Athena: A verifiable, coercion-resistant voting system with linear complexity

Ben Smyth

July 5, 2019

Abstract
Seminal work by Juels, Catalano & Jakobsson delivered a verifiable, coercion-resistant voting system with quadratic complexity. This manuscript attempts to advance the state-of-the-art by delivering a voting system with equivalent security and linear complexity.

1 Introduction
Voting systems must ensure free-choice [49, 50, 63]. A notion of which is formalised by ballot secrecy (i.e., a voter’s vote is not revealed to anyone) [8, 9, 11, 17, 58]. This notion can be satisfied by voting systems that simply instruct voters to encrypt their vote. But, free-choice may be compromised by an adversary that is able to communicate with voters, since the coins used for encryption serve as proof of how voters voted and voters may communicate those coins to the adversary. Hence, formulations of free-choice must be accompanied by operational assumptions and limitations on the adversary’s capabilities. Indeed, ballot secrecy assumes that voters’ ballots are constructed and tallied in the prescribed manner, and that the adversary’s capabilities are limited to controlling ballot collection.

Receipt-freeness (i.e., a voter cannot collaborate with a conspirator to gain information which can be used to prove how they voted) formalises a notion of free-choice in the presence of an adversary that can communicate with voters [10, 14, 25, 42, 48]. Yet free-choice may be compromised if voters deviate from the prescribed voting procedure. Coercion-resistance (i.e., a voter can deviate from a coercer’s instructions, to cast their own vote, without detection) formalises a stronger notion of free-choice assuming that not only can voters deviate, but the adversary can instruct voters how to deviate [36, 45, 64]. The distinction between receipt-freeness and coercion-resistance is subtle: “receipt-freeness deals with a [conspirator] who is only concerned with deducing information about how someone voted from receipts and public information, but who does not give detailed instructions on how to cast the vote. Coercion resistance, on the other hand, includes dealing with a coercer who gives details not just...
on which candidate to vote for but also on how to cast the vote” [33, §1.1]. Both receipt-freeness and coercion-resistance retain the assumption that voters’ ballots are tallied in the prescribed manner, and receipt-freeness additionally assumes voters’ ballots are constructed in the prescribed manner.

Beyond free-choice, voting systems must ensure that only voters vote [49, 50, 63], which can be achieved by issuing credentials to voters and using cryptography to ensure that authorised ballots are unforgeability (i.e., only voters can construct authorised ballots) [59, 61],¹ (Unforgeability is sometimes known as eligibility verifiability.) Moreover, voting systems must ensure that voters have equal influence in the decision [49, 50, 63], which can be achieved by universal verifiability (i.e., anyone can check whether an outcome corresponds to votes expressed in collected ballots that are authorised, except for votes expressed in ballots from the same voter, which are all discarded, except for the voter’s last vote) and individual verifiability (i.e., a voter can check whether their ballot is collected) [22, 42–44, 61].

Seminal work by Juels, Catalano & Jakobsson [35–37] made significant progress towards a voting system satisfying the aforementioned properties, moreover, Clarkson, Chong & Myers [20, 21] implemented their results as Civitas, albeit, complexity is $O(|bb|^2)$, i.e., quadratic in the length of the bulletin board ($bb$). Quadratic complexity arises from the use of pairwise plaintext equality tests on the bulletin board’s ballots to discard all but the last vote cast using a private credential.² Pairwise plaintext equality tests are also used on mixed ballots and mixed public credentials to discard mixed ballots that are unauthorised, with complexity $O(|L| \cdot |bb|)$, where $L$ is the electoral roll.

**Contribution.** We advance the state-of-the-art with Athena: A verifiable, coercion-resistance voting system with linear complexity $O(|bb|)$. Our system reveals anonymised credentials to discard ballots cast using the same private credential (with linear complexity) and uses plaintext equality tests on each individual mixed ballot – which includes a mixed public credential – to discard any mixed ballot that is unauthorised (with linear complexity). Athena works as follows.

**Voting.** Voters are issued with credential pairs, wherein the private credential is a nonce and the public credential is an encryption of that nonce. Each voter encrypts the negation of their private credential and their vote, and publishes the two resulting ciphertexts prepended with their public credential and appended with a counter (to the bulletin board). A voter computes ballots for any re-votes similarly, using an incremented counter. It follows that the bulletin board will contain voters’ ballots, plus any adversarial ballots.

**Tallying.** Any ballots not containing a public credential are discarded, the sec-

¹ Auditing can statistically determine whether non-voters are issued with credentials.
² A plaintext equality test (PET) is a cryptographic primitive for determining whether two ciphertexts encrypt the same plaintext [34].
ond ciphertext of each remaining ballot is homomorphically combined with itself \( n \)-times (for some nonce \( n \)), and the resulting combination is decrypted to reveal an anonymised credential. Entries that share an anonymised credential are discarded, except for the one with the highest counter, thus, only the last vote associated with each anonymised credential is retained. The first two ciphertexts of each retained ballot are homomorphically combined, deriving either: 1) the combination of a private credential and the negation of that credential, or 2) the combination of a private credential and some other message (excluding the credential’s negation). The resulting homomorphic combinations and corresponding encrypted votes are mixed (using the same permutation), plaintext equality tests are used to determine whether the mixed homomorphic combinations where constructed using private credentials, and the corresponding mixed encrypted votes are decrypted if they were.

Intuitively, Athena achieves coercion-resistance, because a well-formed ballot that encrypts the negation of a voter’s private credential is indistinguishable from an ill-formed ballot that encrypts some other message, hence, a voter cannot prove whether they cast a well-formed ballot (that will be counted, as opposed to an ill-formed ballot that will not), during the voting phase. Moreover, mixing ensures that ballots cannot be mapped to votes during tallying. Thus, coercion-resistance is achieved. (Unlike the voting system by Juels, Catalano & Jakobsson, Athena reveals the number of ballots cast using a voter’s public credential, which requires voters to deny casting ballots when instructed by the coercer to abstain. This is a reasonable strategy, since no voter can prove whether they even cast a well-formed ballot.) Moreover, verifiability is achieved too, because only voters have access to private credentials, hence, only voters can construct authorised ballots (unforgeability), tallying produces evidence (specified in Definition 2) demonstrating that election outcomes correspond to the votes expressed in collected ballots that are authorised (universal verifiability), and ballots are recorded on a bulletin board, hence, voters can check whether their ballot is collected (individual verifiability).

**Structure.** We define Athena in Section 3, analyse coercion-resistance in Section 4, prove verifiability in Section 5, present complexity results in Section 6, and study general design principles in Section 7. The remaining sections present syntax (§2), related work (§8), and a brief conclusion (§9), Sidebar 1 introduces game-based security definitions and recalls notation, and the appendices define cryptographic primitives along with relevant security definitions and present proofs.

## 2 Election scheme syntax

We extend syntax by Smyth, Frink & Clarkson [61] to include re-voting, thereby capturing voting systems that consist of the following four steps. First, a tallier generates a key pair and a registrar generates credentials for voters. Secondly,
Sidebar 1 Preliminaries: Games and notation [58]

A game formulates a series of interactions between a benign challenger, a malicious adversary, and a cryptographic scheme. The adversary wins by completing a task that captures an execution of the scheme in which security is broken, i.e., winning captures what should be unachievable. Tasks can generally be expressed as indistinguishability or reachability requirements. For example, universal verifiability can be expressed as the inability to reach a state that causes a voting system’s checks to succeed for invalid election outcomes, or fail for valid outcomes. Moreover, ballot secrecy can be expressed as the inability to distinguish between an instance of a voting system in which voters cast some votes, from another instance in which the voters cast a permutation of those votes. Formally, games are probabilistic algorithms that output booleans. We introduce some notation before formalising games.

We let \( A(x_1, \ldots, x_n; r) \) denote the output of probabilistic algorithm \( A \) on inputs \( x_1, \ldots, x_n \) and coins \( r \), and we let \( A(x_1, \ldots, x_n) \) denote \( A(x_1, \ldots, x_n; r) \), where coins \( r \) are chosen uniformly at random from the coin space of algorithm \( A \). Moreover, we let \( x \leftarrow T \) denote assignment of \( T \) to \( x \); \( x \leftarrow A(x_1, \ldots, x_n) \) denote assignment of \( A(x_1, \ldots, x_n; r) \) to \( x \), where coins \( r \) are chosen uniformly at random from the coin space of algorithm \( A \); and \( x \leftarrow_R S \) denote assignment to \( x \) of an element chosen uniformly at random from set \( S \) (we use the same notation when \( S \) is a distribution). Furthermore, we let \( x[i] \) denote component \( i \) of vector \( x \) and let \( |x| \) denote the length of vector \( x \). Finally, we write \((x_1, \ldots, x_{|T|}) \leftarrow T \) for \( x \leftarrow T; x_1 \leftarrow x[1]; \ldots; x_{|T|} \leftarrow x[|T|] \); when \( T \) is a vector, and \( x, x' \leftarrow_R S \) for \( x \leftarrow_R S; x' \leftarrow_R S \).

Using our notation, we can formulate the following game \( \text{Exp} \) such that \( \text{Exp}(H, S, A) \) tasks an adversary \( A \) to distinguish between a function \( H \) and a simulator \( S \): \( m \leftarrow A()\); \( \beta \leftarrow_R \{0, 1\} \); if \( \beta = 0 \) then \( x \leftarrow H(m) \); else \( x \leftarrow S(m) \); \( g \leftarrow A(x) \); return \( g = \beta \). Adversaries are stateful, i.e., information persists across invocations of an adversary in a game. In particular, adversaries can access earlier assignments. For instance, the adversary’s second instantiation in game \( \text{Exp} \) has access to any assignments made during its first instantiation.

An adversary wins a game by causing it to output true (\( T \)) and the adversary’s success in a game \( \text{Exp}(\cdot) \), denoted \( \text{Succ}(\text{Exp}(\cdot)) \), is the probability that the adversary wins, that is, \( \text{Succ}(\text{Exp}(\cdot)) = \Pr[\text{Exp}(\cdot) = T] \). We focus on computational security, rather than information-theoretic security, and tolerate breaks by adversaries in non-polynomial time and breaks with negligible success, since such breaks are infeasible in practice. Game \( \text{Exp} \) captures a single interaction between the challenger and the adversary. We can extend games with oracles to capture arbitrarily many interactions. For instance, we can formulate a strengthening of \( \text{Exp} \) as follows: \( \beta \leftarrow_R \{0, 1\} \); \( g \leftarrow A^O(x) \); return \( g = \beta \), where \( A^O \) denotes \( A \)’s access to oracle \( O \) and \( O(m) \) computes if \( \beta = 0 \) then \( x \leftarrow H(m) \); else \( x \leftarrow S(m) \); return \( x \). Oracles may access game parameters such as \( \text{bit } \beta \).
each voter constructs and casts a ballot for their vote, and similarly for any re-votes. These ballots are collected and recorded on a bulletin board. Thirdly, the tallier tallies the collected ballots and announces the outcome as a frequency distribution of votes. The chosen representative is derived from this distribution, e.g., as the candidate with the most votes. Finally, voters and other interested parties check that the outcome corresponds to votes expressed in collected ballots.

**Definition 1 (Election scheme).** An election scheme is a tuple of probabilistic polynomial-time algorithms \((\text{Setup}, \text{Register}, \text{Vote}, \text{Tally}, \text{Verify})\) such that:

- **Setup**, denoted \((pk, sk, mb, mc) \leftarrow \text{Setup}(\kappa)\), is run by the tallier. The algorithm takes a security parameter \(\kappa\) as input and outputs a key pair \(pk, sk\), a maximum number of ballots \(mb\), and a maximum number of candidates \(mc\).

- **Register**, denoted \((pd, d) \leftarrow \text{Register}(pk, \kappa)\), is run by the registrar. The algorithm takes as input a public key \(pk\) and a security parameter \(\kappa\), and it outputs a credential pair \((pd, d)\), where \(pd\) is a public credential and \(d\) is a private credential.

- **Vote**, denoted \(b \leftarrow \text{Vote}(d, pk, v, cnt, nc, \kappa)\), is run by voters. The algorithm takes as input a private credential \(d\), a public key \(pk\), a voter’s vote \(v\), a counter \(cnt\), some number of candidates \(nc\), and a security parameter \(\kappa\). Vote \(v\) should be selected from a sequence \(1, \ldots, nc\) of candidates, and counter \(cnt\) should be incremented between a voter’s runs. (The counter might be a timestamp which increments with time or an integer that is manually incremented, for instance.) The algorithm outputs a ballot \(b\) or error symbol \(\bot\).

- **Tally**, denoted \((v, pf) \leftarrow \text{Tally}(sk, bb, nc, L, \kappa)\), is run by the tallier. The algorithm takes as input a private key \(sk\), a bulletin board \(bb\), some number of candidates \(nc\), an electoral roll \(L\), and a security parameter \(\kappa\), where \(bb\) is a set. The algorithm outputs an election outcome \(v\) and a non-interactive tallying proof \(pf\), where \(v\) is a vector of length \(nc\) and each index \(v\) of that vector should indicate the number of votes for candidate \(v\). Moreover, the tallying proof should demonstrate that the outcome corresponds to votes expressed in ballots on the bulletin board.

- **Verify**, denoted \(s \leftarrow \text{Verify}(pk, bb, nc, L, v, pf, \kappa)\), is run to audit an election. The algorithm takes as input a public key \(pk\), a bulletin board \(bb\), some

---

\(^3\)The syntax bounds the number of ballots \(mb\), respectively candidates \(mc\), to broaden the correctness definition’s scope (indeed, voting systems that encrypt votes typically require \(mc\) to be less than or equal to the size of the encryption scheme’s message space and schemes that homomorphically combine votes require \(mb\) to be less than or equal to the size of that space). The syntax represents votes as integers, rather than alphanumeric strings, for brevity. Finally, the syntax employs sets, rather than multisets or lists, to preclude the construction of schemes vulnerable to attacks that arise due to duplicate ballots [13, §2.1 & §4.3] (systems vulnerable to such attacks cannot be modelled using the syntax).
number of candidates \( n_c \), an electoral roll \( L \), an election outcome \( v \), a tallying proof \( pf \), and a security parameter \( \kappa \). The algorithm outputs a bit \( s \), which is 1 if the outcome should be accepted and 0 otherwise. We require the algorithm to be deterministic.

Election schemes must satisfy correctness: there exists a negligible function \( \text{negl} \), such that for all security parameters \( \kappa \), integers \( n_v \) and \( n_c \), vectors of votes \( v_1, \ldots, v_{n_v} \) over \( \{1, \ldots, n_c\} \), and vectors of counters \( c_1, \ldots, c_{n_v} \) such that \( \bigwedge_{1 \leq i \leq n_v} |v_i| = |c_i| \land c_i[1] < \cdots < c_i[|c_i|] \), it holds that, given a zero-filled vector \( v \) of length \( n_c \), we have:

\[
\Pr[(pk, sk, mb, mc) \leftarrow \text{Setup}(\kappa); \quad bb \leftarrow \emptyset; \quad \text{for } 1 \leq i \leq n_v \text{ do } (pd_i, d_i) \leftarrow \text{Register}(pk, \kappa); \quad \text{if } 0 < |v_i| \text{ then } \text{for } 1 \leq j \leq |v_i| \text{ do } b_j \leftarrow \text{Vote}(d_i, pk, v_i[j], c_i[j], n_c, \kappa); \quad bb \leftarrow bb \cup \{b_1, \ldots, b_{|v_i|}\}; \quad v[v_i[|v_i|]] \leftarrow v[v_i[|v_i|]] + 1; \quad (v', pf) \leftarrow \text{Tally}(sk, bb, nc, \{pd_1, \ldots, pd_{nb}\}, \kappa) : |bb| \leq mb \land nc \leq mc \Rightarrow v = v' > 1 - \text{negl}(\kappa).
\]

The syntax provides a language to model voting systems and the correctness condition ensures such systems function, i.e., election outcomes correspond to votes expressed in ballots (except for votes expressed in ballots from the same voter, which are all discarded, except for the voter’s last vote), when ballots are constructed and tallied in the prescribed manner. Athena will be defined in terms of this syntax, moreover, we will adopt definitions of verifiability and privacy expressed in the syntax and prove they are satisfied.

### 3 Our scheme: Athena

Our scheme (\( \text{Setup}, \text{Register}, \text{Vote}, \text{Tally}, \text{Verify} \)) is used as follows: The tallier initiates an election using algorithm \( \text{Setup} \) to compute a key pair, which includes a public key \( pk \) for an underlying multiplicative-homomorphic asymmetric encryption scheme (\( \text{Gen}, \text{Enc}, \text{Dec} \)), for which there exists a generator \( g \) of the scheme’s message space. Next, the registrar uses algorithm \( \text{Register} \) to compute a credential pair, wherein the private credential is a nonce \( d \) and the public credential is an encryption \( \text{Enc}(pk, g^d; r) \) of that nonce, for some coins \( r \). The registrar repeats the process to create further credential pairs and these pairs are issued to voters. Each voter uses algorithm \( \text{Vote} \) to compute their ballot, which includes: their public credential; an encryption \( \text{Enc}(pk, g^{-d}; s) \) of their negated private credential, for some coins \( s \); an encryption \( \text{Enc}(pk, v; t) \) of their vote \( v \), for some coins \( t \); and a counter \( cnt \). A voter similarly computes ballots.
for re-votes, using an incremented counter. It follows that the bulletin board will contain a ballot for a voter’s first vote

\[ \text{Enc}(pk, g^d; r) \quad \text{Enc}(pk, g^{-d}; s_1) \quad \text{Enc}(pk, v_1; t_1) \quad \text{cnt}_1, \]

ballots for any of the voter’s re-votes

\[ \text{Enc}(pk, g^d; r) \quad \text{Enc}(pk, g^{-d}; s_2) \quad \text{Enc}(pk, v_2; t_2) \quad \text{cnt}_2 \]

\[ \vdots \]

\[ \text{Enc}(pk, g^d; r) \quad \text{Enc}(pk, g^{-d}; s_k) \quad \text{Enc}(pk, v_k; t_k) \quad \text{cnt}_k, \]

such that \( \text{cnt}_1 < \cdots < \text{cnt}_k \), and any other ballots cast using the voter’s public credential, without the private credential (including those cast by the adversary or even the voter themselves), namely,

\[ \text{Enc}(pk, g^d; r) \quad \text{Enc}(pk, g^D_1; \pi_1) \quad \text{Enc}(pk, \pi_1; \overline{t}_1) \quad \overline{\text{cnt}}_1 \]

\[ \vdots \]

\[ \text{Enc}(pk, g^d; r) \quad \text{Enc}(pk, g^D_l; \pi_l) \quad \text{Enc}(pk, \pi_l; \overline{t}_l) \quad \overline{\text{cnt}}_l. \]

(Ballots also prove correct ciphertext construction, moreover, they prove that the second ciphertext of each ballot encrypts a message of the form \( g^m \). Hence, we restrict ourselves to well-defined ballots above.) Furthermore, the bulletin board will contain ballots cast using other public credentials.

The tallier uses algorithm \text{Tally} to compute the election outcome as follows: The tallier generates a nonce \( n \), homomorphically combines the second ciphertext of each entry on the bulletin board with itself \( n \)-times, decrypts the resulting homomorphic combinations to reveal anonymised credentials,\(^4\), and prepends entries with anonymised credentials, thereby producing output including

\[ g^{-d \cdot n} \quad \text{Enc}(pk, g^d; r) \quad \text{Enc}(pk, g^{-d}; s_1) \quad \text{Enc}(pk, v_1; t_1) \quad \text{cnt}_1 \]

\[ \vdots \]

\[ g^{-d \cdot n} \quad \text{Enc}(pk, g^d; r) \quad \text{Enc}(pk, g^{-d}; s_k) \quad \text{Enc}(pk, v_k; t_k) \quad \text{cnt}_k \]

\[ g^{D_1 \cdot n} \quad \text{Enc}(pk, g^d; r) \quad \text{Enc}(pk, g^{D_1}; \pi_1) \quad \text{Enc}(pk, \pi_1; \overline{t}_1) \quad \overline{\text{cnt}}_1 \]

\[ \vdots \]

\[ g^{D_1 \cdot n} \quad \text{Enc}(pk, g^d; r) \quad \text{Enc}(pk, g^{D_l}; \pi_l) \quad \text{Enc}(pk, \pi_l; \overline{t}_l) \quad \overline{\text{cnt}}_l. \]

Entries with the same public credential that are prepended with the same value are discarded, except for the one with the highest counter. Hence, the first \( k - 1 \) entries (above) are discarded, whilst the \( k \)th entry is preserved. The remaining

\(^4\)The second ciphertext could simply be decrypted (without harming unforgeability, since knowledge of the exponent must be demonstrated), but this permits some linkability between elections, which is undesirable
entries are similarly processed, therefore, the last will be kept if any entries sharing the prepended value \( (g^{D_l.n}) \) have counter values lower than counter \( c^{n_l} \). (For example, suppose only the penultimate entry shares prepended value \( g^{D_l.n} \), i.e., \( D_l = D_{l-1} \), and further suppose \( c^{n_l} > c^{n_{l-1}} \). Hence, the last entry will be preserved and the penultimate entry will be discarded. By comparison, the penultimate entry will be kept if \( c^{n_{l-1}} < c^{n_l} \).) The first two ciphertexts of preserved entries are homomorphically combined and paired with encrypted votes, producing

\[
\text{Enc}(pk, g^d \odot g^{-d}; r \oplus s_k) \quad \text{Enc}(pk, v_k; t_k),
\]

for the \( k \)th entry (above), and

\[
\text{Enc}(pk, g^d \odot g^{D_l}; r \oplus s_l) \quad \text{Enc}(pk, \tau_l; \tilde{t}_l),
\]

for the last (assuming it is preserved). The homomorphic combinations and encrypted votes are mixed (using the same permutation). The tallier performs (optimised) plaintext equality tests on each of the mixed homomorphic combinations to determine whether they contain plaintext one,

\[5\]

and decrypts the corresponding mixed encrypted votes when the test holds. Thus, the voter’s vote \( v_k \) (above) is revealed, because mixed ciphertext \( \text{Enc}(pk, g^d \odot g^{-d}; r \oplus s_k \oplus w) \) encrypts 1, whereas vote \( \tau_l \) is not revealed, because mixed ciphertext \( \text{Enc}(pk, g^d \odot g^{D_l}; r \oplus s_l \oplus \overline{w}) \) does not (recall \( g^{D_l} \) was constructed without private credential \( d \)), where \( w \) and \( \overline{w} \) are coins introduced during mixing. The election outcome is the tally of revealed votes.

Athena is formally specified by Definition 2, using cryptographic primitives introduced in Appendix A. Those primitives include sigma protocols for proving correct key generation, ciphertext construction, and decryption, which behave as one might expect. They also include a sigma protocol for proving iterative homomorphic combination, that is, proving that a ciphertext \( c \) is computed from another ciphertext \( c' \) such that \( c = \bigotimes^n c' \), for some nonce \( n \). We apply the Fiat-Shamir transformation to sigma protocols to derive non-interactive proof systems, which we use to achieve verifiability.

**Definition 2** (Athena). Suppose \( \Pi = (\text{Gen, Enc, Dec}) \) is a multiplicative-homomorphic asymmetric encryption scheme with a message space that is super-polynomial in the security parameter and for which a generator exists; \( \mathcal{M} \) is a verifiable pairwise mixnet; \( \Sigma_1, \Sigma_2, \Sigma_3 \) and \( \Sigma_4 \) are sigma protocols that prove key generation, ciphertext construction, decryption and iterative homomorphic combination, respectively; and \( \mathcal{H} \) is a hash function. Let \( \text{FS}(\Sigma_1, \mathcal{H}) = (\text{ProveKey, VerKey}) \), \( \text{FS}(\Sigma_2, \mathcal{H}) = (\text{ProveCiph, VerCiph}) \), \( \text{FS}(\Sigma_3, \mathcal{H}) = (\text{ProveDec, VerDec}) \), and \( \text{FS}(\Sigma_4, \mathcal{H}) = (\text{ProveComb, VerComb}) \). Election scheme Athena, denoted \( \text{Athena}(\Pi, \mathcal{M}, \Sigma_1, \Sigma_2, \Sigma_3, \Sigma_4, \mathcal{H}) \) = (Setup, Register, Vote, Tally, Verify), is defined by the following algorithms.

---

\[5\] Mixed homomorphic combinations could simply be decrypted (without harming unforgeability), but this permits some linkability between elections.
• Setup(κ). Compute

\[(pk, sk, m) \leftarrow Gen(\kappa);\]
\[\rho \leftarrow \text{ProveKey}((\kappa, pk, m), (sk, r, \kappa));\]
\[pk \leftarrow (pk, \rho);\]
\[sk \leftarrow (pk, sk),\]

let \(mb\) be the largest integer upper-bounded by a polynomial in the security parameter, let \(mc\) be the largest integer such that \(\{0, \ldots, mc\} \subseteq \{0\} \cup \{m\}\) and \(mc\) is upper-bounded by a polynomial in the security parameter, and output \((pk, sk, mb, mc)\).

• Register(\(pk, k\)). Parse \(pk\) as \((pk, m, \rho)\), outputting \((\bot, \bot)\) if parsing fails or \(\text{VerKey}((\kappa, pk, m), \rho, \kappa) = \bot\), generate nonce \(d\), compute

\[pd \leftarrow Enc(pk, g^d);\]
\[d \leftarrow (pd, d),\]

and output \((pd, d)\), where \(g\) is a generator of message space \(m\).

• Vote(\(d, pk, v, cnt, nc, \kappa\)). Parse \(d\) as a vector \((pd, d)\) and \(pk\) as a vector \((pk, m, \rho)\), outputting \(\bot\) if parsing fails or \(\text{VerKey}((\kappa, pk, m), \rho, \kappa) = \bot \lor v \not\in \{1, \ldots, nc\} \lor \{1, \ldots, nc\} \not\subseteq m\), compute

\[c_1 \leftarrow Enc(pk, g^{-d});\]
\[c_2 \leftarrow Enc(pk, v);\]
\[\sigma_1 \leftarrow \text{ProveCiph}((pk, g, c_1, m), (d, s), m, \kappa);\]
\[\sigma_2 \leftarrow \text{ProveCiph}((pk, c_2, \{1, \ldots, nc\}), (v, t), m, \kappa),\]

and output \((pd, c_1, c_2, \sigma_1, \sigma_2, cnt)\), where message \(m = (pd, c_1, c_2, \text{cnt})\) and \(g\) is the aforementioned generator of message space \(m\).

• Tally(\(sk, bb, nc, L, \kappa\)). Parse \(sk\) as vector \((pk, sk)\), initialise \(v\) as a zero-filled vector of length \(nc\), and proceed as follows.

1. Remove invalid ballots: Let \(\{b_1, \ldots, b_\ell\}\) be the largest subset of senary vectors in \(bb\) such that \(b_1[1] \leq \cdots \leq b_\ell[1]\) and for each \((pd, c_1, c_2, \sigma_1, \sigma_2, cnt)\) in the subset we have \(pd \in L \land \text{VerCiph}((pk, g, c_1, m), \sigma_1, m, \kappa) \land \text{VerCiph}((pk, c_2, \{1, \ldots, nc\}), \sigma_2, m, \kappa)\), where \(g\) is again the aforementioned generator of message space \(m\) and message \(m = (pd, c_1, c_2, \text{cnt})\). If the subset is empty, then output \((v, \bot)\).

2. Mix final votes: Initialise \(\text{pfr}\) as an empty vector and \(A\) as an empty map from pairs (comprising a ciphertext and a group element) to triples (comprising a counter and two ciphertexts), generate nonce \(n\), compute

\[
\text{for } 1 \leq i \leq \ell \text{ do }
\begin{align*}
&c_i' \leftarrow \bigotimes_{j=1}^{\ell} b_i[2]; \\
&N \leftarrow \text{Dec}(sk, c_i'); \\
&t \leftarrow A[(b_i, N)];
\end{align*}
\]

9
if t = null ∨ t[1] < b[6] then
  // Update the map if A[(b[1], N)] is empty
  // or contains a lower counter
  A[(b[1], N)] ← (b[6], b[1] ⊗ b[2], b[3]);
else if t[1] = b[6] then
  // Disregard duplicate counters
  A[(b[1], N)] ← (b[6], ⊥, ⊥);
ς ← ProveDec((pk, c′, N, sk, κ);
if |pfr| > 0 then
  // Prove c′ − 1 and c′ are derived by iterative
  // homomorphic combination wrt nonce n
  ω ← ProveComb((pk, (c′, (b − 1)[2], b[2])), n, κ);
  pfr ← pfr || (c′, N, ω);
else
  pfr ← pfr || (c′, N, ς);
and apply (pairwise) mixnet M to the pairs of ciphertexts in map A to derive vector B.

3. Reveal eligible votes: Initialise pfd as an empty vector, generate nonces n1, ..., n|B|, and compute

for (c1, c2) ∈ B do
  c′ ← ⊗i=n[pfd]c1;
  m ← Dec(sk, c′);
  ω ← ProveComb((pk, c′, n|pfd|+1, κ);
  ς1 ← ProveDec((pk, c′, m, sk, κ);
  if m = 1 then
    // c1 encrypts g0, hence, is derived from homo
    // comb of pub cred and enc of neg priv cred
    v ← Dec(sk, c2);
    v[v] ← v[v] + 1;
    ς2 ← ProveDec((pk, c2, v), sk, κ);
    pfd ← pfd || (c′, v, ω, ς1, ς2);
  else
    pfd ← pfd || (c′, m, ω, ς1);
and output (v, (pfr, B, pfd)), where g is the aforementioned generator of message space m.

• Verify(pk, bb, nc, L, v, pf, κ). Parse pk as vector (pk, m, ρ) and v as a vector of length nc, outputting 0 if parsing fails, VerKey((κ, pk, m), ρ, κ) = ⊥, or nc ≤ mc, where mc is computed as per algorithm Setup. Perform the following checks.
Lemma 1. \(\text{Athena}(\Pi, M, \Sigma_1, \Sigma_2, \Sigma_3, \Sigma_4, \mathcal{H})\) is an election scheme when cryptographic primitives satisfy the preconditions of Definition 2 and asymmetric encryption scheme \(\Pi\) is perfectly correct.

A proof of Lemma 1 appears in Appendix B. Beyond correctness, verifiable election schemes should satisfy completeness, i.e., auditing should succeed for evidence produced by tallying, hence, algorithm \(\text{Verify}\) should accept outputs of algorithm \(\text{Tally}\). We prove \(\text{Athena}\) satisfies completeness in Section 5.1.

\(\text{Athena}\) should be instantiated with an asymmetric encryption scheme satisfying IND-CPA and sigma protocols satisfying special soundness and special honest verifier zero-knowledge. This ensures the non-interactive proof systems derived by application of the Fiat-Shamir transformation satisfy zero-knowledge and simulation sound extractability [12], which help achieve both privacy and verifiability. Moreover, this ensures that ballots are non-malleable [12], which is necessary for privacy [58]. Furthermore, for linear complexity, we require computation \(\bigotimes_i^c\) to be linear in the length of \(c\), which is possible for El Gamal, for instance. (Indeed, we have \(\bigotimes_i^n (g^r, (g^r)^r \cdot M) \equiv (g^r, (g^r)^r \cdot M)^n \equiv (g^r, (g^r)^r \cdot M^n)\).

1. Check ballot removal. Compute \(\{b_1, \ldots, b_t\}\) as per Step 1 of algorithm \(\text{Tally}\) and check \(\{b_1, \ldots, b_t\} = \emptyset\) implies \(v\) is a zero-filled vector.

2. Check mix. Check \(\text{pf}\) parses as a vector \((\text{pfr}, B, \text{pfd})\) and \(\text{pfr}\) parses as a vector \(((c_{11}, N_1, s_1), (c_2, N_2, s_2, \omega_2), \ldots, (c_{t1}, N_t, s_t, \omega_t))\) such that \(\bigwedge_{1 \leq i \leq t} \text{VerDec}((pk, c_i, N_i, s_i, \omega_i)\) and \(\bigwedge_{1 \leq i \leq t} \text{VerComb}((pk, (c_{i-1}', c_i'), (b_{i-1}[2], b_i[2])), \omega_i, \kappa)\), initialise \(A\) as an empty map from pairs to triples, compute

\[
\text{for } 1 \leq i \leq t \text{ do}
\]

\[
\begin{align*}
t & \leftarrow A[(b_i[1], N_i)]; \\
\text{if } t = \text{null} \lor t[1] \leq b_i[6] \text{ then} \\
& [A[(b_i[1], N_i)] \leftarrow (b_i[6], b_i[1] \otimes b_i[2], b_i[3]), \\
\text{else if } t[1] = b_i[6] \text{ then} \\
& [A[(b_i[1], N_i)] \leftarrow (b_i[6], \bot, \bot);
\end{align*}
\]

and check \(B\) was output by the mix applied in Step 2 of algorithm \(\text{Tally}\) on input of the pairs of ciphertexts in \(A\).

3. Check revelation. Checks \(\text{pfd}\) parses as a vector of length \(|B|\) such that for all \(v \in \{1, \ldots, |N|\}\) we have

\[
\exists v \in \{1, \ldots, |B|\} : \exists c_1, c_2, c', \omega, s_1, s_2 : (c_1, c_2) = B[i] \land (c', \omega, s_1, s_2) = \text{pfd}[i] \land \text{VerDec}((pk, c', 1), s_1, \kappa) \land \text{VerDec}((pk, c', 2), v, s_2, \kappa),
\]

and for each remaining integer \(i \in \{1, \ldots, |B|\}\) we have \(B[i]\) parses as \((c_1, c_2)\), \(\text{pfd}[i]\) parses as \((c', m, \omega, s_1)\), and \(\text{VerComb}((pk, c', c_1), \omega, \kappa) \land \text{VerDec}((pk, c', m), s_1, \kappa) \land m \neq 1\).

Output 1 if all the above checks hold.
Implementation. Athena is formally stated independently of the underlying cryptographic primitives (for generality, algorithm agility, and ease of proofs). In practice, Athena can be instantiated with established cryptographic primitives. For instance, we might instantiate asymmetric encryption scheme $\Pi$ as El Gamal \cite{26} and we might instantiate sigma protocols as follows: $\Sigma_1$ as the protocol for proving knowledge of discrete logarithms by Chaum et al. \cite[Protocol 2]{15}, $\Sigma_2$ as the protocol for proving knowledge of disjunctive equality between discrete logarithms by Cramer et al. \cite[Figure 1]{24}, $\Sigma_3$ as the protocol for proving knowledge of equality between discrete logarithms by Chaum & Pedersen \cite[§3.2]{16}, and $\Sigma_4$ as a slight variant of the protocol by Chaum & Pedersen.\footnote{To prove iterative homomorphic combination using equality between discrete logarithms, witness that $\bigwedge_{1 \leq i \leq n} (\alpha_i, \beta_i) = (\alpha_i', \beta_i')^n$ iff $\bigwedge_{1 \leq i \leq n} \log_{\alpha_i} \alpha_i \equiv \log_{\beta_i} \beta_i$ and $\bigwedge_{1 \leq i \leq n} \log_{\alpha_i - 1} \alpha_i - 1 \equiv \log_{\alpha_i} \alpha_i$, where $(\alpha_1, \beta_1), (\alpha_1', \beta_1'), \ldots, (\alpha_n, \beta_n), (\alpha_n', \beta_n')$ are El Gamal ciphertexts.}

4 Privacy results

An Athena ballot contains a public credential, i.e., an encryption of the corresponding private credential, and an encryption of the negated private credential. Yet, no voter can prove that any ballot contains their private credential. Indeed, a well-formed Athena ballot that encrypts the negation of a voter’s private credential is indistinguishable from an ill-formed ballot that encrypts some other value, rather than such a negation. Hence, during the voting phase, a voter cannot prove whether they cast a well-formed ballot (that will be counted, as opposed to an ill-formed ballot that will not), let alone prove how they voted, thereby assuring coercion-resistance during the voting phase.

Associating each public credential with anonymised credentials (to discard early votes prior to mixing) reveals the number of ballots whose second ciphertext contains the same plaintext (be that a private credential or some other value). For instance, a voter that casts a specific number of ballots containing such a plaintext can check to see whether an anonymised credential appears the specified number of times (in association with the voter’s public credential). But, no voter can prove that those ballots are well-formed. Indeed, the voter may cast the expected number of ballots using a nonce in place of their private credential’s negation, which will result in the expected relation, yet the ballots are ill-formed and will not be counted. Hence, coercion-resistance is preserved before mixing.

Finally, homomorphically combining ciphertexts, mixing those combinations and encrypted votes, and using plaintext equality tests to determine voters’ votes (as opposed to adversarial votes) preserves coercion-resistance, as does decrypting mixed (voters’) votes. (Ballots prove that votes are selected from the sequence of candidates, which provides protection against randomisation attacks \cite[§1.1]{37}.) It follows that tallying preserves coercion-resistance. Thus,
Athena is a coercion-resistant voting system. A formal proof is deferred to later work.

**Distributed tallying.** Coercion-resistance does not provide assurances when deviations from the prescribed tallying procedure are possible. Indeed, such deviations include revealing the tallier’s private key, which undermines privacy. Hence, the tallier must be trusted. Alternatively, we can design voting systems that distribute the tallier’s role amongst several talliers and ensure free-choice assuming at least one tallier behaves. Extending Athena in this direction is straightforward, since distributed variants of the underlying primitives are well-known. Ultimately, we would prefer not to trust talliers; unfortunately, this is only known to be possible for decentralised voting systems, e.g., [29, 32, 39–41, 54], which do not scale.

5 Verifiability results

Athena records ballots on a (public) bulletin board, hence, voters can check whether their ballot is collected (individual verifiability). Moreover, tallying produces evidence demonstrating that the announced election outcome corresponds to the votes expressed in collected ballots that are authorised (universal verifiability). Furthermore, only voters can construct authorised ballots (unforgeability). It follows that Athena is a verifiable election scheme, as we shall prove below using formal definitions from Smyth, Frink & Clarkson [61] that we extend to include re-voting (Appendix C).

5.1 Universal verifiability

Universal verifiability asserts that anyone must be able to check whether an election outcome corresponds to votes expressed in collected ballots that are authorised. Since checks can be performed by algorithm Verify, it suffices that the algorithm accept if and only if the outcome corresponds to votes expressed in collected ballots that are authorised. The only if requirement is formalised by **Soundness** (Definition 12), which requires algorithm Verify to only accept correct outcomes, and the if requirement is captured by **Completeness** (Definition 13), which requires election outcomes produced by algorithm Tally to be accepted by algorithm Verify.

**Proposition 2** (Soundness). Election scheme Athena(Π, M, Σ₁, Σ₂, Σ₃, Σ₄, H) satisfies **Soundness**, when asymmetric encryption scheme Π is perfectly correct, mixnet M is verifiable, sigma protocols Σ₁, Σ₂, Σ₃ and Σ₄ satisfy special soundness and special honest verifier zero-knowledge, and H is a random oracle, assuming Injectivity is satisfied.

We defer consideration of Injectivity (Definition 11) to Section 5.3.
Proof sketch of Proposition 2. We must establish that outcomes accepted by algorithm Verify correspond to votes expressed in collected ballots that are authorised. Step 1 of the algorithm ensures accepted outcomes are only influenced by bulletin board entries constructed by algorithm Vote, i.e., only (valid) ballots have influence (invalid ballots do not), and only when they contain a public credential. Step 2 ensures no influence from any mixed ballots that share a public credential (and a anonymised credential) with another mixed ballot, whilst being associated with a (strictly) lower counter value. Moreover, pairs of mixed ballots that share a public credential (and a anonymised credential) and a counter are ensured to have no influence either. It follows that only mixed ballots expressing the last vote associated with a public and an anonymised credential may have influence. Finally, Step 3 restricts influence to mixed ballots associated with a voter’s public and private credential, i.e., only voters’ last votes have influence, hence, accepted outcomes correspond to votes expressed in authorised collected ballots.

A detailed proof of Proposition 2 and all other verifiability proofs appear in Appendix D.

Proposition 3 (Completeness). Election scheme Athena(Π, M, Σ₁, Σ₂, Σ₃, Σ₄, H) satisfies Completeness when mixnet M is verifiable, sigma protocol Σ₂ satisfies special soundness and special honest verifier zero-knowledge, and H is a random oracle.

Proof sketch. We must establish that outcomes produced by algorithm Tally are accepted by algorithm Verify. It is trivial to see that an outcome output by the first step of algorithm Tally will be accepted by the first step of algorithm Verify, and it remains to consider outcomes output by the last step of algorithm Tally. Simulation sound extractability (of sigma protocol Σ₂) assures us that such outcomes are derived from ballots containing well-formed ciphertexts. It is straightforward to see that computations performed by the second step of algorithm Tally can be successfully checked in the second step of algorithm Verify, in particular, proofs can be verified, because proof systems are complete. Moreover, since map A is equivalently computed (from well-formed ciphertexts) by both algorithms and since the mixnet is verifiable, it follows that checks performed on the mixnet’s output succeed. Finally, the checks performed by the third step of algorithm Verify succeed, because proof systems are complete, thus, outcomes produced by algorithm Tally are accepted.

5.2 Unforgeability

Unforgeability asserts that only voters can construct authorised ballots. Since ballots are authenticated by private credentials, it suffices to ensure that knowledge of a private credential is necessary to construct an authentic ballot, which is formalised by Unforgeability (Definition 14). We defer a formal proof to later work.
**Comparison with the voting system by Juels, Catalano & Jakobsson.** Smyth, Frink & Clarkson [61, §6] show that the voting system by Juels, Catalano & Jakobsson only achieves unforgeability assuming the tallier is honest, because the tallier’s private key can be used to discover private credentials (by decrypting public credentials), which enables adversarial construction of authorised ballots. By comparison, Athena achieves unforgeability even if the tallier is dishonest, since the tallier’s private key can only be used to recover $g^d$ or $g^{-d}$, neither of which can be used to construct an authorised ballot, because ballots must prove knowledge of private credential $d$. Thus, Athena improves upon the security of the voting system by Juels, Catalano & Jakobsson. (Their voting system can probably be improved using a similar idea.)

**5.3 Individual verifiability**

Individual verifiability asserts that voters must be able to check whether their ballot is amongst those collected. Since ballots should be collected and recorded on a bulletin board, and since the board must be available to everyone, it suffices for voters to check that their ballot (i.e., the ballot they constructed) is on the bulletin board. Hence, it is necessary for voters to check that their ballot has not been omitted from the bulletin board. Yet, this is insufficient, because the presence of a ballot identical to a voter’s ballot does not imply the presence of the ballot constructed by the voter. Indeed, such a ballot might have been constructed by another voter. Thus, individual verifiability requires that voters must be able to uniquely identify their ballot, i.e., ballots do not collide, which is formalised by **Individual-Verifiability** (Definition 15).

To ensure Athena satisfies individual verifiability, it suffices to require that the underlying encryption scheme produces distinct ciphertexts with overwhelming probability. Smyth explains that “[s]ecurity properties of asymmetric encryption schemes ensure [distinct] ciphertexts...But, such security properties assume public keys are generated (by key generation algorithms) using coins chosen uniformly at random. By comparison, individual verifiability and injectivity assume public keys are constructed by the adversary. Thus, security properties are insufficient to ensure...individual verifiability and injectivity” [60]. Nonetheless, given that Athena checks correct key generation, it suffices that ciphertexts are distinct for correctly generated keys.

**Proposition 4 (Individual-Verifiability).** Election scheme Athena($\Pi, M, \Sigma_1, \Sigma_2, \Sigma_3, \Sigma_4, H$) satisfies Individual-Verifiability if for all probabilistic polynomial-time adversaries $A$ and security parameters $\kappa$ we have $\Pr[(pk, m, p, m') \leftarrow A(\kappa); c \leftarrow \text{Enc}(pk, m); c' \leftarrow \text{Enc}(pk, m') : \text{VerKey}((\kappa, pk, m), p, \kappa) = 1 \land m, m' \in m \Rightarrow c \neq c'] > 1 - \text{negl}(\kappa)$, where $\Pi = (\text{Gen, Enc, Dec})$ and $\text{FS}(\Sigma, H) = (\text{ProveKey, VerKey})$. Moreover, Injectivity is satisfied if the probability is 1 when plaintexts $m$ and $m'$ are distinct.

The preconditions used by Proposition 4 are due to Smyth [60, §3], and our proof is structurally similar to his proof of individual verifiability and injectivity for a class of encryption-based voting systems.
6 Complexity analysis

Analysing the complexity of algorithms Setup, Register, and Vote is straightforward (given the simplicity of those algorithm): Setup generates a key pair and proof of correct generation, Register computes a ciphertext, and Vote computes two ciphertexts along with proofs of correct construction. Hence, complexity of the registration phase is linear in the number of voters and complexity of the voting phase is linear in the number of ballots cast. Algorithms Tally and Verify are more elaborate and analysis is more involved. We proceed by a detailed inspection of each algorithm and find that complexity remains linear.

Tally. We consider each step of algorithm Tally: It is trivial to see that complexity is upper-bound by the bulletin board’s length in Step 1. For Step 2, we have assumed computation $\bigotimes_{i=1}^{c} c$ is linear in the length of $c$ (§3), hence, it is straightforward to see that complexity is upper-bound by the number of for-loop iterations, which is constrained by the number of valid ballots on the bulletin board, therefore, complexity is again upper-bound by the bulletin board’s length, because the number of valid ballots is at most the number of ballots on the bulletin board. Complexity of Step 3 is similarly upper-bound by the number of for-loop iterations, which is constrained by the number of pairs output by the mix and at most the number of ballots on the bulletin board, therefore, complexity is upper-bound by the bulletin board’s length.

Verify. We consider the steps of algorithm Verify: Complexity of Step 1 is trivially linear in the bulletin board’s length; Step 2 is straightforwardly linear in the number of for-loop iterations, which is linear in the bulletin board’s length; and Step 3 is straightforwardly linear in the number pairs output by the mix, which is again linear in the bulletin board’s length.

Thus, Athena has linear complexity $O(|\text{bb}|)$ in the length of the bulletin board ($\text{bb}$), assuming linear complexity of iterative homomorphic combinations (§3), which is possible for El Gamal, for instance.

Comparison with the voting system by Juels, Catalano & Jakobsson. Complexity of the voting system by Juels, Catalano & Jakobsson is quadratic, due to pairwise plaintext equality tests performed on ballots to ensure that only the last choice of each voter has influence, which is needed for universal verifiability. By comparison, Athena uses anonymised credentials in a manner that achieves the same property, whilst reducing complexity. (The voting system by Juels, Catalano & Jakobsson also performs pairwise plaintext equality tests on mixed ballots and mixed credentials, to identify authorised ballots. The complexity of those tests is upper-bound by $O(|L| \cdot |\text{bb}|)$, where $L$ is the electoral roll and $|\text{bb}|$ is the bulletin board’s length. By comparison, the plaintext equality tests performed by Athena are upper-bound by $O(|\text{bb}|)$.)
7 General design principles

General design principles were identified and embraced during the development of Athena. This section shares these principles to aid the development of future voting systems, especially those with linear complexity. Our first design principle is guided by definitions of correctness and universal verifiability:

1. Algorithm Tally must map votes expressed in authorised ballots to the outcome corresponding to those votes, except for any early votes.

It follows that:

2. Algorithm Vote must authenticate ballots.

The next two design principles follow from our informal definitions of ballot secrecy and coercion resistance, respectively:

3. Algorithm Vote must ensure votes cannot be revealed from ballots; and

4. Algorithm Tally must not reveal any (meaningful) mapping between ballots and the outcome.

Forgoing coercion resistance (in favour of ballot secrecy), the previous design principle can be generalised: Algorithm Tally must not reveal any (meaningful) mapping between authorised ballots and the outcome. But this permits revealing authorised ballots, which allows simulation attacks [37, §1.1], whereby a coercer instructs a voter to reveal their private credential, uses that private credential to cast a ballot, and determines whether the voter followed instruction by checking whether the cast ballot is authorised. By comparison, (4) gives way to the following design principle:

5. Algorithm Tally must not reveal authorised ballots.

Similarly, authorised ballots must not be revealed during casting and collection:

6. Algorithm Vote must not reveal authorised ballots.

Since ballots must be authenticated (1) without revealing authorised ballots (5 & 6), the following principle emerges:

7. Algorithm Tally should anonymise ballots prior to authentication.

Given that ballots should be anonymised (7) and that bulletin boards may contain more than just ballots, it is proposed that:

8. Algorithm Vote should prove correct ballot construction; and

9. Algorithm Tally should discard garbage, i.e., non-ballots.
Figure 1 General design principles: Algorithm Tally should filter the bulletin board to discard garbage, early votes, and unauthorised (anonymised) ballots, the algorithm should also map any authenticated (anonymous) ballots to the outcome.

Revealing re-votes after anonymisation can be problematic, for instance, a voter that casts a specific number of ballots can deanonymise their anonymised ballots. Thus, our final design principle is suggested:

10. Algorithm Tally should discard ballots representing early votes prior to anonymisation.

For compatibility between (7 & 10) and (4), our notion of meaningful should exclude garbage and early votes, i.e., algorithm Tally is permitted to reveal mappings between the outcome and garbage, ballots representing early votes, or both.

By combining our design principles, we observe that algorithm Tally should filter the bulletin board to remove garbage (9) and ballots representing early votes (10). Moreover, after anonymising any remaining ballots, the algorithm should authenticate anonymised ballots and remove any unauthorised anonymous ballots (7). Finally, votes expressed in any authenticated anonymous ballots should be mapped to the outcome corresponding to those votes (1). This procedure is illustrated in Figure 1 and underpins the design of Athena.

8 Related work

Acquisti [1], Smith [57], and Weber, Araújo & Buchmann [65] reduce complexity to linear in variants of the voting system by Juels, Catalano & Jakobsson, but those reductions led to the lose of coercion-resistance [3–6, 21]. Araújo et al. [3–5] make better progress, albeit, without supporting audits for statistically determining whether non-voters are issued with credentials [6, 53, 62] and without supporting reuse of credentials between elections [2, 6]. Haghighat, Dousti
& Jalili do not permit reuse either [31], whereas Araújo et al. do [2, 6], albeit, Araújo et al. do not achieve strong non-reusability (i.e., only the last choice of each voter has influence [46]) nor universal verifiability, in the presence of an adversary that can re-order ballots (e.g., a network adversary), because they are reliant on ballot order to discard early votes.\(^7\)\(^8\) (The voting system by Juels, Catalano & Jakobsson does not satisfy strong non-reusability nor universal verifiability against such an adversary either, whereas Civitas does [61, §4.2.2].) Schläpfer et al. and Spycher et al. also make progress, albeit, Schläpfer et al. only achieve linear complexity for a trade in the degree of coercion-resistance and they leak the number of ballots each voter casts [53] and Spycher et al. make an additional trust assumption, namely, they assume the teller introduces a secret number of dummy votes for each voter (without any means for voters to confirm they did) [62]. Beyond variants of the voting system by Juels, Catalano & Jakobsson, distinct voting systems have also been introduced: The system by Schweisgut [56] achieves linear-complexity, but fails to achieve coercion-resistance [5]. Clark & Hengartner propose the Selections voting system, which makes better progress, albeit, some degree of coercion-resistance is traded to achieve linear complexity and the number of ballots each voter casts is leaked [18, 19]. (Athena leaks the number of ballots cast in association with each public credential, but not the number of ballots each voter casts.) Finally, Essex, Clark & Hengartner propose the Cobra voting system, which achieves remarkably fast tallying, albeit, registration has quadratic complexity in the number of voters [27]. With the exception of Juels, Catalano & Jakobsson, Haghighat, Dousti & Jalili, and Clark & Hengartner, none of these prior works present security proofs and proving their security remains an open problem. (Araújo et al. [2, 5] formally state theorems, but defer proofs to full versions of their papers, which do not appear to be public.)

9 Conclusion

For one and a half decades, researchers have strived to improve upon seminal work by Juels, Catalano & Jakobsson. This work attempts to deliver such an improvement: A verifiable, coercion-resistant voting system with linear complexity in the length of the bulletin board. We have seen how several of the ideas can help improve security of existing voting systems. Moreover, they generalise beyond voting to other systems that require strong forms of privacy, authentication, and verifiability, thereby advancing not just voting technology, but the science of security.

---

\(^7\)Araújo et al. can probably achieve strong non-reusability and universal verifiability using counters, as per Athena.

\(^8\)Araújo et al. do not specify sufficient details to ensure non-malleable ballots, without which unforgeability is not satisfied, because authentication material can be replayed. Similarly, encrypted votes can be replayed to violate coercion-resistance (cf. [8, 23, 47, 52, 66]). Nonetheless, including the entire ballot (except proofs) in the hashes used by proofs should suffice to ensure non-malleability.
A Cryptographic primitives

A.1 Asymmetric encryption

Definition 3 (Asymmetric encryption scheme [38]). An asymmetric encryption scheme is a tuple of probabilistic polynomial-time algorithms \((\text{Gen}, \text{Enc}, \text{Dec})\), such that.

- \(\text{Gen}\), denoted \((pk, sk, m) \leftarrow \text{Gen}(\kappa)\), inputs a security parameter \(\kappa\) and outputs a key pair \((pk, sk)\) and message space \(m\).
- \(\text{Enc}\), denoted \(c \leftarrow \text{Enc}(pk, m)\), inputs a public key \(pk\) and message \(m \in m\), and outputs a ciphertext \(c\).
- \(\text{Dec}\), denoted \(m \leftarrow \text{Dec}(sk, c)\), inputs a private key \(sk\) and ciphertext \(c\), and outputs a message \(m\) or an error symbol. We assume \(\text{Dec}\) is deterministic.

Moreover, the scheme must be correct: there exists a negligible function \(\text{negl}\), such that for all security parameters \(\kappa\) and messages \(m\), we have \(\Pr[(pk, sk, m) \leftarrow \text{Gen}(\kappa); c \leftarrow \text{Enc}(pk, m); m' = \text{Dec}(sk, c)] > 1 - \text{negl}(\kappa)\). A scheme has perfect correctness if the probability is 1.

Definition 4 (Homomorphic encryption [61]). An asymmetric encryption scheme \(\Pi = (\text{Gen}, \text{Enc}, \text{Dec})\) is homomorphic, with respect to ternary operators \(\odot\), \(\oplus\), and \(\otimes\), if there exists a negligible function \(\text{negl}\), such that for all security parameters \(\kappa\), we have the following.\(^\ast\)

1. First, for all messages \(m_1\) and \(m_2\) we have \(\Pr[(pk, sk, m) \leftarrow \text{Gen}(\kappa); c_1 \leftarrow \text{Enc}(pk, m_1); c_2 \leftarrow \text{Enc}(pk, m_2); m_1, m_2 \in m \Rightarrow \text{Dec}(sk, c_1 \odot pk c_2) = \text{Dec}(sk, c_1 \odot pk \text{Dec}(sk, c_2))] > 1 - \text{negl}(\kappa)\).
2. Secondly, for all messages \(m_1\) and \(m_2\), and all coins \(r_1\) and \(r_2\), we have \(\Pr[(pk, sk, m) \leftarrow \text{Gen}(\kappa); m_1, m_2 \in m \Rightarrow \text{Enc}(pk, m_1 \oplus pk m_2, r_1) \odot pk \text{Enc}(pk, m_2, r_2) = \text{Enc}(pk, m_1 \odot pk m_2; r_1 \oplus pk r_2)] > 1 - \text{negl}(\kappa)\).

We say \(\Pi\) is multiplicative homomorphic, if for all security parameters \(\kappa\), key pairs \(pk, sk\), and message spaces \(m\), such that there exists coins \(r\) and \((pk, sk, m) = \text{Gen}(\kappa; r)\), we have \(\odot pk\) is the multiplication operator in group \((m, \odot pk)\).

Definition 5 ([IND-CPA [7]]). Let \(\Pi = (\text{Gen}, \text{Enc}, \text{Dec})\) be an asymmetric encryption scheme, \(A\) be an adversary, \(\kappa\) be the security parameter, and IND-CPA(\(\Pi, \mathcal{A}, \kappa\)) be the following game.\(^\dagger\)

\(^9\)Our definition differs from Katz and Lindell’s original definition [38, Definition 10.1] in that we formally state the plaintext space.

\(^\dagger\)For brevity, we write \(\Pi\) is a homomorphic asymmetric encryption scheme as opposed to the more verbose \(\Pi\) is a homomorphic asymmetric encryption scheme, with respect to ternary operators \(\odot, \oplus, \text{and } \otimes\).

\(^\ast\)We write \(X \circ pk Y\) for the application of ternary operator \(\circ\) to inputs \(X, Y\), and \(pk\). We occasionally abbreviate \(X \circ pk Y\) as \(X \circ Y\), when \(pk\) is clear from the context.

\(^\dagger\)Our definition of an asymmetric encryption scheme explicitly defines the plaintext space, whereas Bellare et al. [7] leave the plaintext space implicit; this change is reflected in our definition of IND-CPA. Moreover, we provide the adversary with the message space and security parameter.
IND-CPA(\Pi, \mathcal{A}, \kappa) =
\begin{align*}
(pk, sk, m) &\leftarrow \text{Gen}(\kappa); \\
(m_0, m_1) &\leftarrow \mathcal{A}(pk, m, \kappa); \\
\beta &\leftarrow_R \{0, 1\}; \\
c &\leftarrow \text{Enc}(pk, m_\beta); \\
g &\leftarrow \mathcal{A}(c); \\
\text{return } g = \beta;
\end{align*}

In the above game, we require $m_0, m_1 \in m$ and $|m_0| = |m_1|$. We say $\Pi$ satisfies IND-CPA, if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function $\text{negl}$, such that for all security parameters $\kappa$, we have $\text{Succ}(\text{IND-CPA}(\Pi, \mathcal{A}, \kappa)) \leq \frac{1}{2} + \text{negl}(\kappa)$.

### A.2 Proof systems

**Definition 6.** Let $(\text{Gen}, \text{Enc}, \text{Dec})$ be a homomorphic asymmetric encryption scheme and $\Sigma$ be a sigma protocol for a binary relation $R$.\(^{13}\)

- $\Sigma$ proves key generation \([61]\) if a $((\kappa, pk, m), (sk, s)) \in R \iff (pk, sk, m) = \text{Gen}(\kappa; s)$.

Further, suppose that $(pk, sk, m)$ is the output of $\text{Gen}(\kappa; s)$, for some security parameter $\kappa$ and coins $s$.

- $\Sigma$ proves ciphertext construction if $((pk, c, m'), (m, r)) \in R \iff c = \text{Enc}(pk, m; r) \land m \in m' \land m' \subseteq m$ \([61]\), or $((pk, g, c, m'), (m, r)) \in R \iff c = \text{Enc}(pk, g^m; r) \land m \in m' \land m' \subseteq m$, where $g$ is a generator of message space $m$.

- $\Sigma$ proves decryption \([61]\) if $((pk, c, m), sk) \in R \iff m = \text{Dec}(sk, c)$.

- $\Sigma$ proves iterative homomorphic combination if $((pk, c, c'), n) \in R \iff \land_{1 \leq i \leq |c|} c[i] = \bigotimes_1^n c'[i] \land |c| = |c'|$.\(^{14}\)

**Definition 7** (Non-interactive proof system \([61]\)). A non-interactive proof system for a relation $R$ is a tuple of algorithms $(\text{Prove}, \text{Verify})$, such that:

- **Prove**, denoted $\sigma \leftarrow \text{Prove}(s, w, \kappa)$, is executed by a prover to prove $(s, w) \in R$.

- **Verify**, denoted $v \leftarrow \text{Verify}(s, \sigma, \kappa)$, is executed by anyone to check the validity of a proof. We assume $\text{Verify}$ is deterministic.

---

\(^{13}\)Given a binary relation $R$, we write $((s_1, \ldots, s_l), (w_1, \ldots, w_k)) \in R \iff P(s_1, \ldots, s_l, w_1, \ldots, w_k) \text{ for } (s, w) \in R \iff P(s_1, \ldots, s_l, w_1, \ldots, w_k) \land s = (s_1, \ldots, s_l) \land w = (w_1, \ldots, w_k)$, hence, $R$ is only defined over pairs of vectors of lengths $l$ and $k$.

\(^{14}\)We write $\text{ProveComb}((pk, c), (c'), n, \kappa)$ for $\text{ProveComb}((pk, (c), (c')), n, \kappa)$ when $c$ and $c'$ are ciphertexts (rather than vectors), where $(\text{ProveComb}, \text{VerComb}) = \text{FS}(\Sigma, \mathcal{H})$ for a sigma protocol $\Sigma$ that proves iterative homomorphic combination and a hash function $\mathcal{H}$. 

21
Moreover, the system must be complete: there exists a negligible function $\text{negl}$, such that for all statement and witnesses $(s, w) \in R$ and security parameters $\kappa$, we have $\Pr[\sigma \leftarrow \text{Prove}(s, w, \kappa) : \text{Verify}(s, \sigma, \kappa) = 1] > 1 - \text{negl}(\kappa)$. A system has perfect completeness if the probability is 1.

**Definition 8** (Fiat-Shamir transformation [28]). Given a sigma protocol $\Sigma = (\text{Comm}, \text{Chal}, \text{Resp}, \text{Verify}_\Sigma)$ for relation $R$ and a hash function $H$, the Fiat-Shamir transformation, denoted $\text{FS}(\Sigma, H)$, is the non-interactive proof system $(\text{Prove}, \text{Verify})$, defined as follows:

\[
\text{Prove}(s, w, \kappa) = \\
(\text{comm}, t) \leftarrow \text{Comm}(s, w, \kappa); \\
\text{chal} \leftarrow H(\text{comm}, s); \\
\text{resp} \leftarrow \text{Resp}(\text{chal}, t, \kappa); \\
\text{return} (\text{comm}, \text{resp});
\]

\[
\text{Verify}(s, (\text{comm}, \text{resp}), \kappa) = \\
\text{chal} \leftarrow H(\text{comm}, s); \\
\text{return} \text{verify}_\Sigma(s, (\text{comm}, \text{chal}, \text{resp}), \kappa);
\]

A string $m$ can be included in the hashes computed by algorithms Prove and Verify. That is, the hashes are computed in both algorithms as $\text{chal} \leftarrow H(\text{comm}, s, m)$. We write $\text{Prove}(s, w, m, \kappa)$ and $\text{Verify}(s, (\text{comm}, \text{resp}), m, \kappa)$ for invocations of Prove and Verify which include string $m$.

**Definition 9** (Zero-knowledge [51]). Let $\Delta = (\text{Prove}, \text{Verify})$ be a non-interactive proof system for a relation $R$, derived by application of the Fiat-Shamir transformation [28] to a random oracle $H$ and a sigma protocol. Moreover, let $S$ be an algorithm, $A$ be an adversary, $\kappa$ be a security parameter, and $\text{ZK}(\Delta, A, H, S, \kappa)$ be the following game.

\[
\text{ZK}(\Delta, A, H, S, \kappa) = \\
\beta \leftarrow_R \{0, 1\}; \\
g \leftarrow A^{\text{H,P}}(\kappa); \\
\text{return} g = \beta;
\]

Oracle $P$ is defined on inputs $(s, w) \in R$ as follows:

- $P(s, w)$ computes if $\beta = 0$ then $\sigma \leftarrow \text{Prove}(s, w, \kappa)$ else $\sigma \leftarrow S(s, \kappa)$ and outputs $\sigma$.

And algorithm $S$ can patch random oracle $H$.\textsuperscript{15} We say $\Delta$ satisfies zero-knowledge, if there exists a probabilistic polynomial-time algorithm $S$, such that for all probabilistic polynomial-time algorithm adversaries $A$, there exists a negligible function $\text{negl}$, and for all security parameters $\kappa$, we have $\text{Succ}(\text{ZK}(\Delta, A, H, S, \kappa)) \leq \frac{1}{2} + \text{negl}(\kappa)$. An algorithm $S$ for which zero-knowledge holds is called a simulator for $(\text{Prove}, \text{Verify})$.

\textsuperscript{15}Random oracles can be programmed or patched. We will not need the details of how patching works, so we omit them here; see Bernhard et al. [12] for a formalisation.
Definition 10 (Simulation sound extractability [12, 30, 61]). Suppose $\Sigma$ is a sigma protocol for relation $R$, $\mathcal{H}$ is a random oracle, and $(\text{Prove}, \text{Verify})$ is a non-interactive proof system, such that $\text{FS}(\Sigma, \mathcal{H}) = (\text{Prove}, \text{Verify})$. Further suppose $S$ is a simulator for $(\text{Prove}, \text{Verify})$ and $\mathcal{H}$ can be patched by $S$. Proof system $(\text{Prove}, \text{Verify})$ satisfies simulation sound extractability if there exists a probabilistic polynomial-time algorithm $K$, such that for all probabilistic polynomial-time adversaries $A$ and coins $r$, there exists a negligible function $\text{negl}$, such that for all security parameters $\kappa$, we have:\footnote{We extend set membership notation to vectors: we write $x \in \mathbf{x}$ if $x$ is an element of the set $\{\mathbf{x}[i] : 1 \leq i \leq |\mathbf{x}|\}$.}

$$\text{Pr}[P \leftarrow \emptyset; Q \leftarrow A^{H,P}(\_ ; r); W \leftarrow K^{A'}(H, P, Q) :$$

$$|Q| \neq |W| \lor \exists j \in \{1, \ldots, |Q|\} \cdot (Q[j][1], W[j]) \notin R$$

$$\forall (s, \sigma) \in Q, (t, \tau) \in P : \text{Verify}(s, \sigma, \kappa) = 1 \land \sigma \neq \tau \leq \text{negl}(\kappa)$$

where $A(\_ ; r)$ denotes running adversary $A$ with an empty input and coins $r$, where $H$ is a transcript of the random oracle’s input and output, and where oracles $A'$ and $P$ are defined below:

- $A'()$. Computes $Q' \leftarrow A(\_ ; r)$, forwarding any of $A$’s oracle queries to $K$, and outputs $Q'$. By running $A(\_ ; r)$, $K$ is rewinding the adversary.

- $P(s)$. Computes $\sigma \leftarrow S(s, \kappa); P \leftarrow (P[1], \ldots, P[|P|], (s, \sigma))$ and outputs $\sigma$.

Algorithm $K$ is an extractor for $(\text{Prove}, \text{Verify})$.

Theorem 5 (from [12]). Let $\Sigma$ be a sigma protocol for relation $R$, and let $\mathcal{H}$ be a random oracle. Suppose $\Sigma$ satisfies special soundness and special honest verifier zero-knowledge. Non-interactive proof system $\text{FS}(\Sigma, \mathcal{H})$ satisfies zero-knowledge and simulation sound extractability.

The Fiat-Shamir transformation may include a string in the hashes computed by functions $\text{Prove}$ and $\text{Verify}$. Simulators can be generalised to include such a string too. We write $S(s, m, \kappa)$ for invocations of simulator $S$ which include string $m$. And remark that Theorem 5 can be extended to this generalisation.

B Proof of correctness (Lemma 1)

Let $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$, $\text{FS}(\Sigma_1, \mathcal{H}) = (\text{ProveKey}, \text{VerKey})$, $\text{FS}(\Sigma_2, \mathcal{H}) = (\text{ProveCiph}, \text{VerCiph})$, and $\text{Athena}(\Pi, \mathcal{M}, \Sigma_1, \Sigma_2, \Sigma_3, \Sigma_4, \mathcal{H}) = (\text{Setup}, \text{Register}, \text{Vote}, \text{Tally}, \text{Verify})$.

Suppose $\kappa$ is a security parameter, $nv$ and $nc$ are integers, $v_1, \ldots, v_{nv}$ are vectors over $\{1, \ldots, nc\}$, $c_1, \ldots, c_{nc}$ are vectors such that $\bigwedge_{1 \leq i \leq nc} |v_i| = |c_i| \land c_i[1] < \cdots < c_i[|c_i|]$, and $\mathbf{v}$ is a zero-filled vector of length $nc$. Further suppose we compute:
(pk, sk, mb, mc) ← Setup(κ);
bb ← 0;
for 1 ≤ i ≤ nv do
  (pd_i, d_i) ← Register(pk, κ);
  if 0 < |v_i| then
    for 1 ≤ j ≤ |v_i| do
      b_{i,j} ← Vote(d_i, pk, v_i[j], c_i[j], nc, κ);
      v[v_i[|v_i|]] ← v[v_i[|v_i|]] + 1;

If |bb| ≤ mb ∨ nc ≤ mc, then correctness is trivially satisfied, otherwise (|bb| ≤ mb ∧ nc ≤ mc), we proceed as follows.

By definition of algorithm Setup, we have pk parses as vector (pk, m, ρ) and sk parses as (pk, sk), where (pk, sk, m) = Gen(κ; r) for some coins r and ρ is an output of ProveKey((κ, pk, m), (sk, r), κ). Moreover, by completeness, we have VerKey((κ, pk, m), ρ, κ) holds. Let g be a generator of message space m. By definition of algorithm Register, we have for each i ∈ {1, . . . , nv} that pd_i = Enc(pk, g^{d_i}; r_i) and d_i = (pd_i, d_i), for some coins r_i chosen uniformly at random and nonce d_i. Moreover, by definition of algorithm Vote, we have for each i ∈ {1, . . . , nv} and j ∈ {1, . . . , |v_i|} that b_{i,j} is a vector of length six,

b_{i,j}[1] = pd_i,
b_{i,j}[2] = Enc(pk, g^{-d_i}; s_{i,j}),
b_{i,j}[3] = Enc(pk, v_i[j]; t_{i,j}),
b_{i,j}[4] is an output of ProveCiph((pk, g, b_{i,j}[2], m), (−d_i, s_{i,j}), m, κ), b_{i,j}[5] is an output of ProveCiph((pk, b_{i,j}[3], {1, . . . , nc}), (v_i[j], t_{i,j}), m, κ), and b_{i,j}[6] = c_i[j], where s_{i,j} and t_{i,j} are coins chosen uniformly at random and m = (pd_i, b_{i,j}[2], b_{i,j}[3], b_{i,j}[6]). Let us consider the computation of (v′, pf) by Tally(sk, bb, nc, {pd_1, . . . , pd_nv}, κ).

We have bb = \bigcup_{1 ≤ i ≤ nv \land |v_i| > 0} \{b_{i,1}, . . . , b_{i,|v_i|}\}. Suppose a subset of that set is computed as per Step 1 of algorithm Tally. By completeness and since for each i ∈ {1, . . . , nv} we have pd_i = b_{i,1}[1] = . . . = b_{i,|v_i|}[1], that subset is \{b_{π(1)}, . . . , b_{π(|v_i|)}\}, for some permutation π on \{1, . . . , nv\} and for each i ∈ {1, . . . , nv} some permutation π_i on \{1, . . . , |v_i|\} such that \(b_{π(1)}, . . . , b_{π(|v_i|)}[1] ≤ . . . ≤ b_{π(1)}, . . . , b_{π(|v_i|)}[1] ≤ . . . ≤ b_{π(i)}, . . . , b_{π(i)}[v_i][1]\}. If nv = 0 ∨ \bigwedge_{1 ≤ i ≤ nv} |v_i| = 0, then v and v′ are both zero-filled vectors of length nc, and we conclude immediately, otherwise, we proceed as follows.

Suppose ciphertexts, plaintexts, and a map are computed as per Step 2 of algorithm Tally, with respect to nonce n. Since Π is a multiplicative-homomorphic asymmetric encryption scheme, we have for each i ∈ {1, . . . , nv} and j ∈ {1, . . . , |v_i|} that

\(c_{i,j}[n] = \bigotimes_{1}^{n} b_{i,j}[2] = Enc(pk, \otimes_{1}^{n} g^{-d_i}; \oplus_{1}^{n} s_{i,j}) = Enc(pk, g^{-d_i}; \oplus_{1}^{n} s_{i,j})\),
hence, by (perfect) correctness, we have
\[
N_{i,j} = \text{Dec}(sk, c'_{i,j}) \equiv g^{-d_i},
\]
where ciphertext \( c'_{i,j} \) and plaintext \( N_{i,j} \) are computed by algorithm Tally. (We require perfect correctness, because the adopted definition of homomorphic encryption only considers combination of distinct ciphertexts constructed from distinct coins, whereas we consider iterative combination of a single ciphertext.) Hence, for each \( i \in \{1, \ldots, n\} \) we have
\[
N_{i,1} = \cdots = N_{i,|v_i|}.
\]
Since \( d_1, \ldots, d_{n^v} \) are nonces, \( g \) is a generator of message space \( m \), and \(|m|\) is super-polynomial in the security parameter, we have \( N_{1,|v_1|}, \ldots, N_{n^v,|v_n^v|} \) are pairwise distinct, moreover, since \(|v_i| = |c_i| \land c_i[1] < \cdots < c_i[|c_i|]\) and \( b_{i,j}[6] = c_i[j] \) for \( j \in \{1, \ldots, |c_i|\} \), we have for each \( i \in \{1, \ldots, n\} \) that
\[
A[(pd_i, N_{i,1})] = (b_{i,|v_i|}[6], b_{i,|v_i|}[1] \otimes b_{i,|v_i|}[2], b_{i,|v_i|}[3]),
\]
where map \( A \) is computed by algorithm Tally. It follows that map \( A \) is defined over ciphertexts \( b_{1,|v_1|}[1] \otimes b_{1,|v_1|}[2], b_{1,|v_1|}[3], \ldots, b_{n^v,|v_n^v|}[1] \otimes b_{n^v,|v_n^v|}[2], b_{n^v,|v_n^v|}[3] \). Suppose mixnet \( M \) is applied to those pairs of ciphertexts to derive vector \( B = (b_1, \ldots, b_{n^v}) \), as per Step 2 of algorithm Tally. Since \( \Pi \) is a multiplicative-homomorphic asymmetric encryption scheme, we have for each \( i \in \{1, \ldots, n\} \) that
\[
\begin{align*}
  b_i[1] &= \text{Enc}(pk, g^{d_i} \otimes g^{-d_i}; r_i \oplus s_{i,|v_i|} \oplus w_i) \\
  &= \text{Enc}(pk, g^{d_i}; r_i \oplus s_{i,|v_i|} \oplus w_i) \\
  b_i[2] &= \text{Enc}(pk, v_i[|v_i|]; t_{i,|v_i|} \oplus x_i),
\end{align*}
\]
where \( i \) denotes \( \chi(i) \) and \( \chi \) is a permutation over \( \{1, \ldots, n\} \) and coins \( w_i \) and \( x_i \) were introduced during mixing.

Suppose for each \( i \in \{1, \ldots, n\} \) that \( c' = \bigotimes_i^{n^v} b_i[1] \) and \( m = \text{Dec}(sk, c') \) are computed as per Step 3 of algorithm Tally. It follows by (perfect) correctness and homomorphic properties that \( m = 1 \). Moreover, \( \text{Dec}(sk, b_i[2]) = v_{\chi(i)}[|v_{\chi(i)}|] \) for each \( i \in \{1, \ldots, n\} \). Since \( \chi \) is a permutation over \( \{1, \ldots, n\} \), it follows that \( v \) is equivalent to the outcome that would be computed by Step 3 of algorithm Tally, which concludes our proof. \( \square \)

### C Verifiability by Smyth, Frink & Clarkson

We cast the verifiability definitions by Smyth, Frink & Clarkson [61] into the context of our syntax, extend their definition of Soundness to include re-voting, strengthen definitions of Injectivity, Individual-Verifiability and Unforgeability, and incorporate some minor refinements by Smyth [59, 60]. The definition of Completeness remains unchanged (beyond syntax changes).
C.1 Universal verifiability

Universal verifiability requires algorithm Verify to accept if and only if the election outcome is correct. The only if requirement is captured by soundness and the if requirement is captured by completeness.

**Soundness.** Correct outcomes are formalised using function correct-outcome. The function uses a predicate ($\exists^\ell x : P(x)$) that holds exactly when there are $\ell$ distinct values of $x$ for which $P(x)$ is satisfied [55]. Using the predicate, function correct-outcome is defined such that

$$correct-outcome(pk, nc, bb, M, \kappa)[v] = \ell \text{ iff } \exists^\ell b \in authorised(pk, nc, (bb \setminus \{\bot\}), M, \kappa) : $$

$$\exists d, cnt, r : b = Vote(d, pk, v, cnt, nc, \kappa; r),$$

where correct-outcome$(pk, nc, bb, M, \kappa)$ is a vector of length $nc$, $1 \leq v \leq nc$, and

$$authorised(pk, nc, bb, M, \kappa) =$$

$$\{b_k \mid \exists!b_1, \ldots, b_k \in bb : \exists cnt_1, \ldots, cnt_k : cnt_1 \leq \cdots \leq cnt_{k-1} < cnt_k$$

$$\wedge \exists(pd, d) \in M : \bigwedge_{1 \leq j \leq k} \exists v, r : b_j = Vote(d, pk, v, cnt_j, nc, \kappa; r)$$

$$\wedge \neg \exists b \in bb \setminus \{b_1, \ldots, b_k\}, v, cnt, r : b = Vote(d, pk, v, cnt, nc, \kappa; r)\}.$$  

Function authorised discards all ballots submitted under the same credential, except for a ballot containing the last vote. Hence, component $v$ of vector correct-outcome$(pk, nc, bb, M, \kappa)$ equals $\ell$ iff there exist $\ell$ authorised ballots for vote $v$ on the bulletin board. Function correct-outcome requires that ballots be interpreted for only one candidate, which can be ensured by injectivity, i.e., a ballot for vote $v$ can never be interpreted for a distinct vote $v'$.  

**Definition 11** (Injectivity). An election scheme $(\text{Setup, Register, Vote, Tally, Verify})$ satisfies injectivity, if for all probabilistic polynomial-time adversaries $A$, security parameters $\kappa$ and computations $(pk, nc, d_1, v_1, cnt_1, d_2, v_2, cnt_2) \leftarrow A(\kappa); b_1 \leftarrow Vote(d_1, pk, v_1, cnt_1, nc, \kappa); b_2 \leftarrow Vote(d_2, pk, v_2, cnt_2, nc, \kappa)$ such that $v_1 \neq v_2 \wedge b_1 \neq \bot \wedge b_2 \neq \bot$, we have $b_1 \neq b_2$.

Equipped with a notion of correct outcomes, we formalise soundness (Definition 12) as a game that tasks the adversary to compute inputs to algorithm Verify – including an election outcome and some ballots – that cause the algorithm to accept an incorrect outcome.

**Definition 12** (Soundness). Let $\Gamma = (\text{Setup, Register, Vote, Tally, Verify})$ be an election scheme, $A$ be an adversary, $\kappa$ be a security parameter, and Soundness$(\Gamma; A, \kappa)$ be the following game.
Soundness($\Gamma, \mathcal{A}, \kappa$) =

$$(pk, nv) \leftarrow \mathcal{A}(\kappa);$$

for $1 \leq i \leq nv$ do $(pd_i, d_i) \leftarrow \text{Register}(pk, \kappa);$  
$L \leftarrow \{pd_1, \ldots, pd_nv\};$
$M \leftarrow \{(pd_1, d_1), \ldots, (pd_nv, d_nv)\};$
$$(bb, nc, v, pf) \leftarrow \mathcal{A}(M);$$
return Verify$(pk, bb, nc, L, v, pf, \kappa) = 1$

$\land v \neq \text{correct-outcome}(pk, nc, bb, M, \kappa);$  

We say $\Gamma$ satisfies Soundness, if $\Gamma$ satisfies injectivity and for all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function $\text{negl}$, such that for all security parameters $\kappa$, we have $\text{Succ}(\text{Soundness}(\Gamma, \mathcal{A}, \kappa)) \leq \text{negl}(\kappa)$.

Completeness. We formalise completeness (Definition 13) as a game that tasks the adversary to compute a bulletin board and some number of candidates such that the corresponding election outcome computed by algorithm Tally is rejected by algorithm Verify, when the key pair is computed by algorithm Setup and voter credentials are computed by algorithm Register.

Definition 13 (Completeness). Let $\Gamma = (\text{Setup}, \text{Register}, \text{Vote}, \text{Tally}, \text{Verify})$ be an election scheme, $\mathcal{A}$ be an adversary, $\kappa$ be a security parameter, and $\text{Completeness}(\Gamma, \mathcal{A}, \kappa)$ be the following game.

Completeness($\Gamma, \mathcal{A}, \kappa$) =

$$(pk, sk, mb, mc) \leftarrow \text{Setup}(\kappa);$$
$nv \leftarrow \mathcal{A}(pk, \kappa);$  
for $1 \leq i \leq nv$ do $(pd_i, d_i) \leftarrow \text{Register}(pk, \kappa);$  
$L \leftarrow \{pd_1, \ldots, pd_nv\};$
$M \leftarrow \{(pd_1, d_1), \ldots, (pd_nv, d_nv)\};$
$$(bb, nc) \leftarrow \mathcal{A}(M);$$
$$(v, pf) \leftarrow \text{Tally}(sk, bb, nc, L, \kappa);$$
return Verify$(pk, bb, nc, L, v, pf, \kappa) \neq 1 \land |bb| \leq mb \land nc \leq mc;$

We say $\Gamma$ satisfies Completeness, if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function $\text{negl}$, such that for all security parameters $\kappa$, we have $\text{Succ}(\text{Completeness}(\Gamma, \mathcal{A}, \kappa)) \leq \text{negl}(\kappa)$.

C.2 Unforgeability

We formalise unforgeability (Definition 14) as a game that tasks the adversary to compute a ballot containing a private credential.

Definition 14 (Unforgeability). Let $\Gamma = (\text{Setup}, \text{Register}, \text{Vote}, \text{Tally}, \text{Verify})$ be an election scheme, $\mathcal{A}$ be an adversary, $\kappa$ be a security parameter, and $\text{Unforgeability}(\Gamma, \mathcal{A}, \kappa)$ be the following game.

Unforgeability($\Gamma, \mathcal{A}, \kappa$) =
(pk, nv) ← A(κ);
for 1 ≤ i ≤ nv do (pd_i, d_i) ← Register(pk, κ);
L ← {pd_1, ..., pd_nv};
Crpt ← ∅; Rvld ← ∅;
b ← A(c, R(L));
return ∃i ∈ {1, ..., nv}, v, cnt, nc, r : b = Vote(d_i, pk, v, cnt, nc, κ; r)
∧ b ≠ ⊥ ∧ b ∉ Rvld ∧ d_i ∉ Crpt;

Oracles C and R are defined such that:

• C(i) computes Crpt ← Crpt ∪ {d_i} and outputs d_i, where 1 ≤ i ≤ nv, and
• R(i, v, cnt, nc) computes b ← Vote(d_i, pk, v, cnt, nc, κ); Rvld ← Rvld ∪ {b}
  and outputs b.

We say Γ satisfies Unforgeability, if for all probabilistic polynomial-time ad-
versaries A, there exists a negligible function negl, such that for all security
parameters κ, we have Succ(Unforgeability(Γ, A, κ)) ≤ negl(κ).

C.3 Individual verifiability

We formalise individual verifiability (Definition 15) as a game that tasks the
adversary to compute inputs to algorithm Vote that cause the algorithm to
output ballots that collide.17

Definition 15 (Individual verifiability). Let Γ = (Setup, Register, Vote, Tally,
Verify) be an election scheme, A be an adversary, κ be a security parameter,
and Individual-Verifiability(Γ, A, κ) be the following game.

Individual-Verifiability(Γ, A, κ) =
(pk, nc, d_1, v_1, cnt_1, d_2, v_2, cnt_2) ← A(κ);
b_1 ← Vote(d_1, pk, v_1, cnt_1, nc, κ);
b_2 ← Vote(d_2, pk, v_2, cnt_2, nc, κ);
return b_1 = b_2 ∧ b_1 ≠ ⊥ ∧ b_2 ≠ ⊥;

We say Γ satisfies Individual-Verifiability, if for all probabilistic polynomial-time
adversaries A, there exists a negligible function negl, such that for all security
parameters κ, we have Succ(Individual-Verifiability(Γ, A, κ)) ≤ negl(κ).

D Proof of Propositions 2–4 (verifiability)

D.1 Proof of Proposition 2 (Soundness)

Let II = (Gen, Enc, Dec), FS(Σ_1, ℋ) = (ProveKey, VerKey), FS(Σ_2, ℋ) =
(ProveCiph, VerCiph), FS(Σ_3, ℋ) = (ProveDec, VerDec), FS(Σ_4, ℋ) =
(ProveComb, VerComb), and Athena(II, M, Σ_1, Σ_2, Σ_3, Σ_4, ℋ) = (Setup, Register,
Vote, Tally, Verify).

17Correctness, individual verifiability and injectivity all require that ballots do not collide,
albeit under different assumptions.
Suppose \( \mathcal{A} \) is a probabilistic polynomial-time adversary, \( \kappa \) is a security parameter, \((pk, nv)\) is an output of \( \mathcal{A}(\kappa) \), and \((pd_1, d_1), \ldots, (pd_{nv}, d_{nv})\) are outputs of \( \text{Register}(pk, \kappa) \). Let \( L = \{pd_1, \ldots, pd_{nv}\} \) and \( M = \{(pd_1, d_1), \ldots, (pd_{nv}, d_{nv})\} \).

Suppose \((\mathbb{b}, nc, v, pf)\) is an output of \( \mathcal{A}(M) \) such that \( \text{Verify}(pk, \mathbb{b}, nc, L, v, pf, \kappa) = 1 \). By definition of algorithm \( \text{Verify} \), public key \( pk \) parses as a vector \((pk, m, \rho)\) and outcome \( v \) parses as a vector of length \( nc \) such that \( \text{VerDec}(\mathbb{b}) \) is an output of \( \mathcal{A}(pk, m, \rho, nc) \) and private credential \( pd = \text{Enc}(pk, g^{d_i}; r_i) \) and public credential \( mc = \text{Enc}(pk, m, \kappa) \). Thus, for all ballots \( b \in \{b_1, \ldots, b_\ell\} \) and \( v \), the outcome \( \text{correct-outcome}(pk, nc, b, M, \kappa) \) is computed as per Step 1 of algorithm \( \text{Verify} \).

A proof of (1) follows from the definition of \( \text{correct-outcome} \). If \( \{b_1, \ldots, b_\ell\} = \emptyset \), then outcome \( v \) and \( \text{correct-outcome}(pk, nc, b_1, \ldots, b_\ell, M, \kappa) \) are zero-filled vectors of length \( nc \), hence, \( \text{Soundness} \) is satisfied. Otherwise, we proceed as follows.

By simulation sound extractability, we have for each \( i \in \{1, \ldots, \ell\} \) that there exists messages \( d'_i \in m \) and \( v_i \in \{1, \ldots, nc\} \) and coins \( s_i \) and \( t_i \) such that

\[
\begin{align*}
    b_i[2] &= \text{Enc}(pk, g^{d'_i}; s_i), \\
    b_i[3] &= \text{Enc}(pk, v_i; t_i),
\end{align*}
\]

\( b_i[4] \) is an output of \( \text{ProveCiph}(pk, g, b_i[2], m, (d'_i, s_i), m, \kappa) \), and \( b_i[5] \) is an output of \( \text{ProveCiph}(pk, b_i[3], \{1, \ldots, nc\}, (v_i, t_i), m, \kappa) \), where \( m = (b_1[1], b_2[2], b_i[3], b_i[6]) \). It follows by inspection of algorithm \( \text{Vote} \) that \( \forall i \in \{1, \ldots, \ell\}, \exists r : b_i = \text{Vote}(-d'_i, pk, v_i, b_i[6], nc, \kappa) \), hence, \{b_1, \ldots, b_\ell\} is a set of ballots.

By Step 2 of algorithm \( \text{Verify} \), we have that \( pf \) parses as a vector \((\mathbf{pfr}, B, \mathbf{pfI})\) and \( \mathbf{pfr} \) parses as a vector \((c'_1, N_1, c_1), (c'_2, N_2, c_2, \omega_2), \ldots, (c'_\ell, N_\ell, c_\ell, \omega_\ell)\) such that \( \bigwedge_{1 \leq i \leq \ell} \text{VerDec}(pk, c'_i, N_i, \kappa, c_i) \) and \( \bigwedge_{1 \leq i \leq \ell} \text{VerComb}(pk, c'_{i-1}, c'_i, \omega_{i-1}[2], b_i[2]) \). By simulation sound extractability, there exists a nonce \( n \) such that for all \( i \in \{1, \ldots, \ell\} \) we have

\[
c'_i = \bigotimes_1^n b_i[2] = \text{Enc}(pk, \odot_1^n g^{d'_i}, \oplus_1^n s_i) = \text{Enc}(pk, g^{d'_i}; \oplus_1^n s_i)
\]
and $\text{Dec}(sk, c'_i) = N_i$, moreover, by (perfect) correctness, we have

$$N_i \equiv g^{d'_{i,n}}.$$  

Let map $A$ be computed as per Step 2 of algorithm $\text{Verify}$. It follows for each $i \in \{1, \ldots, \ell\}$ that

$$A[(b_i[1], N_i)] = (b_i[6], b_i[1] \otimes b_i[2], b_i[3]) \leftrightarrow
- \exists j \{1, \ldots, \ell\} \setminus \{i\} : b_j[1] = b_i[1] \land N_j = N_i \land b_j[6] \leq b_i[6]$$

i.e., public credential $b_i[1]$ and anonymised credential $N_i$ are mapped to a triple derived from ballot $b_i$ iff there is no other ballot $b_j$ with the same public credential and the same anonymised credential that has a greater-than or equal-to counter value. Hence, for each anonymised credential, map $A$ contains the encrypted vote associated with the highest counter, that is, the last vote related to the credential. Since (pairwise) mixnet $M$ is verifiable and since Step 2 of algorithm $\text{Verify}$ checks that $B$ was output by the mixnet, there exists an injective function $\chi : \{1, \ldots, |B|\} \to \{1, \ldots, \ell\}$ such that for each $i \in \{1, \ldots, |B|\}$ we have $B[i]$ is a pair $(c_1, c_2)$,

$$c_1 = \text{Enc}(pk, g^{d(x(\lambda(i)))} \odot g^{d'(x(\lambda(i)))}; r_{x(\lambda(i))} \odot s_{x(i)} \oplus w_i), \text{ and}$$

$$c_2 = \text{Enc}(pk, v_{x(i)}; t_{x(i)} \oplus x_i),$$

where coins $w_i$ and $x_i$ were introduced during mixing. It follows that

$$\text{authorised}(pk, nc, \{b_i \mid 1 \leq i \leq |B|\}, M, \kappa) = \text{authorised}(pk, nc, \{b_i[6] \mid 1 \leq i \leq |B|\}, M, \kappa) \quad (2)$$

because any ballot that shares a public credential (and an anonymised credential) with another ballot, whilst being associated with a (strictly) lower counter value can be discarded, as can any pair of ballots that share a public credential (and a anonymised credential) and a counter.

By Step 3 of algorithm $\text{Verify}$, we have for each $i \in \{1, \ldots, |B|\}$ that $B[i]$ parses as a vector $(c_1, c_2)$ and $\text{pfd}[i]$ parses a vector $(c', m, \omega, \varsigma_1)$ or $(c', v, \omega, \varsigma_1, \varsigma_2)$, such that $\text{VerDec}(pk, c', 1, \varsigma_1, \kappa)$ when $|\text{pfd}[i]| = 5$ and $\text{VerDec}(pk, c', m, \varsigma_1, \kappa) \land m \neq 1$ when $|\text{pfd}[i]| = 4$, hence, by simulation sound extractability, (perfect) correctness, and multiplicatively-homomorphic properties, we have

$$|\text{pfd}[i]| = 5 \leftrightarrow \text{Dec}(sk, c') = 1 \Leftrightarrow 1 \equiv g^{d(x(\lambda(i)))} \odot g^{d'(x(\lambda(i)))} \Leftrightarrow d'_{x(i)} \equiv -d_{x(\lambda(i))}.$$  

It follows that $b_{\chi(i)}$ is constructed from $(pd_{x(\lambda(i))}, d_{x(\lambda(i))}) \in M$ iff $|\text{pfd}[i]| = 5$, where $i \in \{1, \ldots, |B|\}$, hence,

$$\text{authorised}(pk, nc, \{b_i \mid 1 \leq i \leq |B|\}, M, \kappa) = \text{authorised}(pk, nc, \{b_i \mid 1 \leq i \leq |B| \land |\text{pfd}[i]| = 5\}, M, \kappa) = \{b_i \mid 1 \leq i \leq |B| \land |\text{pfd}[i]| = 5\} \quad (3)$$

30
because any ballot not constructed from \((pd, d) \in M\) can be discarded and no further ballots can. Moreover, it follows from the remainder of Step 3 that for each \(v \in \{1, \ldots, nc\}\) we have \(\exists v[i] \in \{1, \ldots, |B|\} : \exists c_1, c_2, c', \omega, c_1', c_2' : (c_1, c_2) = B[i] \land (c', v, \omega, c_1', c_2') = pfd[i] \land \text{VerDec}(pk, c_2, v, c_2, \kappa)\), hence, by simulation sound extractability, we have
\[
\exists v[i] \in \{1, \ldots, |B|\} \land |pfd[i]| = 5 : \exists c_1, c_2 : (c_1, c_2) = B[i] \land \text{Dec}(sk, c_2) = v,
\]
furthermore, by (perfect) correctness, we have
\[
\exists v[i] \in \{1, \ldots, |B|\} \land |pfd[i]| = 5 : v = \chi(v(i)).
\]
If follows for each \(v \in \{1, \ldots, nc\}\) that
\[
\exists v[i] \in \{1, \ldots, |B|\} \land |pfd[i]| = 5 : \exists d, cnt, r : b = \text{Vote}(d, pk, v, cnt, nc; \kappa; r).
\]
Finally, by (1)–(3) and since error symbol \(\perp\) is not a vector, we have \(v = \text{correct-outcome}(pk, nc, bb, M; \kappa)\), concluding our proof. 

\section{Proof of Proposition 3 (Completeness)}

Let \(\Pi = (\text{Gen}, \text{Enc}, \text{Dec})\), \(\text{FS}(\Sigma_1, \mathcal{H}) = (\text{ProveKey}, \text{VerKey})\), \(\text{FS}(\Sigma_2, \mathcal{H}) = (\text{ProveCiph}, \text{VerCiph})\), \(\text{FS}(\Sigma_3, \mathcal{H}) = (\text{ProveDec}, \text{VerDec})\), \(\text{FS}(\Sigma_4, \mathcal{H}) = (\text{ProveComb}, \text{VerComb})\), and \(\text{Athena}(\Pi, \mathcal{M}, \Sigma_1, \Sigma_2, \Sigma_3, \Sigma_4, \mathcal{H}) = (\text{Setup}, \text{Register}, \text{Vote}, \text{Tally}, \text{Verify})\).

Suppose \(\kappa\) is a security parameter and \(\mathcal{A}\) is a probabilistic polynomial-time adversary. Further suppose \((pk, sk, mb, mc)\) is an output of \(\text{Setup}(\kappa)\), \(nv\) is an output of \(\text{Setup}(\kappa, \kappa)\), and \((pd_1, d_1), \ldots, (pd_{nv}, d_{nv})\) are outputs of \(\text{Register}(pk, \kappa)\). Let \(L = \{pd_1, \ldots, pd_{nv}\}\) and \(M = \{(pd_1, d_1), \ldots, (pd_{nv}, d_{nv})\}\). Suppose \((bb, nc)\) is an output of \(\text{A}(M)\) and \((v, pf)\) is an output of \(\text{Tally}(sk, bb, nc, L, \kappa)\). If \([bb| \not\equiv mb \lor nc \not\equiv mc\) then we conclude immediately, otherwise \((|bb| \leq mb \land |mc| \leq mc)\) we proceed as follows. By definition of algorithm \(\text{Setup}\), we have \(pk\) parses as \((pk, m, \rho)\) and \(sk\) parses as \((pk, sk)\), where \((pk, sk, m) = \text{Gen}(\kappa; r)\) and \(\rho\) is an output of \(\text{ProveKey}((\kappa, pk, m), (sk, r), \kappa)\), for some coins \(r\) chosen uniformly at random. Moreover, by definition of algorithm \(\text{Tally}\), we have \(v\) is a vector of length \(nc\). It follows that algorithm \(\text{Verify}\) can parse inputs correctly. Moreover, by completeness, we have \(\text{VerKey}((\kappa, pk, m), \rho, \kappa) = 1\).

Suppose subset \(\{b_1, \ldots, b_\ell\}\) is computed as per Step 1 of algorithm \(\text{Tally}\). If that set is empty, then \(v\) is a zero-filled vector, because \(v\) is initialised as a zero-filled vector by algorithm \(\text{Tally}\). Thus, the check holds in Step 1 of algorithm \(\text{Verify}\).

By Step 1 of algorithm \(\text{Tally}\), we have for each \(i \in \{1, \ldots, \ell\}\) that \(b_i\) parses as \((pd, c_1, c_2, \sigma_1, \sigma_2, cnt)\) such that \(pd \in L \land \text{VerCiph}(pk, g, c_1, m, \sigma_1, m, \kappa) \land \text{VerCiph}(pk, c_2, \{1, \ldots, nc\}, \sigma_2, m, \kappa)\), where \(m = (pd, c_1, c_2, cnt)\). Hence, there exists an integer \(j \in \{1, \ldots, nv\}\) such that \(pd = pd_j\). It follows by definition of algorithm \(\text{Register}\) that \(b_i[1] = \text{Enc}(pk, g^{d_i}; r_j)\), for some coins \(r_j\).
chosen uniformly at random and nonce $d_j$ such that private credential $d_j = (p_d_j, d_j)$. Moreover, since $\Sigma_2$ satisfies special soundness and special honest verifier zero-knowledge, we have by simulation sound extractability that $b_1[2] = \text{Enc}(pk, g^{\overline{m}}; s_j)$ and $b_1[3] = \text{Enc}(pk, v_j; t_j)$, for some coins $s_j$ and $t_j$, plaintext $d_j \in m$, and vote $v_j \in \{1, \ldots, nc\}$. It follows that the map ($A$) computed in Step 2 of algorithm Tally is populated with pairs of ciphertexts. Thus, vector $B - \text{derived by application of (pairwise) mixnet $M$ to map $A$ in Step 2 of algorithm Tally} - \text{passes the check in Step 2 of algorithm Verify, because $M$ is verifiable. The preceding checks also pass. Indeed, by definition of algorithm Tally, it is trivial to see that $pf$ parses as a vector ($pf_r, B, pf_d$). Moreover, the vector ($pf_r$) computed in Step 2 of algorithm Tally parses as $((c'_1, N_1, c_1), (c'_2, N_2, \omega_2), \ldots, (c'_j, N_l, \omega_l))$ and, by completeness, the proofs in that vector pass the checks in Step 2 of algorithm Verify. Thus, checks hold in Step 2 of algorithm Verify.}

By Step 3 of algorithm Tally, we have $pf_d$ parses as a vector of length $|B|$, hence, Step 3 of algorithm Verify successfully parses that vector. Since $v$ is initialised as a zero-filled vector of length $nc$ and $v[v]$ is incremented by one for each $i \in \{1, \ldots, |B|\}$ such that $\Sigma_2$ satisfies special soundness and special honest verifier zero-knowledge, we have by simulation sound extractability that $b_1[2] = \text{Enc}(pk, g^{\overline{m}}; s_j)$ and $b_1[3] = \text{Enc}(pk, v_j; t_j)$, for some coins $s_j$ and $t_j$, plaintext $d_j \in m$, and vote $v_j \in \{1, \ldots, nc\}$. It follows that the map ($A$) computed in Step 2 of algorithm Tally is populated with pairs of ciphertexts. Thus, vector $B - \text{derived by application of (pairwise) mixnet $M$ to map $A$ in Step 2 of algorithm Tally} - \text{passes the check in Step 2 of algorithm Verify, because $M$ is verifiable. The preceding checks also pass. Indeed, by definition of algorithm Tally, it is trivial to see that $pf$ parses as a vector ($pf_r, B, pf_d$). Moreover, the vector ($pf_r$) computed in Step 2 of algorithm Tally parses as $((c'_1, N_1, c_1), (c'_2, N_2, \omega_2), \ldots, (c'_j, N_l, \omega_l))$ and, by completeness, the proofs in that vector pass the checks in Step 2 of algorithm Verify. Thus, checks hold in Step 2 of algorithm Verify.}

By Step 3 of algorithm Tally, we have $pf_d$ parses as a vector of length $|B|$, hence, Step 3 of algorithm Verify successfully parses that vector. Since $v$ is initialised as a zero-filled vector of length $nc$ and $v[v]$ is incremented by one for each $i \in \{1, \ldots, |B|\}$ such that $\Sigma_2$ satisfies special soundness and special honest verifier zero-knowledge, we have by simulation sound extractability that $b_1[2] = \text{Enc}(pk, g^{\overline{m}}; s_j)$ and $b_1[3] = \text{Enc}(pk, v_j; t_j)$, for some coins $s_j$ and $t_j$, plaintext $d_j \in m$, and vote $v_j \in \{1, \ldots, nc\}$. It follows that the map ($A$) computed in Step 2 of algorithm Tally is populated with pairs of ciphertexts. Thus, vector $B - \text{derived by application of (pairwise) mixnet $M$ to map $A$ in Step 2 of algorithm Tally} - \text{passes the check in Step 2 of algorithm Verify, because $M$ is verifiable. The preceding checks also pass. Indeed, by definition of algorithm Tally, it is trivial to see that $pf$ parses as a vector ($pf_r, B, pf_d$). Moreover, the vector ($pf_r$) computed in Step 2 of algorithm Tally parses as $((c'_1, N_1, c_1), (c'_2, N_2, \omega_2), \ldots, (c'_j, N_l, \omega_l))$ and, by completeness, the proofs in that vector pass the checks in Step 2 of algorithm Verify. Thus, checks hold in Step 2 of algorithm Verify.}

Since all the above checks succeed, algorithm Verify outputs 1, concluding our proof.

D.3 Proof of Proposition 4 (Individual-Verifiability & Injectivity)

Let $\text{Athena}(\Pi, M, \Sigma_1, \Sigma_2, \Sigma_3, \Sigma_4, \mathcal{H}) = (\text{Setup}, \text{Register}, \text{Vote}, \text{Tally}, \text{Verify})$. Suppose $\mathcal{A}$ is a probabilistic polynomial-time adversary and $\kappa$ is a security. Further suppose $(pk, nc, d_1, v_1, cnt_1, d_2, v_2, cnt_2)$ is an output of $\mathcal{A}^{(\kappa)}$, $b_1$ is an output of $\text{Vote}(d_1, pk, v_1, cnt_1, nc, \kappa)$, and $b_2$ is an output of $\text{Vote}(d_2, pk, v_2, cnt_2, nc, \kappa)$, such that $b_1 \neq \bot$ and $b_2 \neq \bot$. By definition of algorithm Vote, public key $pk$ is a vector $(pk, m, \rho)$ such that $\text{VerKey}(\kappa, pk, m, \rho, \kappa) = 1$ and $v_1, v_2 \in \{1, \ldots, nc\} \subseteq m$. Moreover, $b_1$ and $b_2$ are vectors such that $b_1[2]$ is an output of $\text{Enc}(pk, v_1)$ and $b_2[2]$ is an output of $\text{Enc}(pk, v_2)$. Thus, $b_1 \neq b_2$ by our precondition, with overwhelming probability, therefore, Individual-Verifiability is satisfied. For Injectivity, we further suppose $v_1 \neq v_2$, hence, $b_1 \neq b_2$ by our preconditions, which concludes our proof.
References


