{c} (pp, VC, VC', \\ PK^{\mathcal{S}}, \Upsilon); (j, c_j, \sigma, \\ c'_j, \Pi_{PB}, pk_j^{\mathcal{S}}, \Lambda_j, \Lambda'_j) \end{array} \;\middle|\; \begin{array}{c} \Pi_{BP}.\mathrm{Verify}(c'_j, pk^{\mathcal{E}}, \Upsilon) = 1 \;\wedge\; pk_j^{\mathcal{S}} \in PK^{\mathcal{S}} \\ \wedge\, \mathcal{S}.\mathrm{Verify}(pk_j^{\mathcal{S}}, (c'_j, \Pi_{BP}), \sigma) = 1 \\ \wedge\, \mathcal{VC}.\mathrm{Verify}(pp, VC, c_j, j, \Lambda_j) = 1 \\ \wedge\, \mathcal{VC}.\mathrm{Verify}(pp, VC', c'_j, j, \Lambda'_j) = 1 \wedge \\ (\Lambda'_j, VC') = \mathcal{VC}.\mathrm{ProofUpdate}(pp, VC, \Lambda_j, c', j) \end{array} \right\}.$$

Opening proof While $\mathcal{VC}$.Open method efficiently generates a proof for an opening at a particular position, generating such proofs for every position and verifying them would take a substantial amount of time at the beginning of the election. Instead, a NIZK proof for the relation $(VC; C)$ where $VC = \mathcal{VC}.\text{Commit}(pp, C)$ can be generated so that verifying it would take only marginal amount of time. This relation can be extended with additional constraints. For example, it may be desirable to let C be an empty set, a set of zeros, or zero-encryptions. For the latter, we formally define the opening proof relation as

$$R_{\perp} = \left\{ (pp, pk^{\mathcal{E}}, VC); C \;\middle|\; \begin{array}{c} VC = \mathcal{VC}.\text{Commit}(pp, C) \\ \wedge\, c = \mathcal{E}.\text{Enc}(pk^{\mathcal{E}}, 0) \quad \forall c \in C \end{array} \right\}.$$

Accumulation proof Again, consider that VC is a vector commitment to $C = \{c_1, c_2, \ldots, c_n\}$ before tallying starts. In case of homomorphically tallied elections, ciphertexts are added before the decryption phase. For that, we define accumulation proof Π_{Acc} for the relation

$$R_{acc} = \left\{ (pp, VC, c_{acc}); C \;\middle|\; VC = \mathcal{VC}.\text{Commit}(pp, C) \;\wedge c_{acc} = \sum_{c \in C} c \right\}.$$

Decryption proof The final NIZK proof, Π_{Dec}, guarantees that the plaintext m is a correct decryption of c under the key $sk^{\mathcal{E}}$. We can write the relation formally as

$$R_{Dec} = \left\{ (c, m, pk^{\mathcal{E}}); sk^{\mathcal{E}} \;\middle|\; m = \mathcal{E}.\text{Dec}(sk^{\mathcal{E}}, c) \wedge (pk^{\mathcal{E}}, sk^{\mathcal{E}}) \leftarrow \mathcal{E}.KeyGen(1^{\lambda}) \right\}.$$

3.2 Full protocol

Our voting protocol consists of the following components:

- *Announcement*, *Registration*, *Setup*, *Casting* and *Tallying* phases;
- **Election Setup**, **KeyGen**, **Vote**, **Verify** and **Tally** protocols;
- append-only databases called Private Bulletin Board $\mathcal{SB}$, and Public Bulletin Board $\mathcal{PB}$;
- participants running the protocol: the Election Authority EA, Voters V_j, Vote Collector Server (VCS), and Decryption Authority $O^{\mathcal{D}}$.

The protocol proceeds as follows.

- *Announcement:* First, the EA decides upon a CPA-secure asymmetric encryption scheme $\mathcal{E}$, digital signature scheme $\mathcal{S}$ and vector commitment scheme $\mathcal{VC}$ to use, initializes $\mathcal{PB}$ and $\mathcal{SB}$, announces the election parameters and the chosen cryptographic suite.
- *Registration:* Every potential voter V_j generates her own pair of public and private signing keys $(pk_j^{\mathcal{S}}, sk_j^{\mathcal{S}}) = \mathcal{S}.\text{KeyGen}(1^{\lambda})$, and registers herself by sending the pk_j^S to the $\mathcal{PB}$. Upon receiving the public signing key from

the voter, $\mathcal{PB}$ adds it to the list of public keys $PK^{\mathcal{S}} = PK^{\mathcal{S}} \cup \{pk_j^{\mathcal{S}}\}$ and increments the number of voters n. Thus, at the end of the registration phase, we have $n = |PK^{\mathcal{S}}|$.

- *Setup:* After the Registration phase ends, $O^{\mathcal{D}}$ generates the encryption-decryption key pair $(pk^{\mathcal{E}}, sk^{\mathcal{E}}) = \mathcal{E}.\text{KeyGen}(1^{\lambda})$, and sends the public encryption key to EA. EA runs the vector commitment scheme setup $pp = \mathcal{VC}.\text{Setup}(1^{\lambda}, n)$, and appends an empty commitment $VC = \mathcal{VC}.\text{Com}(pp, C^0)$ where $C^0 = (c_i^0)_{i=1}^n$ and c_i^0 is an encryption of zero message for all $i = 1, \ldots, n$, along with a proof of opening $\Pi_{\perp}$ to $\mathcal{PB}$.
- *Casting:* The voter V_j
  - samples randomness $r \xleftarrow{\$} R$ (where R is the randomness space chosen as a part of the public parameters),
  - encrypts her vote $v \in \Upsilon$ using randomness r and election public encryption key as $c' = \mathcal{E}.\text{Enc}(pk^{\mathcal{E}}, v; r)$,
  - generates the ballot correctness proof Π_{BP} for the relation $((c', pk^{\mathcal{E}}, \Upsilon); (v, r))$,
  - signs the ciphertext using her secret signing key as $\sigma = \mathcal{S}.\text{Sign}(sk_j^{\mathcal{S}}, (c', \Pi_{BP}))$,
  - submits the ballot (σ, c', Π_{BP}) to the Vote Collector Server.

  When VCS receives the ballot (σ, c', Π_{BP}), it checks that the validity of the signature and the ballot proof as the first step. If they pass, VCS
  - updates the vector commitment $VC' = \mathcal{VC}.\text{Update}(pp, VC, c, c', j)$ at position j with the new value c', and
  - generates the opening proof $\Lambda' = \mathcal{VC}.\text{Open}(pp, VC', c', j)$.
  - Assuming VC, Λ and c are the previous vector commitment, an opening proof and its opening at the same position, respectively, VCS generates the proof of valid commitment update Π_U of $((pp, VC, VC', PK^{\mathcal{S}}); (j, c, \sigma, c', \Pi_{BP}, pk_j^{\mathcal{S}}, \Lambda, \Lambda')) \in R_U$.
  - Last, VCS appends $(\sigma, c', \Pi_{BP}, \Lambda')$ to $\mathcal{SB}$, and (Π_U, VC') to $\mathcal{PB}$.
  - Returns unique vote reference vr to the voter.
- *(Homomorphic) Tallying:* At the final phase, EA runs the Tally protocol on the input of all update proofs $(\Pi_U^{(i)})_{i=1}^{\nu}$ and history of the vector commitments $(VC^{(i)})_{i=1}^{\nu}$ and committed ciphertexts $(c_j)_{j=1}^n$. Within the Tally protocol, first all update proofs are verified, checking that indeed VC^{ν} is the correct vector commitment to $(c_j)_{j=1}^n$. Then, all the ciphertexts are added as $c_{acc} = \sum_{j=1}^n c_j$, and a proof of accumulation Π_{Acc} is produced. Last, EA queries the Decryption Authority $O^{\mathcal{D}}$ to decrypt c_{acc}. $O^{\mathcal{D}}$ returns the decrypted tally and proof of correct decryption next to it. EA publishes the tally and all supporting proofs to $\mathcal{PB}$.

3.3 Individual Verifiability

Up to this point, our protocol achieves recorded-as-cast and tallied-as-recorded properties, assuming that the vector commitment is correct and positional binding, proofs are knowledge sound and zero-knowledge. The former follows directly

from the soundness of update proofs and positional binding property of the vector commitment scheme, while the latter is satisfied due to the accumulation proof being sound. Vote secrecy follows from the CPA-security of the encryption scheme. Ballot signatures confirm eligibility of the voters. Finally, voter anonymity is preserved as the update proofs are zero-knowledge.

However, in the presented form, there is no mechanism to verify the cast-as-intended property (that is, we are not yet doing better than [13]). To add this functionality, we expand the protocol with the *Audit* phase, applying cast-and-audit strategy. That is, the voters can query the submitted ballot using a separate audit device and verify that the encrypted ballot contains their vote after they have finished voting (following the approach proposed by Heiberg and Willemson [14]). In addition, we assume that once the VCS returns the a unique vote reference vr to the voter, that reference is not leaked. The voting device used by the voter then displays this vote reference together with the encryption randomness (e.g. as a QR code).

Let c' be encryption of the vote v using randomness r, i.e, $c' = \mathcal{E}.\mathrm{Enc}(pk^{\mathcal{E}}, v; r)$, and let the VCS return the vote reference vr. The voter imports vr and r into the audit device (for example, by scanning the QR code). Then the audit device queries VCS to get the encrypted ballot c', proof of update Π_U, and the proof of opening Λ'. Next, it verifies Π_U and checks that $\mathcal{VC}.\mathrm{Verify}(pp, VC', c', i, \Lambda') = 1$. Note that the position i can be provided by VCS within Λ or obtained by the voter herself simply by looking at the index of her public signing key in the list PK^S. If both verification checks pass, the audit device attempts to extract the vote v^* using special decryption algorithm as $v^* = \mathcal{E}.SpecDec(c', r)$. The voter now can verify if $v = v^*$. If VCS is honest, $v = \mathcal{E}.SpecDec(\mathcal{E}.\mathrm{Enc}(pk^{\mathcal{E}}, v; r), r) = v^*$ holds due to correctness of the encryption algorithm, and the voter accepts the verification result. If $v \neq v^*$, the voter rejects. As a result, our proposed voting protocol achieves the cast-as-intended property. Moreover, as (Π_U, VC') pair is on $\mathcal{PB}$, and assuming the zero-knowledge proof is sound and the vector commitment scheme is complete, the voter is convinced that her vote is cast-as-intended and recorded-as-cast. Combining this with the tallied-as-recorded property, we get end-to-end verifiability.

We observe that our cast-as-intended and recorded-as-cast verification forms a receipt of how the voter voted. However, due to allowing for re-voting, the risk of coercion is mitigated. Assuming the coercer has no access to the communication between the voters and the VCS, or cannot read the contents of the private bulletin board, it is not possible for him to to tell whether the voter has re-voted or not. In addition, the update proofs do not leak any information regarding the voter. Therefore the coercer can not simply see whether a voter has voted or not, so the protocol also provides protection against forced abstention attacks.

4 Discussion

The protocol described in Section 3 can be easily augmented to comply with different election requirements. For example, currently there is a single author-

ity who holds the private encryption key, and the authority is modeled as an oracle. This is a threat to vote secrecy, because the tallier may query the oracle arbitrarily many times to learn the contents of each ballot. The trivial solution would be limiting access to the oracle to only one query; however, in practice, it would be hard to enforce this rule. Malicious tallier and decryption oracle can collude to decrypt any submitted ballot. The standard solution is to split the private decryption key into a number of shares and distribute them among several registered talliers. Since each tallier can keep their share of the private key, there is no need for a separate decryption oracle. In the modified tallying phase, each tallier will run the Tally protocol, decrypting the accumulated ciphertext and generating the proof of correct decryption by themselves, in parallel. As a result, each tallier will publish their share of decryption. If at least one tallier is honest and does not collude with others, vote secrecy will be retained.

We can also consider more general algorithms for computing election result. For example, a mix-net version of the protocol given in Section 3.2 is relatively straightforward to implement. We can add a *Mixing* phase before *Tallying*, and a Mixer will run the **Mixing** protocol at this phase.

- The *Mixing* phase will consist of the following steps:
  - Verify all the previous proofs.
  - Get a list of ciphertexts from $\mathcal{SB}$.
  - Re-randomize each ciphertext with fresh randomness.
  - Shuffle the ciphertexts using a secret permutation, commit to them.
  - Generate NIZK proof of correct shuffle, $\Pi_{shuffle}$.
  - Publish ciphertexts on $\mathcal{PB}$ in the shuffled order together with $\Pi_{shuffle}$,

where $\Pi_{shuffle}$ is a proof for the shuffle relation $R_{shuffle}$:

$$R_{shuffle} = \left\{ \begin{array}{c} ((pp, pk^{\mathcal{E}}, VC, \\ c'_1, \ldots, c'_n); \\ (\pi, c_1, \ldots, c_n, \\ r_1, \ldots, r_n)) \end{array} \middle| \begin{array}{c} VC = \mathcal{VC}.\text{Commit}(pp, c_1, \ldots, c_n) \wedge \\ c'_{\pi(i)} = \mathcal{E}.\text{ReRand}(pk^{\mathcal{E}}, c_i, r_i)\ \forall i = 1, \ldots, n \end{array} \right\}$$

In this version, the talliers do not need to add ciphertexts and produce the accumulation proof. However, they have to decrypt all the ciphertexts one by one and produce the accompanying proofs of correct decryption. Note that because the Mixer gets input directly from the private bulletin board, this input is not part of the statement, but rather part of the witness in the shuffle proof. Regular mix-nets, however, treat incoming and outgoing ciphertexts as public values. Finally, one can add many regular mix-nets after the Mixer to further reduce the link between the voter and her vote. All such modifications do not change the main properties of the voting system.

Finally, it is important to mention that who can generate valid signatures on behalf of the voter, can also submit ballots that would go undetected by the voter since the update proofs do not leak voter identity. Therefore, we assume a trusted signing device that hides private signing keys, and require confirmation

from the voter every time she has to sign a document. In practice, this can be realized by separate hardware (smart cards[2]) or software (digital wallets).

4.1 Implementation and benchmarks

The implementation details of the original Harrison and Haines scheme were presented in Section 2.2. We use the same components to instantiate our protocol. That is, we consider simple YES/NO election with one question, ElGamal encryption scheme and corresponding NIZK ballot proof, ECDSA digital signature, Merkle Tree with Pedersen hash function as the vector commitment scheme and STARK proof system. All these schemes are defined over the STARK curve. We perform membership checks for public signature keys using another Merkle Tree.

We have extended the source code provided in [13], and added circuits for relations defined in Section 3.1. All the circuits are written in Cario 0 language, and proofs are generated using the STONE prover. Our test device has a 4-core (8-threads) Intel(R) Core(TM) i5-82500 CPU running at 1.60 GHz, and 16 GB of RAM. The operating system is Ubuntu 24.04.2 LTS.

First, we simulated the vote casting phase with up to 2^{32} voters. In all the experiments, the (average) times required to generate and verify a single update proof remained constant, being 0.7 **core-minutes** and 0.15 **core-seconds**, respectively. As the proof file contains public input, its size grew linearly from 980 KB to 1.0 MB. Peak memory usage was consistently less than 1 GB.

Next, we measured the performance of the accumulation circuit for different numbers of voters. Our implementation reached its memory limit with 1024 voters. In this case, it takes 20.8 **core-minutes** to generate the 1.8 MB accumulate proof, and only 0.21 **core-seconds** to verify the proof with peak memory usage 11.82 GB. In comparison, Harrison and Haines were only able to accommodate up to 64 voters at the similar RAM usage, but slightly longer proof generation time. We summarize all the experiment results in Table 1, with projections for bigger inputs given in gray. Our work noticeably outperforms [13] in all the input sizes.

For a fair comparison, we considered the performance of our protocol with 2^{24} voters. In [13], the authors estimate the total proof size to be 1.54 GB, and the verification to take 5.62 **core-minutes** (after adjusting for clock frequency) if batching is applied (512 batches with 2^{15} batch size). With this configuration, the total proof size for accumulation proofs using our protocol would be 1.14 GB and the verification would take 2.13 **core-minutes**. Moreover, less than 0.5 TB RAM is enough in contrast to the assumed 6 TB for the tallier machine in [13]. The downside of our proposed scheme is that minimum 2^{24} MB $\approx$ 16 TB disk space is needed to store the individual update proofs, and verifying them sequentially requires 41 **core-days**. This is the cost of having verifiable vote updating.

[2] In case of smart cards, voters are advised to use smart card readers with keypad to avoid PIN caching attacks

Input	Π_U				Harrison and Haines [13]			
Votes	Proving	Verification	Size	Peak RAM	Proving	Verification	Size	Peak RAM
4	0.19	0.11	0.542	0.57	1.77	0.18	1.009	0.72
8	0.22	0.14	0.901	0.62	3.23	0.16	1.062	1.38
16	0.35	0.14	0.923	0.73	6.46	0.17	1.148	2.82
32	0.64	0.15	1.014	0.95	12.91	0.17	1.317	5.50
64	1.15	0.15	1.047	1.38	26.25	0.22	1.620	11.0
128	2.37	0.16	1.113	1.84	52.35	0.23	1.824	21.96
256	4.78	0.17	1.227	3.97	104.73	0.24	1.972	43.88
512	10.16	0.18	1.431	7.42	209.49	0.25	2.120	87.73
1024	20.8	0.21	1.835	11.82	419.01	0.27	2.267	175.41
2048	41.36	0.21	1.737	24.00	838.04	0.28	2.415	350.79
4096	82.80	0.22	1.861	47.33	1676.10	0.29	2.563	701.53
8192	165.68	0.23	1.987	93.99	3352.22	0.31	2.711	1403.03
16384	331.4	0.24	2.111	187.30	6704.46	0.33	2.855	2803.02
32768	662.95	0.25	2.235	373.93	13408.94	0.34	3.004	5612.01

Table 1. Comparing performance of accumulate proof generated by the tallier. Entries in gray are projected results. Units for Proving, Verification, Size and Peak RAM columns are in core-minutes, core-seconds, MB and GB, respectively.

4.2 Post-Quantum Instantiation

Unfortunately, lattice-based encryption and digital signature schemes are not STARK-friendly. Therefore, we choose Labrador [4] proof system and Falcon digital signatures, whose verification has been implemented in [1]. Next, BGV [5] without bootstrapping is a good choice for an additively homomorphic encryption scheme. Finally, Merkle Trees with any post-quantum secure hash function stays as the vector commitment scheme. Choosing parameters to satisfy e.g. the 128-bit security level is currently an open problem, and it is not a straightforward task due to non-trivial interdependencies between the parameters.

4.3 Further Optimizations

Even though the proposed proof systems have extremely fast verification times, for mid- and large-scale elections, downloading and verifying update proofs may take several hours. Luckily, there is an elegant solution – Incrementally Verifiable Computation (IVC) [23], a recent breakthrough in zero-knowledge proof systems. The main idea behind IVC is to recursively generate a new proof for a certain relation and verify the previous proof as a subtask inside the new proof. If we think of generating the proof of update as an incremental function F that takes the input statement and the witness, we can generate a single succinct proof at the end of the voting phase. This will drastically reduce the workload of external verifiers. In practice, this increases memory usage and proof generation time at the prover's side. In other words, the Vote Collector Server should be very powerful in order to provide a seamless voting experience.

5 Conclusion

This paper presents a possible approach to designing an end-to-end verifiable and coercion-resistant voting protocol with cast-and-audit individual verification and re-voting. Our work is inspired by [13]; however, our construction is based on a verifiably updatable vector commitment scheme to bind all submitted ballots to the election outcome. All the voting protocol participants produce associated zero-knowledge proofs which are verifiable by anyone. A simple cast-as-intended verification ensures voters that their interaction with the election is not tampered with. We also allow for re-voting to mitigate coercion attacks, while promising that the last vote is included in the tally. We achieve these results by assuming voters generate digital signatures using a trusted device. Simple proof of concept implementation shows that the scheme is deployable in practice. With further optimizations, it is possible to reduce the number of produced artifacts, so the workload of external verifiers. We leave such optimizations and post-quantum implementation a possible direction for future works.

Acknowledgments

The paper has been supported by the Estonian Research Council under the grant number PRG2177.

References

1. Aardal, M.A., Aranha, D.F., Boudgoust, K., Kolby, S., Takahashi, A.: Aggregating Falcon Signatures with LaBRADOR. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology – CRYPTO 2024. pp. 71–106. Springer Nature Switzerland (2024)
2. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019. pp. 701–732. Springer International Publishing, Cham (2019)
3. Benaloh, J., Naehrig, M.: Electionguard design specification (version 2.0.0). Tech. rep., Microsoft Research (2023)
4. Beullens, W., Seiler, G.: LaBRADOR: Compact Proofs for R1CS from Module-SIS. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023. pp. 518–548. Springer Nature Switzerland, Cham (2023)
5. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully homomorphic encryption without bootstrapping. Electron. Colloquium Comput. Complex. **TR11** (2011), https://api.semanticscholar.org/CorpusID:2182541
6. Catalano, D., Fiore, D.: Vector commitments and their applications. In: Kurosawa, K., Hanaoka, G. (eds.) Public-Key Cryptography – PKC 2013. pp. 55–72. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
7. Cortier, V., Galindo, D., Küsters, R., Müller, J., Truderung, T.: SoK: Verifiability Notions for E-Voting Protocols. In: IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016. pp. 779–798. IEEE Computer Society (2016). https://doi.org/10.1109/SP.2016.52
8. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. Eur. Trans. Telecommun. **8**(5), 481–490 (1997). https://doi.org/10.1002/ETT.4460080506

9. Cuvelier, É., Pereira, O., Peters, T.: Election verifiability or ballot privacy: Do we need to choose? In: Computer Security – ESORICS 2013. pp. 481–498. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
10. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) Advances in Cryptology — CRYPTO' 86. pp. 186–194. Springer Berlin Heidelberg, Berlin, Heidelberg (1987)
11. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct nizks without pcps. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology – EUROCRYPT 2013. pp. 626–645. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
12. Haines, T., Müller, J., Querejeta-Azurmendi, I.: Scalable coercion-resistant e-voting under weaker trust assumptions. In: Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing. p. 1576–1584. SAC '23, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3555776.3578730
13. Harrison, M., Haines, T.: On the applicability of starks to counted-as-collected verification in existing homomorphic e-voting systems. In: Financial Cryptography and Data Security. FC 2024 International Workshops. pp. 50–65. Springer Nature Switzerland, Cham (2025)
14. Heiberg, S., Willemson, J.: Verifiable Internet voting in Estonia. In: Krimmer, R., Volkamer, M. (eds.) 6th International Conference on Electronic Voting: Verifying the Vote, EVOTE 2014, Lochau / Bregenz, Austria, October 29-31, 2014. pp. 1–8. IEEE (2014). https://doi.org/10.1109/EVOTE.2014.7001135
15. Hopwood, D., Bowe, S., Hornby, T., Wilcox, N.: Zcash protocol specification 2022.3.8 [nu5]. Tech. rep., Electric Coin Company (2022)
16. Johnson, D., Menezes, A., Vanstone, S.A.: The elliptic curve digital signature algorithm (ECDSA). Int. J. Inf. Sec. **1**(1), 36–63 (2001). https://doi.org/10.1007/S102070100002
17. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing. p. 723–732. STOC '92, Association for Computing Machinery, New York, NY, USA (1992). https://doi.org/10.1145/129712.129782
18. Lueks, W., Querejeta-Azurmendi, I., Troncoso, C.: VoteAgain: A scalable coercion-resistant voting system. In: 29th USENIX Security Symposium (USENIX Security 20). pp. 1553–1570. USENIX Association (Aug 2020), https://www.usenix.org/conference/usenixsecurity20/presentation/lueks
19. Micali, S.: Cs proofs. In: Proceedings 35th Annual Symposium on Foundations of Computer Science. pp. 436–453 (1994). https://doi.org/10.1109/SFCS.1994.365746
20. Mueller, J., Pejo, B., Pryvalov, I.: DeVoS: Deniable yet verifiable vote updating. Cryptology ePrint Archive, Paper 2023/1616 (2023), https://eprint.iacr.org/2023/1616
21. StarkEx: STARK curve. https://docs.starkware.co/starkex/crypto/stark-curve.html (2025), [Accessed 22-01-2025]
22. Vakarjuk, J., Snetkov, N., Willemson, J.: Comparing security levels of postal and Internet voting. Information Security Journal: A Global Perspective pp. 1–21 (2024). https://doi.org/10.1080/19393555.2024.2410332
23. Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: Canetti, R. (ed.) Theory of Cryptography. pp. 1–18. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)