Burn Your Vote: Decentralized and Publicly Verifiable Anonymous Voting at Scale

Stefan Dziembowski
University of Warsaw
stefan.dziembowski@crypto.edu.pl

Shahriar Ebrahimi*
Zero Savvy
sh.ebrahimi92@gmail.com

Haniyeh Habibi
University of Warsaw
h.habibi@uw.edu.pl

Parisa Hassanizadeh*
IPPT PAN / Zero Savvy
parisaa.hassanizadeh@gmail.com

Pardis Toolabi
University of Warsaw
p.toolabi@uw.edu.pl

Abstract

Secure and trustworthy electronic voting requires more than correctness and censorship resistance, it must also ensure voter privacy, vote confidentiality, and protection against coercion. Prior work attempt to address these challenges using heavyweight cryptographic primitives such as homomorphic encryption, time-lock puzzles, or multi-party computation. These approaches often involve complex computations, depend on trusted parties, and typically do not scale well. We propose a lightweight, fully on-chain anonymous voting protocol based on a novel application of the *proof-of-burn* (PoB) mechanism. Voters anonymously commit to their votes by burning tokens to pseudorandom addresses and later submit zero-knowledge proofs attesting to their valid participation. Our design achieves vote integrity, coercion resistance, and unlinkability without relying on encrypted ballots, trusted third parties, or centralized tallying. The tallying process is public and operates on plaintext votes that are authenticated yet unlinkable to voters. This enables flexible voting models—including token-weighted and quadratic voting—with minimal on-chain overhead. We formally analyze the protocol's security guarantees and demonstrate support for a broad range of voting models. We implement the protocol as an open-source library fully compatible with the Ethereum Virtual Machine (EVM), and our experimental evaluation confirms its high scalability and improved efficiency compared to the state-of-the-art.

1 Introduction

Blockchains have fundamentally transformed our understanding of decentralization, enabling a wide range of decentralized applications, such as decentralized exchanges [8, 45] in the financial domain. Electronic Voting was one of the earliest and most promising applications of blockchain technology. While blockchains naturally provide correct execution -guaranteeing that smart contracts behave exactly as coded- and censorship resistance, these features alone are not sufficient for trustless and privacy-preserving voting systems [13].

Anonymous voting enables participants to express their genuine preferences freely, without fear of retaliation, coercion [33], or social pressure risks often present in centralized voting systems and electronic voting systems lacking robust privacy guarantees. The motivation for this type of voting extends beyond simple confidentiality. By preventing any link between a voter and their ballot, voting schemes undermine coercion and collusion [28]. Preserving the privacy of the voters also encourages honest participation, especially in sensitive decision-making in decentralized governance settings such as DAOs and digital political platforms. Despite the advances, existing systems continue to face challenges in simultaneously achieving privacy, transparency, scalability, and trustlessness. These properties often impose conflicting requirements—for example, enhancing privacy can obscure auditability, while eliminating trusted intermediaries can complicate coordination and scalability.

A privacy-preserving voting system should fulfill several foundational security properties. **Voter anonymity** ensures that the identity of each voter remains hidden throughout and after the election, preventing the association between voters and their choices. **Coercion resistance** protects voters from external pressure by ensuring that even if they are coerced, they cannot convincingly prove how they voted, thereby disincentivizing vote-buying and intimidation [2, 9, 19]. **Decentralized tallying** eliminates the need for trusted third parties by enabling the outcome to be computed transparently and autonomously, reducing the risk of manipulation or bias. Finally, **public verifiability** guarantees that any observer can independently audit the entire process, promoting transparency and trust in the result.

Recent studies have explored several cryptographic techniques, such as fully homomorphic encryption (FHE), multi-party computation (MPC), and zero-knowledge proofs (ZKPs), to achieve privacy-preserving voting. FHE-based systems support tallying over encrypted data [16, 18, 24], but restrict the type of votes they can process—typically only supporting numerical values and not more complex inputs such as strings. Moreover, many of these systems rely on MPC for the decryption phase [5, 11], introducing significant computational overhead and requiring complex coordination among multiple parties, or alternatively assume a trusted decryption authority. These limitations hinder their scalability and practicality for large-scale or fully decentralized deployments. For instance, MACI [4] proposes an infrastructure for on-chain voting. It aims to resist vote collusion through zkSNARKs and asymmetric encryption, but relies on a trusted coordinator who can view all individual submissions [4]. Cicada [25] uses homomorphic time-lock puzzles (HTLP) [37] for non-interactive vote submission, but reveals all votes post-tally and encounters scalability issues due to complexity of tallying. Chaintegrity [46] supports scalability and verifiability through blind signatures [14] and threshold decryption [41], but relies on authorities for tallying. Kite [40] enables delegation in DAOs with encrypted vote aggregation using Additive

*The author was affiliated with IDEAS NCBR partially during the work on this paper.

Homomorphic Encryption (AHE), but uses a trusted tallying authority, limiting transparency. These examples illustrate that while existing solutions address some privacy concerns, they frequently introduce centralization or compromise coercion resistance.

1.1 Our Approach

Considering the limitations of existing blockchain-based voting systems, we propose a fully on-chain, trustless, and privacy-preserving voting protocol that avoids the computational costs of techniques like FHE and MPC, making our solution more scalable and practical for real-world deployment. Our protocol operates without any trusted parties or off-chain tallying, relying solely on smart contracts for all verification and aggregation steps.

At the core of our protocol is a novel use of the *proof-of-burn* [34] mechanism: voters irreversibly burn tokens on-chain and generate a ZKP attesting to a valid burn. The protocol is designed to be highly flexible, supporting different participation workflows and decoupling registration from voting when needed. To prevent double voting, each submission includes a nullifier derived from private data [20, 30, 43], which the smart contract checks for uniqueness. Votes are submitted in plaintext alongside the proof, enabling public tallying and verifiability, while unlinkability between voters and votes is preserved through fresh burn addresses and optional relayers (as in Semaphore [31]) or meta-transactions (e.g., ERC-4337 paymasters [22]).

One of the key features of our protocol is its natural coercion resistance. Since the proof-of-burn mechanism allows each voter to perform multiple independent burns to random, unlinkable addresses, voters can commit to different votes without revealing any association to their identity or previous actions. This enables them to plausibly deny any coerced vote and later cast their true preference. Optional features such as revoting allow voters to overwrite previously submitted ballots, further enhancing coercion resistance and improving usability. The protocol benefits from easy vote delegation, since certain proofs-of-burn can be sent to a trusted representative to cast the vote on their behalf. Furthermore, the public and transparent tallying process supports a broad range of voting models, including majority, token-weighted, quadratic [35, 42], and ranked-choice voting, all implemented through configurable on-chain logic.

We have implemented the proposed protocol in a publicly available open-source library[1], and evaluated its practicality and scalability on EVM-compatible blockchains. For ZKP generation, we use the Circom [7] domain-specific language to define the constraint system and generate Groth16 [29] proofs. While our implementation employs Groth16, the protocol is compatible with alternative ZKP schemes. All experiments were conducted on commodity hardware (e.g., a MacBook Air) to demonstrate the feasibility of our approach in real-world settings. We provide a detailed analysis of gas consumption across all smart contract functionalities and compare the results with the state-of-the-art voting protocols.

The contributions of this paper are:

- **Burn-to-Vote Protocol:** To the best of our knowledge, we are the first to use the proof-of-burn mechanism for on-chain voting. The protocol is designed to be highly flexible, supporting different participation workflows and offers following features:
  - **Minimum Trust Assumptions:** Eliminates the need for any coordinator or trusted tallying authority, a smart contract is in charge of tallying.
  - **Public Verifiability:** Auditing the result of the tallying is open to everyone at anytime.
  - **Privacy-Preserving Mechanism:** There is no connection between vote value and voter identity, w.r.t using Proof-of-brun in the proposed protocol.
  - **Coercion Resistance:** Voters can issue multiple unlinkable burn transactions from different sources and with varying vote values, allowing plausible deniability under coercion. Additionally, the protocol supports optional revoting during the submission phase, enabling voters to overwrite previously submitted ballots.
  - **Vote Delegation:** Voters can delegate their vote at no cost to a trusted representative by sending them their proofs for *burn-to-register* mode (Section 3).
  - **Support for Diverse Voting Schemes:** Since vote values are submitted in plaintext, the protocol can support any tallying logic expressible in Solidity. This enables flexible and modular implementation of a wide range of voting models, including token-weighted, equal-weight, quadratic, ranked-choice, and even generalized models such as open-content (arbitrary data) voting. To the best of our knowledge, we are the first to support such expressive, free-form voting.
- **Security Analysis:** We provide formal security proofs by reducing the soundness of our construction to the security of the underlying cryptographic primitives. Additionally, we analyze the protocol's guarantees in terms of coercion resistance, public verifiability, double-voting prevention, and discuss its robustness against common attacks and adversarial strategies.
- **Proof of Concept:** We implement and evaluate our constructions on public EVM-compatible blockchains using only commodity hardware (e.g., a MacBook Air). The results demonstrate the protocol's practicality and scalability, confirming its feasibility for real-world deployment without requiring specialized infrastructure.

[1]https://github.com/zero-savvy/burn-to-vote

Note on Token Burning

While burning tokens may initially seem costly for users, it is in fact cheaper than the registration phases required by many alternative protocols, which involve smart contract interactions and higher gas costs. In contrast, a burn transaction is a simple token transfer to an unspendable address, incurring minimal fees. Moreover, the burn amount can be trivially small (e.g., 0.000001 ETH), often costing less than a cent. Beyond efficiency, token burning can be economically beneficial: by permanently removing tokens from circulation, it contributes to deflationary pressure. Moreover, recent Ethereum EIPs such as EIP-7503 [3] further explore the potential of reclaimable burns. It is worth noting that the protocol supports all types of blockchain assets (native or ERC-20 tokens or NFTs to limit number of possible burns by one user).

Table 1 summarizes the properties of our protocol compared to related work, highlighting how the proposed design balances privacy, scalability, and decentralized verifiability with minimum trust assumptions and without costly computations.

The remainder of this paper is organized as follows. Section 2 reviews the cryptographic foundations underlying our protocol. Section 3 details the protocol design. Section 4 presents the security and soundness analysis. Section 5 discusses practical applications and deployment scenarios. Section 6 outlines implementation details and reports experimental results. Section 7 reviwes the related works, and Section 8 concludes the paper and sketches the directions for future research.

2 Background

This section introduces the core cryptographic building blocks used in our protocol, including commitment schemes, zero-knowledge proofs, and Merkle proofs. We also define the proof-of-burn concept. In addition, we define the notation used throughout the paper in Table 2.

2.1 Commitment Schemes

Commitment schemes are cryptographic protocols that allow one party to commit to a value while keeping it hidden, with the ability to reveal the value later in a verifiable manner. A commitment scheme is a triple (Gen, Commit, Open) such that:

- $pp \leftarrow Gen(1^\lambda)$
- for any $m \in M, (c,d) \leftarrow Commit_{pp}(m)$, which c is the commitment value, and d is the opening value.
- $m' \leftarrow Open_{pp}(c,d), m' \in M \cup \{\perp\}$, which $\perp$ is the result in case of the invalid commitment to any message.

A commitment scheme should satisfy two key security properties[17]:

(1) **Hiding:** The commitment should conceal the committed value so that no information about it is leaked prior to the opening. Here it should be computationaly hard for the adversary $\mathcal{A}$, to generate two messages, $m_0, m_1 \in M$, such that $\mathcal{A}$ can distinguish between their correspondig commitment values, c_0, c_1.

For any PPT $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$:

$$\left| \frac{1}{2} - \Pr\left[b = b' \;\middle|\; \begin{array}{c} pp \leftarrow \text{Gen}(1^\lambda) \\ (m_0, m_1, st) \leftarrow \mathcal{A}_0(pp) \\ b \leftarrow\!\!\$\ \mathbb{Z}_2, \\ (c,d) \leftarrow \text{Commit}_{pp}(m_b) \\ b' \leftarrow \mathcal{A}_1(c, st) \end{array} \right] \right| = \text{negl}(\lambda)$$

(2) **Binding:** Once the prover has committed to a value, the prover cannot change the value. It is computationally hard for the adversary $\mathcal{A}$ to build (c,d,d'), such that (c,d) and (c,d') are valid commitments for m_0 and m_1, which $m_0 \neq m_1$.

For any PPT $\mathcal{A}$:

$$\Pr\left[\begin{array}{c} m_0 \neq m_1 \\ m_0, m_1 \neq \perp \end{array} \;\middle|\; \begin{array}{c} pp \leftarrow \text{Gen}(1^\lambda) \\ (c,d,d') \leftarrow \mathcal{A}(pp) \\ m_0 \leftarrow \text{Open}_{pp}(c,d) \\ m_1 \leftarrow \text{Open}_{pp}(c,d') \end{array} \right] = \text{negl}(\lambda)$$

As a basic illustrative example of commitment schemes, consider a scenario where the prover P possesses an RSA key pair, and the verifier V knows the corresponding public key, consisting of modulus n and public exponent e. To commit to a bit b, the prover selects a random value x_b modulo n, ensuring that its least significant bit encodes the value b. The commitment is then computed as $C = x_b^e \mod n$, which is sent to V. It can be intuitively understood that the commitment is both binding —since C uniquely determines x_b, and therefore b—and hiding, provided that the RSA encryption remains secure and inverting it is computationally infeasible[17].

2.2 zkSNARKs

Zero Knowledge Succinct Non-interactive Arguments of Knowledge (zkSNARKs) are cryptographic proof systems that allow the prover to convince the verifier that a statement is true without revealing any additional information beyond the validity of the claim[10, 26]. zkSNARKs are widely adopted in privacy-preserving applications due to their succinctness, non-interactivity, and strong soundness guarantees.

Here, we begin by formally defining SNARKs, highlighting their core principles. We then extend this framework to define IVC schemes as a specialized subclass and indicate their key distinctions.

Definition 2.1. Let $\mathcal{R}$ be a binary relation corresponding to a non-polynomial (NP) language $L_\mathcal{R}$ with λ as the security parameter. We define an argument system for $\mathcal{R}$ as a tuple of PPT algorithms $\Pi = (\mathcal{G}, \mathcal{P}, \mathcal{V}, \mathcal{S})$ as follows:

$pp \leftarrow \mathcal{G}(1^\lambda)$**:** The generator creating the public parameters pp with respect to λ.

$(pk, vk) \leftarrow \mathcal{K}(pp, s)$**:** A deterministic encoding algorithm that derives a proving and verifying key pair (pk, vk) for a given statement s with respect to pp.

$\pi \leftarrow \mathcal{P}(pk, u, w)$**:** The prover algorithm for generating a proof π for the specific statement s and the witness w. Formally, proving that $\exists\, w\ s.t.\ (pp, s, u, w) \in \mathcal{R}$.

$b \leftarrow \mathcal{V}(vk, u, \pi)$**:** The verifier algorithm for checking the validity of π and returning $b \in \{0, 1\}$.

Table 1: Comparison of our protocol with prior voting systems.

Protocol	Voting Scenarios	Assumptions	Primitives	Coordinator Tallier	Voter Anonymity	Vote Confidentiality	Coercion Resistance	Verifiability	Re-vote Support
Cicada [25]	Multiple ✓	Public Blockchain ✓	HTLP, MPC*, ZKP	Smart contract ✓	✓	Hidden until Tally ✓	✓	Public ✓	✓
Kite [40]	Limited ✗	Trusted Authority ✗	AHE, ZKP	Trusted authority ✗	✓*	TA-based ✓*	✗	TA-based ✓*	✓*
PPVSS [6]	Additive ✓*	Honest Majority ✓*	PVSS, MPC, ZKP	Talliers (off-chain) ✓*	✗	Encrypted shares ✓	✗	Public ✓	✗
Chaintegrity [46]	Multiple ✓	Honest Maj., Pub. chain ✓*	HE, MPC, ZKP	Threshold talliers ✓	✓	Encrypted ballots ✓	✗	Public ✓	✗
This Work	Multiple ✓	Public Blockchain ✓	ZKP	Smart contract ✓	✓	Hidden until Tally* ✓	✓	Public ✓	✓

✓* Partially supported or with limitations. TA Trusted Authority. * Technically, votes are visible during submission, but this occurs after the voting period has ended as part of the tally phase.

Table 2: Terminology of This work

Notation	Description
H	Poseidon [27] hash function.
Λ	Burn address.
a	Token burn amount.
v, w	Vote value and weight.
v_{id}	Voting session identifier.
u_{id}	user identifier.
$\langle sk, pk\rangle$	secret and public key pair.
l	The leaf of the allow list merkle tree.
r_M	The root hash of a Merkle tree.
s_r	Merkle Patricia root: the root hash of the Ethereum's state Merkle Patricia Trie (MPT) in the last block of the voting.
k_u	The remaining unique nibbles of the address hash after traversing state trie.
key_b	The state trie leaf key.
$value_b$	The state trie leaf value .
rlp_Λ	The RLP encoding of address Λ. rlp = RLP(nonce, balance, storageRoot, codeHash).
h_i	The hash of the i-th node in the state trie.
β	Random blinding factor.
η	Nullifier to prevent double voting.
π	zkSNARKs proof

$\pi \leftarrow \mathcal{S}(pp, u, \tau)$: The simulator that produces a proof π given a trapdoor τ.

We can label any argument system Π as a SNARK if it satisfies the following properties:

- **Completeness**: An honest prover with valid witness should convince any verifier. Formally, for any PPT adversary $\mathcal{A}$:

$$\Pr\left[\mathcal{V}(vk, u, \pi) = 1 \;\middle|\; \begin{array}{c} pp \leftarrow \mathcal{G}(1^\lambda) \\ (s, (u, w))) \leftarrow \mathcal{A}(pp) \\ (pp, s, u, w) \in \mathcal{R} \\ (pk, vk) \leftarrow \mathcal{K}(pp, s) \\ \pi \leftarrow \mathcal{P}(pk, u, w) \end{array}\right] = 1$$

- **Knowledge Soundness**: A dishonest prover, should not be able to convince any verifier. To formally define this we require that for all PPT adversaries $\mathcal{A}$ there exists an extractor $\mathcal{E}$ that can compute witness given any randomness ρ, such that:

$$\Pr\left[\begin{array}{c} \mathcal{V}(vk, u, \pi) = 1, \\ (pp, s, u, w) \notin \mathcal{R} \end{array} \;\middle|\; \begin{array}{c} pp \leftarrow \mathcal{G}(1^\lambda) \\ (s, (u, w)) \leftarrow \mathcal{A}(pp) \\ (pk, vk) \leftarrow \mathcal{K}(pp, s) \\ w \leftarrow \mathcal{E}(pp, \rho) \end{array}\right] = \mathrm{negl}(\lambda)$$

- **[Optional] Zero-knowledge**: The argument dos not reveal anything beyond the truth of the statement. There must exist a simulator $\mathcal{S}$ such that for all PPT adversaries $\mathcal{A}$ following distributions are indistinguishable:

$$\left\{ (pp, s, u, \pi) \;\middle|\; \begin{array}{l} pp \leftarrow \mathcal{G}(1^\lambda) \\ (s, (u, w))) \leftarrow \mathcal{A}(pp) \\ (pp, s, u, w) \in \mathcal{R} \\ (pk, vk) \leftarrow \mathcal{K}(pp, s) \\ \pi \leftarrow \mathcal{P}(pk, u, w) \end{array} \right\}$$

$$\approx$$

$$\left\{ (pp, s, u, \pi) \;\middle|\; \begin{array}{l} (pp, \rho) \leftarrow \mathcal{S}(1^\lambda) \\ (s, (u, w))) \leftarrow \mathcal{A}(pp) \\ (pp, s, u, w) \in \mathcal{R} \\ (pk, vk) \leftarrow \mathcal{K}(pp, s) \\ \pi \leftarrow \mathcal{S}(pp, u, \rho) \end{array} \right\}$$

2.3 Proof-of-Burn

Proof-of-Burn (PoB) is a cryptographic mechanism that enables the verifiable destruction of cryptocurrency tokens. In practice, this is achieved by sending tokens to a provably unspendable address. Originally introduced in the context of blockchain systems [34], PoB has been applied in various settings, including anonymous transaction pools and trustless bridges between blockchains. The similar ideas for creating burn address is also proposed in EIP-7503[3].

To formally define PoB, we follow the construction introduced by [34]. The core idea is that a user generates a special address—referred to as a *burn address*—which functions as a commitment to some user-defined metadata, called a *tag*. The user then transfers an arbitrary amount of cryptocurrency to this address, thereby irreversibly removing it from circulation. Afterward, the user can prove to any verifier that the burn occurred and that it is associated with a particular tag.

A secure PoB scheme must satisfy the following properties:

- **Unspendability:** No one should be able to spend the tokens sent to the burn address.

- **Binding:** The burn address must be uniquely tied to a particular tag, preventing a user from claiming multiple interpretations.
- **Hiding/Uncensorability:** In the context of PoB, the hiding property also implies uncensorability. Specifically, a burn transaction should be indistinguishable from a regular transaction. This requires that the burn address be computationally indistinguishable from standard addresses, ensuring that the transaction cannot be identified or censored as a burn until the user explicitly reveals the tag associated with the address.

We now formalize PoB with respect to a generic secure commitment scheme C:

Definition 2.2. Let pp be the security parameter. A proof-of-burn system consists of the following algorithms:

GenBurnAddr$(f, t) \rightarrow \Lambda$: Generates a burn address Λ using a function f and input tag t. The address must be unspendable and serve as a commitment to $f(t)$. Formally, we can write GenBurnAddr$(f, t) = \text{Commit}_{pp}(f(t), r)$, where r is a random value used as a hiding factor.

VerifyBurnAddr$(\Lambda, f, t) \rightarrow b$: Verifies that the burn address Λ corresponds to the tag t under function f. This can be expressed as
VerifyBurnAddr$(\Lambda, f, t) = \text{Verify}_{pp}(\Lambda, \text{Open}_{pp}(\Lambda, f(t), r))$, returning $b \in \mathbb{B}$ as the result.

3 Voting Protocol

We begin this section by presenting the overview of the protocol, then we describe the details of the burning mechanism and proof generation. Finally, we discuss the procedures for vote submission, vote verification and tallying. Note that the vote submission process is part of the tallying phase, as the contract assesses the submitted proofs and counts the votes upon receiving each vote transaction.

3.1 Overview

Figure 1 illustrates a high-level overview of the protocol. The protocol is divided into three main stages: the *setup phase*, the *voting phase* (which includes token burning and proof generation), and the *tallying phase*.

- **Setup Phase:** In this phase, the organizer defines the voting parameters, such as the available options to vote for, voting duration, and the statement for the proof-of-burn circuit. The organizer may either deploy a new contract or configure an existing one.
- **Voting Phase:** Each voter locally computes a unique, unspendable burn address and sends specific amount of tokens to this unspendable address. The voter then generates a proof of burn to submit it along with their vote to the smart contract.
- **Tallying Phase:** Tallying is performed entirely on-chain. The smart contract verifies the submitted proof and the validity of the vote, then updates the tally accordingly.

Anonymity Pool. A key advantage of the proposed voting mechanism lies in the size of its anonymity pool. Unlike the rest of the

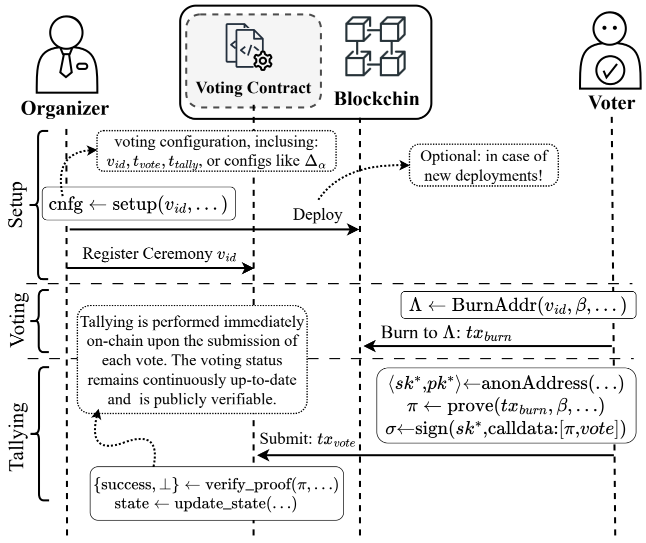
Organizer
Voting Contract
Blockchin
Voter
Setup
Voting
Tallying
voting configuration, inclusing: $v_{id}, t_{vote}, t_{tally}$, or configs like Δ_α
Optional: in case of new deployments!
cnfg ← setup($v_{id}, \ldots$)
Deploy
Register Ceremony v_{id}
Λ ← BurnAddr($v_{id}, \beta, \ldots$)
Burn to Λ: tx_{burn}
Tallying is performed immediately on-chain upon the submission of each vote. The voting status remains continuously up-to-date and is publicly verifiable.
⟨sk^*,pk^*⟩←anonAddress(…)
π ← prove($tx_{burn}, \beta, \ldots$)
σ←sign(sk^*,calldata:[π,$vote$])
Submit: tx_{vote}
{success, ⊥} ← verify_proof(π, …)
state ← update_state(…)

Figure 1: Protocol Overview

approaches [25, 46] that restrict anonymity to a fixed set of pre-registered voters, our mechanism enables an anonymity set that encompasses all externally owned accounts (EOAs) involved in any token transfer during the voting period. Even in the case of the burn-to-register scheme, the actual number of participating voters remains hidden, as registration occurs through indistinguishable burn transactions. This design significantly amplifies voter privacy and resists attempts to infer voter participation.

An important feature of the proposed protocol is its inherent support for preserving voter's privacy or registration submissions, even through aggregated transactions. Voters are not required to interact directly with the blockchain; instead, they may delegate the transaction to a relayer. For example, a group of voters can collectively submit their burn addresses to a relayer, which then performs the token transfers on their behalf. This process preserves anonymity, as the relayer cannot link the addresses to individual voters. Such flexibility is particularly useful in large-scale voting scenarios, where many participants may rely on centralized exchanges or custodial wallets to initiate the burn transaction without compromising their privacy.

3.2 Setup Phase

During the setup phase, the organizer either deploys a new smart contract or configures an existing one to support multiple instances of similar voting ceremonies. The organizer is responsible for specifying the parameters that define the structure and constraints of the voting instance. These parameters include:

- **Voting ID:** A unique identifier assigned to the specific voting instance.
- **Voting Options:** The set of choices available for voters to select from.
- **Voting Period Duration:** Defines the timing parameters for the voting process, including the start and end block

numbers for the voting phase, and the start and end timestamps for the submission (tallying) phase.

- **Proof-of-Burn Statement:** A formal statement that defines the logic and rules embedded in the proof-of-burn circuit. This statement determines the structure of the zero-knowledge proof, as well as the associated proving and verification keys.

3.3 Voting Phase

We define two distinct burning mechanisms that can be employed during the voting phase, each serving different purposes and offering flexibility in how votes are cast. In general, depending on the voting ceremony conditions, voters may choose to burn tokens either solely to register their participation in the voting ceremony (*Burn-to-Register*) or to simultaneously commit to a specific vote (*Burn-to-Vote*). In the following, we describe both mechanisms in detail.

(1) **Burn-to-Vote:** In this mechanism, the vote itself is incorporated into the burn address computation, effectively committing the voter to their chosen vote at the time of burning. As discussed in Section 4.3, this approach enables voters to cast multiple votes that are unlinkable from one another, thereby providing strong coercion resistance. The burn address Λ is computed as follows:

$$\Lambda = H(u_{id} \| v_{id} \| v \| \beta),$$

where:

- u_{id} is the voter's unique public identifier (e.g., wallet address or public key), which optionally binds the burn address to a specific voter. This can be useful in scenarios that involve allowlists or pre-registered voter sets.
- v_{id} is the identifier of the voting ceremony. It ensures that the burn address is specific to the voting instance. This value is exposed as a public input in the proof-of-burn circuit.
- v is the actual vote value and is also exposed as a public input to the proof-of-burn circuit to link the commitment with the submitted vote.
- β is a random blinding factor that ensures the hiding property of the commitment. It is also later used to generate a nullifier that prevents double voting.

We note that, depending on the application, some of these parameters may be omitted or replaced with null values. For instance, if voter eligibility is not restricted, u_{id} may be set to a null or default value.

(2) **Burn-to-Register:** In this mechanism, the user's vote is not included in the burn address computation. Instead, the voter burns tokens prior to the start of the voting period to register their eligibility for participation in the voting ceremony. This approach may also serve economic purposes in certain scenarios where users are required to burn a predefined amount of a specific token, thereby reducing the circulating supply. Such deflationary behavior can, in some ecosystems, positively influence the market value of the token. The burn address Λ is computed as follows:

$$\Lambda = H(u_{id} \| v_{id} \| \beta)$$

where, u_{id}, v_{id} and β are as defined in the previous mechanism.

3.3.1 *Calculation of Nullifier.*

The nullifier is a critical component of the protocol, ensuring that a single burn transaction cannot be reused for multiple votes, thereby preventing double voting with the same burn address while preserving voter anonymity. To achieve this, the nullifier must be uniquely tied to the user, the voting ceremony, and the random blinding factor β. At the same time, it must not reveal any information about, or linkage to, the corresponding burn address. A secure construction that satisfies these requirements is given by:

$$\eta = H(u_{id} \parallel v_{id} \parallel H(\beta)),$$

where u_{id} is the voter's unique identifier (e.g., a public key or wallet address), v_{id} is the identifier of the voting ceremony, and β is a user-selected blinding factor. The use of the inner hash $H(\beta)$ ensures that the nullifier is deterministically derived from the same blinding factor used in burn address generation (provable within ZK circuit), yet produces an unlinkable pseudorandom value. This construction guarantees that nullifiers do not reveal any information about, or linkage to, the corresponding burn address.

3.3.2 *Proof Generation.*

After burning the tokens, the user generates a zkSNARK proof for the burn transaction to demonstrate the following properties without revealing any private information:

- **Correct Burn Address:** The user knows a valid burn address Λ, which is computed according to the rules specified by the chosen burning mechanism—either *burn-to-vote* or *burn-to-register*.
- **Valid Token Burn:** The user proves that a valid amount of tokens a has been transferred to the burn address Λ, with $a \in [a_{\min}, a_{\max}]$. Depending on the underlying blockchain architecture, this condition is formalized as follows:
  - **General Blockchains:** There exists a transaction that sends an acceptable amount of tokens to Λ.
  - **Account-Based Blockchains:** The burn address Λ is associated with a balance in the range $[a_{\min}, a_{\max}]$ and is provably included in the Merkle Patricia Trie (MPT) of a block finalized within the voting period.
- **Correct Nullifier Derivation:** The nullifier is correctly computed from the same blinding factor β used in the generation of Λ, thereby linking the nullifier to the burn commitment without leaking identifying information.
- **Vote Binding (Burn-to-Vote only):** For burn-to-vote, the submitted vote v is valid according to the rules of the voting scheme and is correctly embedded in the burn address Λ.

In the following, we formally define the statement that a voter must prove before submitting to the smart contract. We assume the underlying blockchain is an account-based chain, such as Ethereum. To ensure compatibility with common account-based chains, we adopt Ethereum's exact block standards. The definition below addresses the *burn-to-vote* scenario. We omit the *burn-to-register* definition as it is analogous, differing only in the construction of the burn address Λ.

Definition 3.1. Let β be the random blinding factor, u_{id} the voter's unique identifier, v_{id} the voting ceremony identifier, and v the vote value. Let `s_r` denote the state root of the MPT in the last finalized block within the voting period. We denote by π_Λ the MPT authentication path from the leaf to the root corresponding to the burn address Λ (including all intermediate nodes). The prover claims knowledge of values satisfying the following statement:

$$S\big[\mathtt{s\_r}, \eta, v_{id}, v, a_{\min}, a_{\max}\big] : \Big\{\exists\, \Lambda, \pi_\Lambda, \mathtt{rlp}_\Lambda, \beta, u_{id}, a \ \ s.t.$$
$$\mathtt{rlp}_\Lambda = \mathsf{RLP}(0, a, \phi, \phi) \ \wedge \ \Lambda = H(u_{id} \| v_{id} \| v \| \beta)$$
$$\wedge\ a \in [a_{\min}, a_{\max}] \ \wedge \ \eta = H(u_{id} \| v_{id} \| H(\beta))$$
$$\wedge\ \mathcal{V}_{MPT}(\mathtt{s\_r}, \pi_\Lambda, \Lambda, \mathtt{rlp}_\Lambda) = 1\Big\}$$

We analyze the soundness of this proof statement in Theorem 2. Section 6.1 details the construction of the RLP encoding, the MPT verifier $\mathcal{V}_{\mathsf{MPT}}$, and the implementation of each step of the statement using the `Circom` DSL. An overview of the top-level `Circom` circuit is provided in Appendix B.

3.4 Tally

After the voting (or registration) phase concludes, users submit their votes along with the corresponding burn proofs to the smart contract for verification.

3.4.1 ***Vote Submission.*** The vote submission includes the following components:

- The vote/revote value (v).
- The burn proof (π_{burn}), demonstrating that the user has correctly burned the required tokens.
- Public inputs, such as the nullifier, voting ID and state trie root, to ensure verifiability.

The smart contract performs the following checks upon receiving a vote submission:

(1) Ensures the nullifier has not been previously used to prevent double voting (except when revoting is enabled) and records it as spent.
(2) Calls the verifier contract to validate the correctness of the burn proof and related cryptographic commitments (see Definition 3.1 for details).
(3) Confirms the vote value (v) complies with the voting scheme rules.
(4) If the `revote` flag is enabled, allows vote replacement by reusing the nullifier and overwriting the previous vote.
(5) Verifies that the vote is submitted within the allowed voting period.
(6) Validates the public inputs against predefined state variables.

To preserve voter anonymity, the protocol employs the following privacy-enhancing mechanisms:

- **Decentralized Relayers:** Relayers receive vote transactions off-chain and broadcast them on-chain on behalf of voters. This decouples the transaction origin from the voter's identity, preventing linkage between the voter's wallet and the vote submission. [31]
- **Fresh Addresses:** Vote submissions use fresh addresses with no prior transaction history to prevent linkage to voters' primary identities. This is achieved by either deploying ERC-4337 [22] smart contract wallets with paymasters covering gas fees, or by creating EOAs funded through privacy mixers (e.g., Tornado Cash [43]). Both methods ensure the voting address is unlinkable to previous activity, maintaining anonymity.

3.4.2 ***Vote Calculation.*** The tally mechanism aggregates votes according to the rules of the selected voting scheme. Depending on the scheme, votes may be simple binary values, weighted by tokens burned, or follow more complex structures.

The smart contract applies the relevant tally logic accordingly, ensuring flexibility across diverse voting models. For example, in majority voting (First-Past-the-Post), votes are binary and equally weighted. The outcome is determined by whether the number of affirmative votes exceeds half of the total votes.

For detailed descriptions of supported voting types and their tally methods, see Section 5 and related subsections.

4 Soundness and Security

In this section, we will show that the proposed voting protocol is sound and secure. for this, we first discuss the system and adversary model, then we will show the soundness and security of the protocol w.r.t. the adversary model.

4.1 System and Adversary Model

We consider a standard adversarial model in which the adversary is a probabilistic polynomial-time (PPT) entity. The adversary is assumed to have full control over the communication network, allowing it to observe, intercept, modify, or inject arbitrary messages. Despite this capability, we assume the following cryptographic primitives remain secure against any PPT adversary:

- **Collision-Resistant Hash Functions:** Hash functions used in the protocol are assumed to be collision-resistant, meaning it is computationally infeasible to find two distinct inputs that hash to the same output.
- **Secure Digital Signatures:** The protocol assumes that the underlying digital signature scheme is unforgeable under chosen-message attacks.
- **Succinct Zero-Knowledge Proof Systems:** Our protocol is compatible with any zkSNARK scheme. In our implementation, we employ Groth16 [29], assuming it is computationally infeasible to produce a valid proof for a false statement.
- **Blockchain Assumptions:** We assume a public, permissionless blockchain that provides standard guarantees, such as ledger immutability, consensus-based state agreement, and trustless, deterministic execution of smart contracts, which is publicly verifiable. Furthermore, malformed or adversarial blocks are assumed to be rejected by the consensus protocol, preserving consistency and integrity.

4.2 Soundness

Theorem 1. *The proposed method of generating a burn address using a collision-resistant hash function serves as a hash-based commitment scheme, satisfying both the* hiding *and* binding *properties.*

Proof sketch. In the proposed method, the burn address is computed as: $\Lambda = H(u_{id} \| v_{id} \| \beta\, v)$, where s is the voter's secret key, v_{id} is the ceremony identifier, β is a salt or randomizer, and v is the vote. Suppose an adversary $\mathcal{A}$ can predict the exact value of v (i.e., the committed vote) with probability $p = \frac{1}{2} + \epsilon$. We define the preimage of Λ as: $x = u_{id} \| v_{id} \| \beta \| v$. We consider the standard hiding experiment:

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} pp \leftarrow \mathrm{Gen}(1^{\lambda}) \\ (v_0, v_1, st) \leftarrow \mathcal{A}_0(pp) \\ b \leftarrow\!\!\$\; \mathbb{Z}_2, r \leftarrow\!\!\$\; \mathbb{Z}_q \\ \Lambda \leftarrow H(u_{id} \| v_{id} \| \beta \| v_b) \\ b' \leftarrow \mathcal{A}_1(st, \Lambda) \end{array} \right] = \frac{1}{2} + \epsilon$$

If ϵ is non-negligible, then $\mathcal{A}$ gains partial information about the committed value v_b, which is a component of x. Denote x_i as the bit of x corresponding to v_b. If $\mathcal{A}$ can distinguish x_i with advantage $\epsilon \gg \mathrm{negl}(1^{\lambda})$, then we can use a hybrid argument to recover x bit by bit:

- Fix a candidate $x' \in \{0,1\}^n$.
- For each bit $j \in \{1, \ldots, n\}$, construct two variants $x^{(0)}$ and $x^{(1)}$ such that they differ only at bit j.
- Compute $y^{(0)} = H(x^{(0)})$ and $y^{(1)} = H(x^{(1)})$.
- Run $\mathcal{A}$ on both outputs. If $\mathcal{A}$ distinguishes between $y^{(0)}$ and $y^{(1)}$ with non-negligible probability, then it leaks information about bit j.

By applying this bit-recovery process iteratively and using majority decoding over repeated queries, the full preimage x can be reconstructed. This contradicts the preimage resistance of the hash function H. Therefore, under the assumption that H is collision-resistant and preimage-resistant, the proposed burn address generation satisfies the *hiding* property.

The *binding* property is guaranteed by the collision resistance of the hash function H: it must be computationally infeasible to find two distinct inputs that produce the same burn address. To formalize this, we define the following experiment executed by $\mathcal{A}$:

$$\Pr\left[\begin{array}{c} v_0 \neq v_1 \\ \wedge\, v_0, v_1 \neq \perp \end{array} \;\middle|\; \begin{array}{c} pp \leftarrow \mathrm{Gen}(1^{\lambda}) \\ (\Lambda, v_0, v_1, \beta_0, \beta_1) \leftarrow A(pp) \\ v_0 \leftarrow \mathrm{Open}_{pp}(\Lambda, v_0, \beta_0) \\ v_1 \leftarrow \mathrm{Open}_{pp}(\Lambda, v_1, \beta_1) \end{array} \right] = \epsilon$$

This captures the probability that $\mathcal{A}$ can find two distinct openings $(v_0, \beta_0) \neq (v_1, \beta_1)$ such that both are accepted as valid openings of the same burn address Λ. If $\mathcal{A}$ succeeds with non-negligible probability, it would imply that: $H(u_{id} \parallel v_{id} \parallel \beta_0 \parallel v_0) = H(u_{id} \parallel v_{id} \parallel \beta_1 \parallel v_1)$. Therefore, under our assumptions, the adversary's advantage ϵ must be negligible.

Hence, the scheme achieves both *hiding* and *binding*, and acts as a sound commitment mechanism. □

Theorem 2. *The probability that a PPT adversary successfully breaking the soundness of the statement in Definition 3.1 is negligible.*

Proof sketch. To break the soundness of the statement in Definition 3.1, an adversary $\mathcal{P}^*$ must succeed in one of the following cases: (1) The adversary forges a valid proof of burn π_{Λ} for a false statement. This means that some constraints of the statement are not satisfied; for example, the computations of $\texttt{rlp}_{\Lambda}$, Λ, or η do not conform to the statement's constraints, or the burn address Λ is not included in the MPT corresponding to the state root hash `s_r`. However, the adversary still produces a proof that passes verification. This implies breaking the soundness of the underlying proof system, contradicting our security assumptions. (2) Alternatively, the adversary may produce false witnesses corresponding to a valid transaction within the voting period. In this case, the adversary finds a preimage Λ' such that $\Lambda' = H(u_{id} | v_{id} | v | \beta)$, where H is the collision-resistant hash function (e.g., `Keccak`). Successfully doing so breaks the collision resistance of H, which contradicts our assumptions.

We calculate the overall probability of the adversary's successfully breaking the soundness of the statement in Definition 3.1 as follows:

$$\Pr\left[\begin{array}{c} \Big(\mathcal{V}_{MPT}(\texttt{s\_r}, \pi_{\Lambda}, \Lambda, \texttt{rlp}_{\Lambda}) \neq 1 \\ \vee\, \texttt{rlp}_{\Lambda} \neq \mathsf{RLP}(0, a, \phi, \phi) \\ \vee\, \Lambda \neq H(u_{id} \| v_{id} \| v \| \beta) \\ \vee\, \eta \neq H(u_{id} \| v_{id} \| H(\beta))\Big) \wedge \\ \mathcal{V}_{\pi}(vk, \langle \eta, v_{id}, v \rangle, \pi_{\mathrm{burn}}) = 1 \end{array} \;\middle|\; \begin{array}{c} pp \leftarrow \mathcal{G}(1^{\lambda}), \\ (pk, vk) \leftarrow \mathcal{K}(pp, S), \\ v_{id}, \texttt{rlp}_{\Lambda}, \Lambda, \eta, \beta \leftarrow \mathcal{P}^* \\ \pi_{\Lambda} \leftarrow \mathcal{E}(pp, \rho) \end{array} \right] = \mathrm{negl}(1^{\lambda})$$

□

4.3 Coercion Resistance and Receipt-Freeness

We argue that the proposed method provides strong coercion resistance. As discussed in Section 3.3, voters are able to cast multiple votes, each of which is cryptographically unlinkable from the others. Moreover, voters may optionally use different EOAs to further reduce the risk of correlation between their voting actions. These features collectively mitigate the risk of coercion, as voters can plausibly deny which vote corresponds to their true intent.

Nevertheless, our system does not provide full receipt-freeness. Since votes are publicly verifiable upon submission, voters can confirm the inclusion of their vote in the tally immediately after casting it. While this property improves transparency and auditability, it also allows voters to construct partial receipts, which could be misused in coercion scenarios. Preventing this would require additional protocol design that obfuscates the link between the vote and the voter even after submission, which we leave for future work.

4.4 Security Guarantees

In our system model, we aim to satisfy the following security properties:

- **Double Voting:** Double voting is prevented by deriving a unique nullifier from the blinding factor β used in the burn address computation, this bounds the vote to the nullifier and then each nullifier is bound to one vote submission by keeping track of all used nullifiers.

- **Vote Tampering:** Once a vote is cast, it cannot be altered by any party, including the original voter or any external adversary. This immutability is ensured by two properties: the binding nature of the burn address (Theorem 1) and the soundness of the proof-of-burn (Theorem 2).
- **Vote Suppression/Censorship:** Valid votes cannot be censored or ignored by the system. Any proof-of-burn submission that satisfies the verification criteria is automatically accepted and included in the on-chain tally. This guarantee is underpinned by the decentralization and censorship-resistance properties of the blockchain.
- **Eligibility Enforcement:** Only users who perform a valid burn transaction tied to a specific election are eligible to vote—ensured by the binding property of the burn address and the soundness of the proof-of-burn (see Section 4.2). If an *allowlist* is used, voters must also provide an inclusion proof (e.g., a Merkle proof) relative to a committed list. The secrets used to hide inclusion in that list can be also used to deterministically derive the corresponding burn address.
- **Tally Integrity:** The final tally accurately reflects only the set of valid votes submitted during the voting period, with no manipulation, omission, or duplication. This integrity is guaranteed by the on-chain execution of the tallying logic, which automatically verifies each proof-of-burn and associated vote before inclusion. Furthermore, the process is publicly auditable and trustless: any observer can independently verify the correctness of the tally using the publicly available on-chain data.
- **Soundness of Burn Process:** This is guaranteed by the Theorem 2.
- **Vote Validity:** In addition to the constraints within the ZK circuit, vote validity is further guaranteed by on-chain verification during submission. Since the vote is revealed at the time of submission, the smart contract can enforce additional validity checks (e.g., option range or format).
- **Voter Anonymity and Vote Unlinkability:** The protocol ensures that no link can be established between a vote and the identity of the voter at any stage. This is achieved through the hiding property of the burn address commitment (Theorem 1) and the zero-knowledge guarantees of the proof system. Note that vote submission occurs from an address unrelated to the one used for burning tokens.
- **Coercion Resistance:** The protocol provides coercion resistance by design as discussed in Section 4.3.
- **Support for Revoting:** In the *burn-to-vote* model, revoting is inherently supported—voters can perform multiple burn operations and later choose which corresponding vote to submit. To enhance flexibility, the protocol optionally supports a *revote* flag. When enabled, voters may overwrite a previously submitted vote by submitting another vote with the same nullifier before the voting period ends. This feature strengthens coercion resistance by allowing voters to invalidate coerced or observable votes through a subsequent, unlinkable submission.

5 Overview of Supported Voting Scenarios

The protocol accommodates a wide range of voting models configurable through the tally function in the smart contract, while ensuring vote integrity and verifiability across all scenarios. Note that the core components, such as burn address generation, nullifier derivation, and vote submission, remain unchanged which gives the protocol a high degree of modularity that can be extended to a wide range of voting models. For example, additive voting schemes can be integrated with minimal effort, as they require only simple summation logic in the tally.

Table 3 provides a non-exhaustive summary of voting models supported by our protocol, along with their associated cryptographic and computational characteristics. Below, we briefly describe two of the most commonly used models *Majority Voting* and *Token-Weighted Voting* in addition to *Open-Content Voting*. For further supported scenarios and implementation details, see Appendix A.

Majority Voting (First-Past-the-Post): FPTP is a binary majority voting system where each voter casts a single vote: *yes* (1) or *no* (0). The outcome is determined by whether the number of affirmative votes exceeds half of the total number of votes N. In our protocol, this is realized as an equal-weight scheme where each voter burns a fixed amount of tokens and submits a binary vote. Let N denote the total number of voters. The tally sums all support votes:

$$\text{score}_{\text{yes}} = \sum_{i=1}^{N} v_i, \quad \text{where } v_i \in \{0, 1\}$$

The outcome is determined by $\text{score}_{\text{yes}} > \frac{N}{2}$.

Token-Weighted Voting: Token-weighted voting allows each voter to cast a vote for exactly one candidate by burning a chosen amount of tokens. The vote weight for that candidate is proportional to the number of tokens burned. We suggest the *Burn-to-vote* mechanism since the weight must be tied to the exact amount of tokens burned at vote time. The tally is performed as follows:

$$\text{score}_{\text{candidate}} = \sum_{i=1}^{N} T_i$$

where N is the total number of voters and T_i is the number of tokens burned by voter i for the selected candidate.

Open-Content Voting: This voting scheme allows voters to submit votes containing **arbitrary content**, such as any string, number, or textual proposal – essentially, any data the voter wishes to express. This is suitable for open-ended decisions like selecting a protocol name, slogan, or other customizable inputs where predefined options are insufficient.

Each voter burns tokens and submits their chosen content directly as their vote. The content can be any sequence of characters or numeric value, without restrictions on format or domain. The tally is performed by counting the number of votes for each unique content item, and the content with the highest count is considered the winner. Thus, this scheme can be viewed as a generalized form of majority voting extended to arbitrary vote values. To the best of our knowledge, we are the first to support such expressive, privacy-preserving free-form voting.

Table 3: Supported Voting Schemes

Type	Burn Mode	Vote	Domain	Tally Method
FPTP	Both	Binary	$\{0,1\}$	Majority
Token-Wtd	B2V	Scalar	$[0, w]$	Weighted Sum
Quadratic	B2V	Vector	$[0, w]^m$	Weighted Sum
Ranked-Choice	Both	Perm.	$\pi([0, m-1])$	Borda/IRV
Open-Content	Both	Arbitrary	Any	Freq. Cnt.

Burn: B2R = Burn-to-Register, B2V = Burn-to-Vote, Both = supports both.
Format: Perm. = permutation; Arbitrary = string/number.
Domain: Set of allowed vote values; scalar means single value, vector means multiple values, permutation means ranking of candidates.
m: Number of candidates. *w*: Maximum allowed voting weight or token amount.
$\pi(S)$ denotes the set of permutations of S. **Tally:** Freq. Cnt. = frequency count.

Table 4: Circuit Complexity Breakdown

Circuit	NL	L	In (P)	In (S)	Out
Vote	8,485K	3,899K	5	8,827	1
BurnAddress	567	785	2	3	2
Nullifier	264	341	1	2	1
MerkleTree	494	548	0	5	1
MPT (w/RLP)	8,482K	3,897K	0	8,857	0
RLP	2,859	403	0	83	66

P = Public, S = Secret, NL = Non-linear Constraints, L = linear Constraints

Scalability and Proof Complexity: All supported voting models share a common proof structure centered around burn validity and nullifier uniqueness. Differences lie in the vote integrity logic (e.g., range checks), but these have minor impact on the prover complexity. Since proof-of-burn (e.g., inclusion in MPT of EVM-based blockchains) holds the dominant number of constraints, overall proof size and verification cost remain almost consistent across different voting models. This uniformity ensures that scalability is not affected by the choice of voting model.

6 Implementation

In this section, we first present an overview of the proof generation process, followed by a detailed analysis of the circuit-level complexity. We then describe key implementation aspects of the on-chain components, including smart contract logic and proof-of-burn verification. Finally, we evaluate the system's performance by measuring proof generation time and on-chain gas costs for verification and tallying, and compare our results with the state-of-the-art voting protocols.

Note on Support for Optional Allow-List: While not required by default, the protocol supports integration with an *allow-list* when needed by specific voting models. In such cases, the circuit includes a proof of inclusion within a committed list of eligible voters. We assume the following setup:

- Voters are registered (on/off-chain) into a list represented by a vector commitment, such as a Merkle tree over Poseidon hashes of tuples (sk, β).
- The same secrets used for inclusion in the allow-list are reused to derive the burn address, ensuring binding between eligibility and vote casting.
- A public commitment to the allow-list, such as a Merkle root r_M, is published on-chain.

6.1 Proof Generation

Voters generate a proof that jointly attests to four key properties: (1) the correctness of the burn address (Λ) and the corresponding nullifier (η); (2) the binding of the vote to these elements as described in Section 3.3; (3) the inclusion of the burn address within the last block's state root, i.e., a valid MPT proof; and (4) the voter's membership in an optional allow-list via a Merkle proof.

Allow-List Inclusion: Prove knowledge of (sk, β) such that $l = H(sk \parallel \beta)$ exists in the Merkle tree with root r_M.

State-Trie Inclusion: Prove that burn address Λ exists in the state trie with root `s_r` (last voting block) and has nonce=0 (verifying no private key). The proof path consists of MPT nodes from leaf to root:

1. **Leaf Verification:** Confirm that the leaf node contains:
   - Key: Remaining nibbles of $\mathtt{keccak}(\Lambda)$ after shared prefix (k_u)
   - Value: $\mathrm{rlp}_\Lambda = \mathrm{RLP}(0, balance, storageRoot, codeHash)$
2. **Path Validation:** For each node in path (leaf $\rightarrow$ root):
   - Compute $h_i = \mathtt{keccak}(\mathrm{RLP}(node_i))$
   - Verify h_i matches child reference in parent node
3. **Root Match:** Confirm final $h_n = $ `s_r`

Circom implementation: We implemented the statement of Definition 3.1 in Circom. The circuit verifies vote validity, enforces the existence of the burn address Λ within the specified MPT state, and generates the nullifier. Specifically, it performs the following checks:

- Enforces input bounds for vote value and revote flag
- Checks RLP encoding structure and length constraints
- Validates MPT proof parameters
- Confirms account state parameters:
  `nonce=0`, `balance>0`, `storageRoot=`ϕ, `codeHash=`ϕ
- Computes and checks the burn address commitment
- Verifies Merkle tree inclusion of the vote
- Generates a nullifier tied to the burn address and secret
- Validates account state inclusion in the MPT

Table 4 reports the constraint counts for our circuit embedding RLP and MPT logic. The full Circom implementation is provided in Appendix B.

6.2 Voting Smart Contract

For each voting ceremony, the appropriate voting contract is instantiated by a factory (generator) contract using the `CREATE2` opcode, ensuring a deterministic address. Depending on the chosen voting scheme (e.g., majority vote, quadratic voting), the factory deploys a specialized implementation.

The voting system is implemented as a smart contract in Solidity, with core functionalities like `submitVote`, `submitRevote` and `tally`. The implementation ensures privacy through zkSNARKs while maintaining auditability of the voting process. See the functionality details and the smart contract algorithm in Appendix D.

Table 5: Performance Metrics of Cryptographic Operations

Operation	Time (s)	Size (MB)	Gas Cost (k)
Circuit Compilation	1,500 ± 100	1843.2	–
Proving Key	–	5.5 GB	–
Verification Key	–	3.8 KB	–
Witness Generation	30 ± 0.02	372	–
Proof Generation	37 ± 2	0.804	–
Proof Verification	0.01 ± 0.005	–	198
Smart Contract Deploy	–	–	1,809
Vote Submission	–	–	313
Revote Operation	–	–	525
Tally Execution	–	–	50.5
Tally Result	–	–	1.03

To evaluate the system with real-world data, we deployed[2] the implemented contract system on the Sepolia testnet. We also successfully executed the `submitVote`[3] and `tally`[4] transactions on-chain.

6.3 Experimental Results

System Setup: The experimental environment was a *MacBook Air* with an Apple M3 chip running macOS Sequoia; see Appendix C for details.

We analyze the computational and communication costs of our protocol and benchmarked its performance against leading solutions. In terms of proof complexity, since we use Groth16, the proof size and verification time remain always constant regardless of the number of votes or the specific voting scenario. on the other hand, the prover complexity is also not much meaning that the prover can be run on a lightweight laptop (MacBook Air). Table 5 shows the performance metrics of our prototype implementation. the proof generation time is only 37 seconds, which is acceptable for a commodity laptop.

In terms of on-chain costs, the contract performs only a Groth16 proof verification and updates the vote tally accordingly. Specifically, the `submitVote` transaction consumes approximately 300k gas, which is significantly lower than related work [25, 40]. The `tally` transaction incurs a cost of only around 50k gas, making it highly efficient.

We compare the gas costs of our design with those of [40] and [25] in a simple Yes/No voting scenario, as summarized in Table 6. Our protocol achieves up to 25% lower gas consumption compared to Cicada [25] and up to 21% lower compared to Kite [40]. To further demonstrate the scalability of our system, we evaluate its performance against [25] in a majority voting setting with varying numbers of candidates, ranging from 1 to 1000, as shown in Figure 2.

A key observation is that, as the number of votes or candidates increases, our gas cost grows slowly, whereas the costs of other systems rise sharply, becoming impractical at scale. The primary

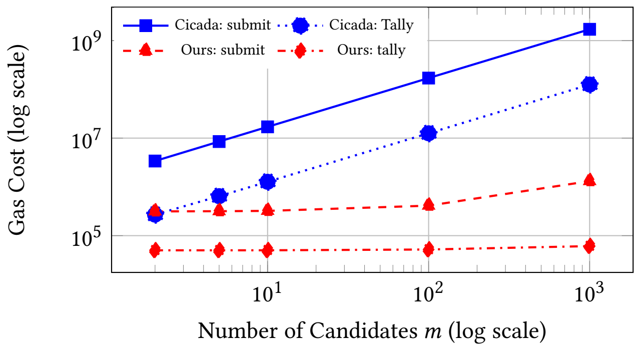

Figure 2: Gas cost comparison of cicada vs. our design for *vote submission* and *tally* over varying the number of candidates.

Table 6: Yes/No Voting Gas Cost Comparison

System	Gas Cost	Δ vs. Cicada	Δ vs. Kite
This work	312 927	-25.2 %	-21.0 %
Kite	396 323	-5.3 %	—
Cicada	418 358	—	+5.6 %

reason for our improved efficiency is that the on-chain logic consists of simple vote count aggregation after the voting deadline. In contrast, many other designs rely on off-chain tallying mechanisms that require complex coordination or state updates. Our system supports over one million voters with tally costs equivalent to those in a setting with just 100 voters. Furthermore, the gas cost of vote submission remains constant regardless of the number of votes or candidates. Table 7 summarizes the qualitative comparison of our design with the other two schemes.

7 Related Work

Private voting has been evolved and developed usign defferent cryptographic methods and primitives such as FHE, ZKP, and MPC to enhance privacy, verifiability, and decentralization. In this section, we address some of the recent and most relevant related work by their primary methodology and summarize their core mechanisms, advantages, and limitations in comparison to our protocol. Table 1 summarizes the key trade-offs across some of these protocols.

Kite [40] is a voting system for DAOs that enables private delegation of voting power and encrypted vote aggregation. It combines HE and ZKPs: voters submit encrypted ballots and use ZKPs to prove their correctness without revealing the content. A trusted authority (TA) decrypts the final tally off-chain, which limits transparency and introduces a critical point of trust and potential vulnerability. Kite supports both public and private delegate voting, depending on the configuration chosen by the DAO. Our protocol avoids the overhead of using HE, enabling transparent on-chain vote verification while preserving voter anonymity.

Chaintegrity [46] is a blockchain-enabled e-voting system emphasizing scalability, robustness, and verifiability. It uses distributed homomorphic encryption (threshold Paillier) [41] for vote aggregation and threshold decryption, blind signatures [14] for anonymous

[2]Deployed contract: https://sepolia.etherscan.io/address/0xbf36ae91d220ca3f387ee63979eb433ed9b1d0be#code
[3]`submitVote` transaction: https://sepolia.etherscan.io/tx/0x7acda7e3677372246dd523e6fd039f61110f7f8d970af36606bbc30dd60f08cf
[4]`tally` transaction: https://sepolia.etherscan.io/tx/0xeb07a4fae6115a6eaba390aedac47dfcc67454df5b95db38667ee1f6c460db46

Table 7: Qualitative Comparison of Voting Schemes

Characteristic	TLP-based (Cicada)	FHE-based (Kite)	Burn Your Vote (Ours)
Registration	Yes, public bulletin board	Yes, delegates register on-chain	Optional burn-to-register
Pre-computation	High (HTLP setup & proof generation)	High (AHE+SNARKs+Merkle)	Low (ZK proof generation only)
Per-Vote Cost	Medium (HTLP + NIZKs)	Medium-High (ZK + AHE vote)	Low (ZKP + plaintext vote)
Tally Complexity	Medium (off-chain HTLP + on-chain finalize)	High (trusted auth. decrypts tally)	Low (on-chain ZKP verify + tally)
Tally Trust	Smart contract + untrusted solver	Trusted authority decrypts	Fully decentralized, no coordinator
Maximum Voters	Medium (HTLP aggregation bound)	Medium (performance bound by encryption + Merkle verification)	High (scales with EVM throughput)

authentication, and zero-knowledge proofs for vote correctness. Smart contracts manage the process on-chain. The system aims for privacy, efficiency, universal/end-to-end verifiability, and robustness. It also employs a hybrid data structure (Bloom filter [21] + Merkle tree [39]) for fast authentication and introduces the code-voting technique for robustness. However, Chaintegrity relies on multiple rounds of interaction, blind signature issuance, and trusted setup [12, 15] for threshold decryption, which introduces coordination complexity and may pose deployment challenges in high-coercion or adversarial settings.

Cicada [25] uses homomorphic time-lock puzzles (HTLPs)[37] to enable private, one-round vote submission. Voters encrypt their ballots so they can only be opened after a preset time. This avoids interaction but leads to post-tally vote disclosure, as all ballots are eventually revealed. Cicada can prevent coercion or vote selling by revoting but it still depends on trusted cryptographic setup. Our protocol also supports a one-round voting process, but unlike Cicada, it submits vote values directly to the smart contract along with zkSNARK proofs, which preserves verifiability while still maintaining unlinkability between voters and votes.

Semaphore [43] is a general-purpose anonymity tool that allows users to prove Merkle tree membership and submit anonymous signals using zkSNARKs. Semaphore supports unlinkability and mitigates coercion by hiding user identity. This is achieved through ZKPs allowing users to prove group membership and send signals without revealing their identity, coupled with unique nullifiers derived from a secret identity nullifier and an external nullifier that prevent linking multiple signals from the same identity for a specific event. Because a coercer cannot verify which identity sent a specific signal, they cannot confirm compliance with their demands, thereby resisting coercion. However, it is not a complete voting protocol—it lacks eligibility checks, vote valid- ity enforcement, and tallying functionality.

McCorry et al. [38] propose a self-tallying voting protocol using Schnorr-based NIZKs. Voters publish encrypted binary votes to the blockchain, and the final tally is computed by homomorphically aggregating all ciphertexts. The protocol achieves privacy guarantees—an individual vote remains hidden unless the voter reveals their key or all other voters collude. It also enables verifiable, decentralized tallying without requiring a trusted authority. However, the approach depends on a global public key setup and requires per-voter key management, which can limit scalability and usability in larger elections.

Dimitriou [19] introduces a coercion-resistant protocol using a tamper-resistant token randomizer. [36] Voters never learn the randomness used in ballot creation, preventing them from proving how they voted. Votes are encrypted and posted along with zkSNARKs and nullifiers. While effective against coercion, this system relies on trusted platform module (TPM) [1] and external decryption, which limits decentralization. Our protocol does not rely on trusted hardware to achieve coercion resistance.

PPVSS [6] enhances traditional PVSS by enabling dealers to publish encrypted shares and commitments prior to the reconstruction phase. This supports *optimistic reconstruction*: if no complaints are raised, the shared secret can be reconstructed efficiently without additional interaction. The protocol improves upon Schoenmakers' scheme by reducing verification complexity to $O(nm)$ for n talliers and m voters, making it suitable for scalable applications like e-voting, This is because it leverages a more efficient PPVSS scheme requiring only $O(n)$ exponentiations for verification per ballot. While PPVSS maintains privacy and verifiability under standard cryptographic assumptions, it assumes a dealer-based setup and does not specify full on-chain execution.

MACI [4] Minimal Anti-Collusion Infrastructure (MACI) is an open-source protocol for private on-chain voting that uses Ethereum smart contracts and zkSNARKs to enhance privacy and collusion resistance. While designed to prevent bribery by making it impossible for voters to prove how they voted and ensuring correct tallying via zkSNARKs verified on-chain, MACI has limitations primarily related to the trusted coordinator. A corrupt coordinator can decrypt votes and halt the voting process by not publishing the final results or proofs. However, a corrupt coordinator cannot publish incorrect results or censor valid votes due to the on-chain verification of zkSNARK proofs.

8 Conclusion

This work presents a protocol for a fully decentralized, on-chain voting system based on the proof-of-burn mechanism. The protocol preserves voter privacy by ensuring unlinkability between the vote value and the voter's identity, achieves high scalability due to the use of plaintext votes, and maintains efficiency without relying on trusted parties or off-chain components. Furthermore, the protocols achieves coercion resistance and easy vote delegation. To enhance flexibility, the protocol supports two modes: burn-to-vote and burn-to-register, each designed for different use cases.

We implemented the full protocol as a proof of concept using the Circom [7] DSL and the Groth16 [29] ZKP system, and deployed the tallying smart contracts on the Ethereum testnet. The results demonstrate that the proposed voting mechanism significantly reduces gas costs compared to existing protocols, indicating strong practical viability for real-world deployments.

While our system provides coercion resistance, it does not yet achieve full receipt-freeness, a limitation shared with other on-chain voting protocols. The protocol is also well-suited for auction mechanisms, which we plan to explore in future work.

Acknowledgments

The authors used ChatGPT4 only for grammatical revision of the text in the paper to correct any typos, grammatical errors, and awkward phrasing. This work was supported by the European Research Council (ERC) under the European Union's Horizon 2020 innovation program (grant PROCONTRA-885666).

References

[1] Trusted platform module (tpm) summary. Accessed: 2025-05-11.

[2] Rethinking voter coercion: The realities imposed by technology, 2013.

[3] Eip-7503: Zero-knowledge wormholes, 2023.

[4] Minimal anti-collusion infrastructure. https://maci.pse.dev/, 2025.

[5] Joël Alwen, Rafail Ostrovsky, Hong-Sheng Zhou, and Vassilis Zikas. Incoercible multi-party computation and universally composable receipt-free voting. In *CRYPTO*, 2015.

[6] Karim Baghery, Noah Knapen, Georgio Nicolas, and Mahdi Rahimi. Pre-constructed publicly verifiable secret sharing and applications. Cryptology ePrint Archive, Paper 2025/576, 2025.

[7] Marta Bellés-Muñoz, Miguel Isabel, Jose Luis Muñoz-Tapia, Albert Rubio, and Jordi Baylina. Circom: A circuit description language for building zero-knowledge applications. *IEEE Transactions on Dependable and Secure Computing*, pages 1–18, 2022.

[8] Josh Benaloh. Simple verifiable elections. In *Proceedings of the USENIX/Accurate Electronic Voting Technology Workshop 2006 on Electronic Voting Technology Workshop*, EVT'06, page 5, USA, 2006. USENIX Association.

[9] Josh Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '94, page 544–553, New York, NY, USA, 1994. Association for Computing Machinery.

[10] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 329–349. 2019.

[11] Peter Bogetoft, Ivan Damgård, Thomas Jakobsen, Kurt Nielsen, Jakob Pagter, and Tomas Toft. A practical implementation of secure auctions based on multiparty integer computation. In *Financial Crypto*, 2006.

[12] Dan Boneh and Matthew Franklin. Efficient generation of shared rsa keys. *J. ACM*, 48(4):702–722, July 2001.

[13] Vitalik Butarin. Blockchain voting is overrated among uninformed people but underrated among informed people, 2021.

[14] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology*, pages 199–203, Boston, MA, 1983. Springer US.

[15] Megan Chen, Carmit Hazay, Yuval Ishai, Yuriy Kashnikov, Daniele Micciancio, Tarik Riviere, abhi shelat, Muthu Venkitasubramaniam, and Ruihan Wang. Diogenes: Lightweight scalable RSA modulus generation with a dishonest majority. Cryptology ePrint Archive, Paper 2020/374, 2020.

[16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. A homomorphic lwe based e-voting scheme. In *PQC*, 2016.

[17] Ivan Damgård. Commitment schemes and zero-knowledge protocols. In *School organized by the European Educational Forum*, pages 63–86. Springer, 1998.

[18] Rafaël Del Pino, Vadim Lyubashevsky, Gregory Neven, and Gregor Seiler. Practical quantum-safe voting from lattices. In *ACM CCS*, 2017.

[19] Tassos Dimitriou. Efficient, coercion-free and universally verifiable blockchain-based voting. *Computer Networks*, 174:107234, 2020.

[20] Electric Coin Company. Zcash protocol specification. https://zips.z.cash/protocol/protocol.pdf, 2022. Accessed: 2024-05-01.

[21] Li Fan, Pei Cao, J. Almeida, and A.Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.

[22] Ethereum Foundation. Eip-4337: Account abstraction via entrypoint contract specification. https://eips.ethereum.org/EIPS/eip-4337, 2023. Accessed: 2024-05-15.

[23] Jon Fraenkel and Bernard Grofman and. The borda count and its real-world alternatives: Comparing scoring rules in nauru and slovenia. *Australian Journal of Political Science*, 49(2):186–205, 2014.

[24] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.

[25] Noemi Glaeser, István András Seres, Michael Zhu, and Joseph Bonneau. Cicada: A framework for private non-interactive on-chain auctions and voting. *Cryptology ePrint Archive*, 2023.

[26] Shafi Goldwasser, Silvio Micali, and Chales Rackoff. The knowledge complexity of interactive proof-systems. In *Providing sound foundations for cryptography: On the work of shafi goldwasser and silvio micali*, pages 203–225. 2019.

[27] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In *USENIX Security Symposium*, volume 2021, 2021.

[28] Gurchetan S. Grewal, Mark D. Ryan, Sergiu Bursuc, and Peter Y.A. Ryan. Caveat coercitor: Coercion-evidence in electronic voting. In *2013 IEEE Symposium on Security and Privacy*, pages 367–381, 2013.

[29] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[30] Aayush Gupta and Kobi Gurkan. PLUME: An ECDSA nullifier scheme for unique pseudonymity within zero knowledge proofs. Cryptology ePrint Archive, Paper 2022/1255, 2022.

[31] Kobi Gurkan, Barry Whitehat, Koh Wei Jie, David Wong, Jacob Illum, Tom French, and Vitalik Buterin. Semaphore: Anonymous signaling on ethereum and beyond. https://semaphore.pse.dev/whitepaper-v1.pdf, 2021. Version 1.0.

[32] Luke Harrison, Samiran Bag, and Feng Hao. Camel: E2E verifiable instant runoff voting without tallying authorities. Cryptology ePrint Archive, Paper 2023/1920, 2023.

[33] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, WPES '05, page 61–70, New York, NY, USA, 2005. Association for Computing Machinery.

[34] Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. Proof-of-burn. In *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24*, pages 523–540. Springer, 2020.

[35] Steven P. Lalley and E. Glen Weyl. Quadratic voting: How mechanism design can radicalize democracy. *AEA Papers and Proceedings*, 108:33–37, May 2018.

[36] Byoungcheon Lee and Kwangjo Kim. Receipt-free electronic voting scheme with a tamper-resistant randomizer. In Pil Joong Lee and Chae Hoon Lim, editors, *Information Security and Cryptology — ICISC 2002*, pages 389–406, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[37] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. Homomorphic time-lock puzzles and applications. Cryptology ePrint Archive, Paper 2019/635, 2019.

[38] Patrick McCorry, Siamak F Shahandashti, and Feng Hao. A smart contract for boardroom voting with maximum voter privacy. In *Financial Cryptography and Data Security: 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers 21*, pages 357–375. Springer, 2017.

[39] Ralph C. Merkle. Protocols for public key cryptosystems. In *1980 IEEE Symposium on Security and Privacy*, pages 122–122, 1980.

[40] Kamilla Nazirkhanova, Vrushank Gunjur, X Jesus, and Dan Boneh. Kite: How to delegate voting power privately. *arXiv preprint arXiv:2501.05626*, 2025.

[41] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[42] Sunoo Park and Ronald L. Rivest. Towards secure quadratic voting. *Public Choice*, 172(1/2):pp. 151–175, 2017.

[43] Alexey Pertsev, Roman Semenov, and Roman Storm. Tornado cash privacy solution version 1.4. 2019.

[44] Douglas R. Woodall. The original borda count and partial voting. *Social Choice and Welfare*, 40(2):353–358, 2013.

[45] Xun Yi and Eiji Okamoto. Practical internet voting system. *Journal of Network and Computer Applications*, 36(1):378–387, 2013.

[46] Shufan Zhang, Lili Wang, and Hu Xiong. Chaintegrity: blockchain-enabled large-scale e-voting system with robustness and universal verifiability. *International Journal of Information Security*, 19(3):323–341, 2020.

A Additional Voting Types

A.0.1 ***Quadratic Voting [42].*** Quadratic voting allows voters to express the intensity of their preference by allocating votes such that the cost grows quadratically with the number of votes cast.

Formally, a voter's ballot is a vector $\mathbf{b} = (b_1, b_2, ..., b_m)$ where the quadratic norm satisfies

$$\|\mathbf{b}\|_2^2 = \sum_{j=1}^{m} b_j^2 \leq w,$$

with w representing the voter's budget constraint.

In our protocol, vote weight corresponds to the square root of tokens burned, i.e., burning T tokens yields vote weight $\sqrt{T}$. Only absolute values of burned tokens are considered. Burn-to-vote is suggested since the weight must be tied to the exact amount of tokens burned at vote time.

Tally:

$$\text{score}_j = \sum_{i=1}^{N} b_{i,j},$$

where $b_{i,j} = \sqrt{T_{i,j}}$ is the weight that voter i assigns to the candidate j. This approach enables voters to weigh stronger preferences more heavily while keeping total token cost quadratic in vote weight.

A.0.2 ***Ranked-Choice Voting (RCV).*** Ranked-choice voting allows voters to rank candidates in order of preference, enabling more expressive voting compared to single-choice schemes.

Each voter submits a ranking vector $\mathbf{R}$, which is a permutation of the m candidates, where R_k denotes the candidate ranked at position k. For example, R_1 is the top choice, R_2 the second choice, and so on.

Two common tally methods are:

- **Borda Count:** [23, 44] Each candidate receives points inversely proportional to their ranking position:

$$\text{score}_{\text{candidate } j} = \sum_{\text{all voters}} (m - k_j)$$

  where k_j is the position of candidate j in the voter's ranking. The candidate with the highest total score wins.
- **Instant-Runoff Voting (IRV):** [32] Votes are counted in rounds. In each round, the candidate with the fewest first-choice votes is eliminated, and those votes are transferred to the next preferred candidate on each ballot. This continues until a candidate obtains a majority.

In our protocol, voters burn a fixed amount of tokens and submit the ranking vector $\mathbf{R}$. The tally logic is flexible to support different scoring rules without changes to the proof structure.

B Circom Vote Circuit

```
template Vote(maxDepth) {
    // Input signals
    signal input address;
    signal input nullifier;
    signal input secret;
    signal input random_secret;
    signal input blinding_factor;
    signal input ceremonyID;
    signal input nonce;
    signal input balance;
    signal input storage_hash[32];
    signal input code_hash[32];
    signal input state_root[64];
    signal input account_rlp[164];
    signal input account_rlp_len;
    signal input account_proof[maxDepth][1064];
    signal input account_proof_length;
    signal input node_length[maxDepth];
    signal input leaf_nibbles;
    signal input vote;
    signal input revote;

    // nonce check
    nonce === 0;

    // Burn address verification
    component burn_address = BurnAddress();
    burn_address.secret <== secret;
    burn_address.blinding_factor <== blinding_factor;
    burn_address.ceremonyID <== ceremonyID;
    burn_address.random_secret <== random_secret;
    burn_address.vote <== vote;
    address === burn_address.address;

    // Merkle tree inclusion proof
    component merkle_tree_inclusion =
        MerkleTreeChecker(2);
    merkle_tree_inclusion.leaf <== mt_leaf;
    merkle_tree_inclusion.pathElements <==
        mt_pathElements;
    merkle_tree_inclusion.pathIndices <==
        mt_pathIndices;
    mt_root === merkle_tree_inclusion.root;
    mt_leaf === burn_address.secret_commitment;

    // Nullifier verification
    component nullifier_generator = Nullifier();
    nullifier_generator.secret <== secret;
    nullifier_generator.blindingFactor <==
        blinding_factor;
    nullifier_generator.ceremonyID <== ceremonyID;
    nullifier === nullifier_generator.nullifier;

    // MPT proof verification
    component check_account = Mpt(maxDepth);
    check_account.address <== hex_address;
    check_account.nonce <== nonce;
    check_account.balance <== balance;
    check_account.storage_hash <== storage_hash;
    check_account.code_hash <== code_hash;
    check_account.state_root <== state_root;
    check_account.account_rlp <== account_rlp;
    check_account.account_rlp_len <== account_rlp_len
        ;
    check_account.account_proof <== account_proof;
    check_account.account_proof_length <==
        account_proof_length;
    check_account.node_length <== node_length;
    check_account.leaf_nibbles <== leaf_nibbles;
}
component main{public[ceremonyID, nullifier, vote,
    revote, mt_root, state_root]} = Vote(8);
```

Listing 1: Vote Circuit in Circom

C System-setup

This section details the experimental setup for developing and evaluating our privacy-preserving voting protocol, following reproducibility guidelines from ACM SIGSAC.

C.1 Experimental Environment

- **Hardware:** Apple MacBook Air (Mac15,12 MXCV3ZE/A)
  - Chip: Apple M3 (4 performance cores, 4 efficiency cores)
  - Memory: 16GB Unified Memory
  - Storage: 500GB SSD
- **Operating System:**
  - macOS Sequoia 15.1 (24B83)
  - Kernel: Darwin 24.1.0
  - Shell: Zsh 5.9 (arm64-apple-darwin24.0)

C.2 Software Stack

- **Smart Contract Development:**
  - Foundry v0.2.0 (forge 1.0.0-stable)
  - Solidity v0.8.23 with via-IR optimizer (200 runs)
  - Slither v0.10.1 for static analysis
  - rustup v1.28.1 (f9edccde0 2025-03-05)
- **ZKP Circuit Development:**
  - Circom 2.2.1 with circomlib 2.0.5
  - snarkjs v0.6.11 (Groth16 implementation)
  - rapidsnark v0.0.7 for WASM-based witness generation

C.3 Cryptographic Parameters

- **ZKP System:**
  - Groth16 over BN-254 curve (EIP-196 compatible)
  - Trusted setup: Perpetual Powers of Tau (phase 1) + circuit-specific phase 2
  - Final zKey size: 5.5GB

D Solidty Vote Smart contract

D.0.1 Contract Initialization. To deploy a majority-vote contract, the factory must supply the following constructor arguments:

- $verifier$: Address of the Groth16 zkSNARK verifier.
- $submissionDeadline$: UNIX timestamp by which all proofs must be submitted.
- $tallyDeadline$: UNIX timestamp after which tallying is permitted.
- $merkle\_root$: Root of the allow-list Merkle tree for voter eligibility.
- $state\_root$: Ethereum state root at the time voting ended, used for inclusion proofs.

D.0.2 Core Functionality. The contract implements three main functions:

submitVote. Accepts zkSNARK proofs for new votes with the following checks:

- Voting period must be active
- Vote value must be valid (0 or 1)
- Revoting flag must be false
- Merkle root and ceremony ID must match
- State root must be valid
- zkSNARK proof must verify

submitRevote. Allows voters to change their vote by submitting:

- Proof of original vote
- Proof of new vote
- Matching nullifiers between proofs
- Opposite vote values
- Proper revoting flag

Updates vote counts accordingly while maintaining privacy.

tallyVotes. Finalizes the voting results after the tally deadline:

- Can only be called once
- Requires tally deadline to have passed
- Aggregates the votes
- Emits final results

D.0.3 Gas Optimization. The contract employs several gas-saving techniques:

- Uses `uint128` for vote counters
- Packed storage layout
- Early reverts for invalid states
- Minimal storage operations

D.0.4 Security Considerations. Key security features include:

- Nullifier tracking prevents double voting
- Time-based function restrictions
- Immutable verifier contract
- Comprehensive input validation
- Private vote counters to prevent manipulation

D.0.5 Performance Analysis. The contract demonstrates:

- Proof verification in constant time
- O(1) time complexity for vote submission
- Minimal storage overhead
- Efficient nullifier tracking

D.0.6 voting smart contracts algorithm.

Algorithm 1: zkSNARK Based Voting Contract with Revoting

```
Input: $\pi$: zkSNARK proof (proofA, proofB, proofC)
pubSignals: public inputs [nullifier, ceremony_id, vote, isRevote, merkle_root, state_root]
$T_{\texttt{submission}}$, $T_{\texttt{tally}}$: vote submission and tally deadlines
merkle_root, ceremony_id, state_root: system parameters
Output: Vote is recorded; final outcome published after tallying
Initialization:
  Set verifier contract
  Set deadlines $T_{\texttt{submission}}$, $T_{\texttt{tally}}$
  Initialize yesVotes $\leftarrow 0$, noVotes $\leftarrow 0$
  Set merkle_root, ceremony_id, state_root
  Initialize tallyCompleted $\leftarrow$ false
  Initialize usedNullifiers mapping

Function submitVote($\pi$, pubSignals):
    if current time $> T_{\texttt{submission}}$ then
        Revert: VotingPeriodEnded
    if vote $\notin \{0, 1\}$ then
        Revert: InvalidVote
    if usedNullifiers[nullifier] $\neq 0$ then
        Revert: NullifierAlreadyUsed
    if isRevote $= 1$ then
        Revert: RevotingNotAllowed
    if merkle_root $\neq$ stored value then
        Revert: InvalidMerkleRoot
    if ceremony_id $\neq$ stored value then
        Revert: InvalidCeremonyId
    if state_root $\neq$ stored value then
        Revert: InvalidStateRoot
    if proof verification fails then
        Revert: InvalidProof
    Mark nullifier as used
    if vote $= 1$ then
        yesVotes $\leftarrow$ yesVotes $+ 1$
    else
        noVotes $\leftarrow$ noVotes $+ 1$
    end
  Emit VoteSubmitted event

Function submitRevote( $\pi_{\texttt{old}}$, pubSignals$_{\texttt{old}}$, $\pi_{\texttt{new}}$, pubSignals$_{\texttt{new}}$ ):
    if current time $> T_{\texttt{submission}}$ then
        Revert: VotingPeriodEnded
    if nullifier$_{\texttt{old}} \neq$ nullifier$_{\texttt{new}}$ then
        Revert: NullifierMismatch
    if vote$_{\texttt{old}} \notin \{0, 1\}$ then
        Revert: InvalidVote
    if isRevote$_{\texttt{old}} = 1$ then
        Revert: RevotingNotAllowed
    if merkle_root$_{\texttt{old}} \neq$ stored value then
        Revert: InvalidMerkleRoot
    if ceremony_id$_{\texttt{old}} \neq$ stored value then
        Revert: InvalidCeremonyId
    if old proof verification fails then
        Revert: InvalidProof
    if vote$_{\texttt{new}} \notin \{0, 1\}$ then
        Revert: InvalidVote
    if vote$_{\texttt{new}} =$ vote$_{\texttt{old}}$ then
        Revert: InvalidRevoteValue
    if usedNullifiers[nullifier] $\neq 1$ then
        Revert: NullifierAlreadyUsed
    if isRevote$_{\texttt{new}} = 0$ then
        Revert: RevotingNotAllowed
    if merkle_root$_{\texttt{new}} \neq$ stored value then
        Revert: InvalidMerkleRoot
    if ceremony_id$_{\texttt{new}} \neq$ stored value then
        Revert: InvalidCeremonyId
    if new proof verification fails then
        Revert: InvalidProof
  Update vote counts based on old and new votes
  Emit VoteSubmitted event

Function tallyVotes():
    if current time $< T_{\texttt{tally}}$ then
        Revert: TallyingNotAllowed
    if tallyCompleted then
        Revert: TallyAlreadyCompleted
  Set tallyCompleted $\leftarrow$ true
  Compute passed $\leftarrow$ (yesVotes $>$ noVotes)
  Emit VotingResults event
```