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

Shahriar Ebrahimi[1], Haniyeh Habibi[2], Parisa Hassanizadeh[1,3], Pardis Toolabi[2], Vincenzo Botta[4], Stefan Dziembowski[2],
[1]Zero Savvy Ltd, {shahriar, parisa}@zerosavvy.xyz
[2]University of Warsaw, {stefan.dziembowski, h.habibi, p.toolabi}@uw.edu.pl
[3] Institute of Fundamental Technological Research Polish Academy of Science (IPPT PAN)
[4]Sapienza University of Rome / Provably Technologies, botta.vin@gmail.com

***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 attempts 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* mechanism in the context of voting. 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 ballot encryption, trusted parties, or centralized tallying. The tallying process is public and operates on plaintext votes that are authenticated yet unlinkable to voters. This enables support of 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[1] fully compatible with the Ethereum Virtual Machine and evaluate it on commodity hardware (a consumer-grade laptop). All phases of the voting process are executed on the Sepolia testnet. Experimental results confirm the protocol's scalability and improved on-chain efficiency, achieving over 20% reduction in gas consumption compared to the state of the art.

I. Introduction

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 [1].

Anonymous voting enables participants to express their genuine preferences freely, without fear of retaliation, coercion, or social pressure risks often present in centralized voting systems [2], [3]. Despite the advances, existing systems continue to face challenges in simultaneously achieving privacy, transparency, scalability, and trustless authenticity. 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 [4], [5], [6]. **Trustless 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.

Prior work has explored several cryptographic techniques, such as fully homomorphic encryption (FHE), multi-party computation (MPC), and zero-knowledge proofs (ZKPs), to achieve above-mentioned properties. FHE-based systems support tallying over encrypted data [7], [8], [9], but restrict the type of votes they can process, i.e. 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 [10], [11], introducing computational and communicational 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 [12] aims to resist vote collusion through zkSNARKs and asymmetric encryption, but relies on a trusted coordinator who can view all individual submissions [12]. Cicada [13] uses homomorphic time-lock puzzles (HTLP) [14] for non-interactive vote submission, but reveals all votes post-tally and encounters scalability issues due to complexity of tallying. Chaintegrity [15] supports scalability and verifiability through blind signatures [16] and threshold decryption [17], but relies on authorities to handle tallying. Kite [18] enables delegation in DAOs with encrypted vote aggregation, but uses a trusted tallying authority, which limits transparency. These

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

examples illustrate that while existing solutions address some privacy concerns, they frequently introduce centralization that is in contrast with trustless verifiability and voter anonymity, or they compromise on scalability.

In this paper our goal is to design an anonymous voting protocol that is fully decentralized and can operate autonomously on-chain without requiring on any off-chain coordination. This would help the protocol to scale without compromising privacy of the voters. Since we cannot assume a smart contract on blockchain would be able to handle private keys, the protocol must preserve privacy without use of any encryption techniques as this would result in requiring some sort of coordinators to handle the decryption off-chain.

At the core of our protocol is a novel use of the *proof-of-burn* [19] 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 deterministically from private data, as used in [20], [21], [22], which the smart contract checks for uniqueness. Votes are submitted in plaintext alongside the proofs, enabling public tallying and verifiability, while providing unlinkability between voters and votes. The protocol ensures public verifiability, as all votes and associated proofs are published and checked on-chain, allowing any observer to independently audit the outcome.

One of the key features of our protocol is its inherent coercion resistance. The proof-of-burn mechanism enables each voter to perform multiple independent burns to random, unlinkable addresses, allowing them to commit to different votes without revealing any association to their identity or previous actions. As a result, voters can plausibly deny any coerced vote and subsequently cast their true preference. We note, however, that the nullifier derivation technique ensures that each individually registered voter can submit at most one of their burned votes. Optional features such as post-submission revoting allow overwriting previously submitted ballots, further enhancing coercion resistance and improving flexibility.

Moreover, we show that with a minor modification to the burn address generation procedure, the protocol naturally supports vote delegation. In this model, certain proofs-of-burn can be transferred to a trusted representative, who casts the vote on behalf of the original voter, without learning the voter's identity or linkable history.

Furthermore, the public and transparent tallying of plaintext votes enables support for a wide range of voting models, including majority, token-weighted, quadratic [23], [24], and ranked-choice voting.

We formally analyze the security and soundness of the protocol under a clearly defined system and adversary model. The protocol is implemented in a publicly available open-source library[2] under the MIT license, and its practicality and scalability are evaluated on EVM-compatible blockchains. For zero-knowledge circuit construction, we use the Circom [25] domain-specific language (DSL) to define the constraint system, with Groth16 [26] as the proving backend. Although our implementation currently employs Groth16, the protocol remains compatible with alternative ZK proof systems. All experiments were conducted on commodity hardware (e.g., a MacBook Air laptop) to demonstrate feasibility in realistic environments. We have deployed our smart contracts on Ethereum's Sepolia testnet, executed all protocol phases, and provide a detailed analysis of gas consumption across all smart contract functionalities, comparing our results with those of state-of-the-art voting protocols.

The main contributions of this paper are as follows:

- **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:
  - **Autonomous Tally Process:** the protocol eliminates the need for any coordinator(s) or trusted authority, a smart contract is in charge of tallying.
  - **Public Verifiability:** The entire process, including registration and tallying, is publicly auditable.
  - **Voter-Vote Unlinkability:** The protocol ensures that there is no on-chain linkage between voter registration, voting, vote value, and voter identity.
  - **Coercion Resistance:** Voters can issue multiple unlinkable burn transactions from different sources and with varying vote values, allowing plausible deniability under coercion. Additionally, we support optional revoting during submission phase.
  - **Vote Delegation:** Voters can delegate their vote at no cost to a trusted representative by sending them their proofs for *burn-to-delegate* mode (Section IV).
  - **Support for Diverse Voting Schemes:** The protocol supports any tallying logic expressible in smart contract's logic. 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 on blockchain.
- **Security Analysis:** We provide a formal analysis of the soundness of our constructions under a well-defined system and adversary model. Furthermore, we evaluate the protocol's guarantees w.r.t. coercion resistance, public verifiability, and double-voting prevention, and discuss its robustness against different adversarial strategies.
- **Practical Open-Source Deployment:** We provide a fully open-source MIT-licensed implementation of our protocol as a library compatible with EVM-based blockchains. All experiments, including deployment on the Sepolia testnet, were conducted using commodity hardware (e.g.,

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

a MacBook Air), demonstrating the protocol's practicality and feasibility for real-world use. Our experimental evaluation confirms both scalability and improved on-chain efficiency, achieving over 20% reduction in gas consumption compared to related work.

Note on Token Burning

While token burning may initially appear costly for users, it is, in practice, highly efficient. A burn transaction is implemented as a standard token transfer with minimal gas costs (21K on EVM). Additionally, the burn amount can be trivially small (e.g., 0.000001 ETH), often costing less than a cent in total. Moreover, token burning can be economically incentivized by removing tokens from circulation. Furthermore, recent proposals, such as EIP-7503 [28], explore the concept of reclaimable burns.

Table I summarizes 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 II reviews the cryptographic foundations underlying our protocol. Section III defines the system and adversary model. Section IV details the protocol design. Section V presents the security and soundness analysis. Section IV-F discusses practical applications and deployment scenarios. Section VII outlines implementation details and reports experimental results. Section VIII reviwes the related works, and Section IX concludes the paper and sketches the directions for future research.

II. Background

This section introduces the core cryptographic building blocks used in our protocol, including commitment schemes, ZKPs, and proof-of-burn mechanism. Table II defines the notations used throughout the paper.

A. Commitment Schemes

Commitment schemes are cryptographic protocols that allow a party to commit to a value while keeping it hidden, with the ability to reveal the value later in a verifiable manner [29].

Definition II.1. A commitment scheme over a message space $\mathcal{M}$ is a triple of algorithms $(\mathsf{Gen},\mathsf{Com},\mathsf{Open})$ such that:

- $pp \leftarrow \mathsf{Gen}(1^\lambda)$: Takes as input the security parameter λ and outputs public parameters pp.
- $(c,d) \leftarrow \mathsf{Com}_{pp}(m)$: For any $m \in \mathcal{M}$, outputs a commitment c and an opening value d.
- $b \leftarrow \mathsf{Open}_{pp}(m,c,d)$: Returns a bit $b \in \{0,1\}$ indicating whether the opening is accepted.

Definition II.2 (Hiding)**.** The commitment should conceal the committed value so that no information about it is leaked prior to the opening. Here it should be computationally hard for the any PPT adversary $\mathcal{A}$, to generate two messages, $m_0, m_1 \in \mathcal{M}$, such that $\mathcal{A}$ can distinguish between their future commitment values:

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

Definition II.3 (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 $\mathcal{A}$:

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

When committing to a sequence of messages rather than a single one, vector commitment (VC) schemes can be employed. These schemes ensure both the integrity and the ordering of the committed values. Among various constructions, Merkle trees are the most widely used and well-understood instantiation of VC schemes. We formally define VC schemes and describe their construction in Appendix A.

B. Non-Interactive Zero-Knowledge Arguments of Knowledge

We follow the original definition from [26] to formally define Non-Interactive Zero-Knowledge Arguments of Knowledge.

Definition II.4. Let $\mathcal{R}$ be a relation generator that, given a security parameter λ in unary, outputs a binary relation R that is decidable in polynomial time. For any pair $(\phi, w) \in R$, we refer to ϕ as the *statement* and w as the *witness*. Let $\mathcal{R}_\lambda$ denote the set of relations that $\mathcal{R}$ may output when given input 1^λ. Now, we define a publicly verifiable non-interactive argument system for $\mathcal{R}$ as a quadruple of probabilistic polynomial algorithms $\Pi = (\mathsf{Setup}, \mathsf{Prv}, \mathsf{Vfy}, \mathsf{Sim})$ such that:

- $(\sigma, \tau) \leftarrow \mathsf{Setup}(R)$: produces the CRS σ and a trapdoor τ for the relation R.
- $\pi \leftarrow \mathsf{Prv}(R, \sigma, \phi, w)$: given that $(\phi, w) \in R$, the algorithm returns an argument π.
- $b \leftarrow Vfy(R, \sigma, \phi, \pi)$: using the same statement ϕ and CRS σ used in generating π, the algorithm Vfy returns $b \in \mathbb{Z}_2$: 0 (reject) or 1 (accept).
- $\pi \leftarrow \mathsf{Sim}(R, \tau, \phi)$: Given the trapdoor τ and statement ϕ, the simulator can return the argument π.

Definition II.5. According to [26], we can label Π as a "*perfect non-interactive zero-knowledge argument of knowledge*" if it satisfies following properties: **Perfect Completeness**: An honest prover with valid witness should convince an honest verifier. Formally:

$$\Pr\left[\mathsf{Vfy}(R, \sigma, \phi, \pi) = 1 \;\middle|\; \begin{array}{c} (\sigma, \tau) \leftarrow \mathsf{Setup}(R) \\ (\phi, w) \in R \\ \pi \leftarrow \mathsf{Prv}(R, \sigma, \phi, w) \end{array} \right] = 1$$

TABLE I: 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 [13]	Multiple ✓	PubBC ✓	HTLP, MPC, ZKP	Contract ✓	✓	Hidden until Tally ✓	✓	Public ✓	✓
Kite [18]	Limited ✗	TA ✗	AHE, ZKP	TA ✗	✓*	TA-based ✓*	✗	TA-based ✓*	✓*
PPVSS [27]	Additive ✓*	HM ✓*	PVSS, MPC, ZKP	MPC ✓*	✗	Encrypted shares ✓	✗	Public ✓	✗
Chaintegrity [15]	Multiple ✓	HM, PubBC ✓*	HE, MPC, ZKP	Trsh. MPC ✓	✓	Encrypted ballots ✓	✗	Public ✓	✗
This Work	Multiple ✓	PubBC ✓	ZKP	Contract ✓	✓	Hidden until Tally ✓	✓	Public ✓	✓

✓* Partially supported or with limitations. * PubBC=Public Blockchain, TA=Trusted Authority, HM=Honest Majority.

TABLE II: Terminology of This work

Notation	Description
H	Any hash function, such as Poseidon [30].
Λ, a	Burn address and token burn amount.
v, w	Vote value and weight.
$\mathsf{V_{id}}, u_{\mathsf{id}}$	Voting session and user identifiers.
$\langle sk, pk\rangle$	secret and public key pair.
$\mathcal{U}, \Delta$	Leaves and the root of the allow-list Merkle tree (Registration phase).
δ_k	Opening (Merkle path) for the k-th value in Δ.
$\mathtt{sr}$	Merkle Patricia Trie (MPT) root.
rlp_Λ	RLP encoding of the address Λ: $\mathsf{rlp}_\Lambda = \mathsf{RLP}$(nonce, balance, storageRoot, codeHash).
β, ξ	Opening values for commitments Λ and $\mathcal{U}$.
η	Nullifier to prevent double voting.
$\pi_{\mathsf{vote}}, \pi_{\mathsf{reg}}$	NIZK proofs for voting and registration.

Definition II.6. Computational Soundness: The Π is sound if it is not possible to prove a false statement. Let L_R denote the language consisting of all statements such as ϕ that have matching witness in R. Therefore, for all PPT adversaries $\mathcal{A}$:

$$\Pr\left[\begin{array}{c}\mathsf{Vfy}(R,\sigma,\phi,\pi)=1,\\ \phi\notin L_R\end{array}\middle|\begin{array}{c}(R,z)\leftarrow\mathcal{R}(1^\lambda)\\ (\sigma,\tau)\leftarrow\mathsf{Setup}(R)\\ (\phi,\pi)\leftarrow\mathcal{A}(R,z,\sigma)\end{array}\right]=\mathrm{negl}(\lambda)$$

Computational Knowledge Soundness: To call Π an "*argument of knowledge*", we require existence of a PPT extractor $\mathcal{X}_\mathcal{A}$ for all PPT adversaries $\mathcal{A}$ that can compute the witness whenever the adversary manages to provide a valid argument:

$$\Pr\left[\begin{array}{c}\mathsf{Vfy}(R,\sigma,\phi,\pi)=1,\\ (\phi,w)\notin R\end{array}\middle|\begin{array}{c}(R,z)\leftarrow\mathcal{R}(1^\lambda)\\ (\sigma,\tau)\leftarrow\mathsf{Setup}(R)\\ ((\phi,\pi),w)\leftarrow(\mathcal{A}\|\mathcal{X}_\mathcal{A})(R,z,\sigma)\end{array}\right]=\mathrm{negl}(\lambda)$$

Definition II.7. Perfect Zero-knowledge: The argument does not leak anything beyond the truth of the statement. We say Π is perfect zero-knowledge if for all adversaries $\mathcal{A}$ with access to z, which is an auxiliary side information derived from relation generator:

$$\Pr\left[\begin{array}{c}\mathcal{A}(R,z,\sigma,\\ \tau,\pi)=1\end{array}\middle|\begin{array}{c}(\sigma,\tau)\leftarrow\mathsf{Setup}(R)\\ (\phi,w)\in R\\ \pi\leftarrow\mathsf{Prv}(R,\sigma,\phi,w)\end{array}\right]=\Pr\left[\begin{array}{c}\mathcal{A}(R,z,\sigma,\\ \tau,\pi)=1\end{array}\middle|\begin{array}{c}(\sigma,\tau)\leftarrow\mathsf{Setup}(R)\\ (\phi,w)\in R\\ \pi\leftarrow\mathsf{Sim}(R,\tau,\phi)\end{array}\right]$$

C. Proof-of-Burn

Proof-of-Burn (PoB) [19] 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. PoB has been applied in various settings, including anonymous transaction pools (EIP-7503[28]) and trustless bridges between blockchains. Before defining the PoB construction, we first formalize the standard address protocol of public blockchains as follows:

Definition II.8. The standard address protocol of public blockchains, such as Ethereum or Bitcoin, consists of the following three functions:

- $(sk, pk) \leftarrow \mathsf{GetKeyPair}(1^\lambda)$: Generates a standard asymmetric key pair using the security parameter λ.
- $\Lambda \leftarrow \mathsf{GetStdAddr}(pk)$: A deterministic one-way function that derives a unique address Λ from pk.
- $b \leftarrow \mathsf{VfySpnd}(\mathsf{trx}, \sigma)$: Verifies whether the sender address of transaction trx (i.e., $\mathsf{trx.from}$) matches the address derived from the public key recovered from the signature σ. Formally, it checks whether:
$$\mathsf{trx.from} \stackrel{?}{=} \mathsf{GetStdAddr}(\mathsf{RecoverPK}(\mathsf{trx}, \sigma)).$$

Definition II.9. PoB mechanism can be constructed over a *perfectly hiding* and *computationally binding* commitment scheme such as $\Pi_{\mathsf{com}} = (\mathsf{Gen}, \mathsf{Com}, \mathsf{Open})$. A PoB system consists of the following algorithms:

- $(\Lambda, \beta) \leftarrow \mathsf{GenBrnAddr}(f, t) = \Pi_{\mathsf{com}}.\mathsf{Com}(f(t))$: Generates a burn address Λ using a function f and input tag t. Λ as a commitment for message $m = f(t)$. The β acts as the opening value for the commitment.
- $b \leftarrow \mathsf{VfyBrnAddr}(\Lambda, f, t, \beta) = \Pi_{\mathsf{com}}.\mathsf{Open}(f(t), \Lambda, \beta)$: Verifies that the burn address Λ corresponds to the tag t under function f. This can be viewed as openning Λ with message $m = f(t)$ and opening value β.

Following [19], the construction in Definition II.9 satisfies the following properties:

- **Unspendability:** No one should be able to spend tokens sent to the burn address. That is, under the standard blockchain address protocol (Definition II.8), for all PPT adversaries $\mathcal{A}$, the probability of finding a private and public key pair (sk, pk) such that the resulting address is a valid burn address is negligible:
$$\Pr\left[\begin{array}{c}\mathsf{VfyBrnAddr}(\Lambda,f,t,\beta)=1\\ \wedge\,\Lambda=\mathsf{GetStdAddr}(pk)\end{array}\middle|\begin{array}{c}\Lambda,\beta,t,sk\leftarrow\mathcal{A}\\ pk\leftarrow\mathsf{PubKey}(sk)\end{array}\right]=\mathrm{negl}(\lambda)$$
- **Binding:** The burn address must be uniquely tied to a particular tag, preventing a user from producing different interpretations of the same address. This follows from the binding property of the underlying commitment scheme.

Formally, for all PPT adversaries $\mathcal{A}$, the probability of successfully verifying the same burn address Λ under two distinct tags $t_0 \neq t_1$ is negligible:

$$\Pr\left[t_0 \neq t_1 \;\middle|\; \begin{array}{c} \Lambda, t_0, t_1, \beta_0, \beta_1 \leftarrow \mathcal{A}(pp) \\ \mathsf{VfyBrnAddr}(\Lambda, f, t_0, \beta_0){=}1 \\ \mathsf{VfyBrnAddr}(\Lambda, f, t_1, \beta_1){=}1 \end{array} \right] = \mathsf{negl}(\lambda)$$

- **Hiding / Uncensorability:** The hiding property of the underlying commitment scheme implies uncensorability. Specifically, a burn transaction should be computationally indistinguishable from a regular transaction. This requires that the burn address generated by GenBrnAddr be indistinguishable from a standard spendable address generated by GetStdAddr, unless the user explicitly reveals the associated tag. Formally, the following two distributions are computationally indistinguishable to any PPT adversary:

$$\left\{ \Lambda \;\middle|\; \begin{array}{c} t \leftarrow\!\$\, \mathcal{T} \\ (\Lambda,\beta)\leftarrow \mathsf{GenBrnAddr}(f,t) \end{array} \right\} \sim \left\{ \Lambda \;\middle|\; \begin{array}{c} (sk,pk)\leftarrow \mathsf{GetKeyPair}(1^\lambda) \\ \Lambda \leftarrow \mathsf{GetStdAddr}(pk) \end{array} \right\}$$

III. Voting System and Adversary Model

In this section, we formally define the system model and the roles of each actor in our protocol. We then specify the adversary model, including the computational capabilities and limitations. Finally, we outline the desired security guarantees with respect to our system and adversary assumptions.

A. Actors

The *Burn-to-Vote* protocol consists of four actors: *voters*, *organizer(s)*, a public *blockchain*, and the *voting contract*.

- **Voters:** interact with blockchain and voting contract. Each voter may control multiple unlinked/linked addresses; however, we assume that each individual voter can be authenticated only once. Therefore, when the voting scheme restricts each individual to a single vote, the registration phase allocates only one slot per voter.
- **Organizer:** is responsible for authenticating voters during the registration phase and setting up the voting ceremony by interacting with the contract.
- **Blockchain:** in our scheme, the blockchain acts as the public bulletin board. Voters must be able to interact freely with the blockchain and broadcast transactions.
- **Voting Contract:** serves as the autonomous entity responsible for managing the entire voting ceremony, including tallying and vote verification. The contract is not trusted in any traditional sense and its computations are publicly verifiable. It does not require any private state, and all operations are performed over public state.

Unlike the centralized architecture of traditional voting systems [31] or those relying on vote encryption [32], [24], [8], [33], our protocol does not require any *voting servers*, *trustees*, or *tallying servers*. In contrast, the *voting contract* verifies submitted votes and tallies results autonomously.

B. Threat and Adversary Model

We consider a standard PPT adversary model, consistent with prior work such as [31]. The adversary has full control over the communication network, allowing it to observe, intercept, modify, or inject arbitrary messages. Despite this capability, we assume that the underlying cryptographic primitives—Collision-Resistant Hash Functions, Digital Signatures, zkSNARKs (e.g., Groth16)—remain secure against any PPT adversary.

We also assume a public, permissionless blockchain that offers standard guarantees: ledger immutability, consensus-based state agreement, and trustless, deterministic execution of publicly verifiable smart contracts. Malformed or adversarial blocks are rejected by the consensus protocol, ensuring consistency and integrity.

Definition III.1 (Coercion)**.** We assume that the adversary may attempt to coerce voters into voting in certain ways. Under coercion, the voter fully complies and the adversary learns all information received or stored by the voter during the coercion period. Following previous work [31], [34], [35], we assume that after coercion ends (but before the voting deadline), there is a time window during which the adversary no longer controls the voter, allowing the voter to cast additional unlinkable burn transactions, freely. We further assume that the voter's means of authentication used during registration (i.e., α_{info}) is inalienable. If it were transferable, the adversary could impersonate the voter entirely and independently.

C. Security Guarantees

The proposed e-voting scheme should satisfy a set of desirable security properties.

Definition III.2 (Security Guarantees of the Voting Protocol)**.** A secure e-voting protocol must satisfy the following:

- **Eligibility:** Only eligible voters can participate in the voting. Any attempt by ineligible ones must be rejected.
- **Vote Uniqueness:** Each eligible voter can cast at most one effective vote. Any attempt at double voting must be rejected.
- **Vote confidentiality:** The contents of a voter's ballot remain secret from all other parties, including the organizer, contract, and other voters.
- **Voter-Vote Unlinkability:** It is computationally infeasible to link a vote to the identity or registration data of the voter based on on-chain transaction data.
- **Coercion Resistance:** After any coercion attempt, the adversary should not be able to determine whether the coerced voter complied.
- **End-to-End Verifiability and Tally Integrity:** Voters can verify that their vote was recorded/included correctly, while any observer can independently verify correctness of the overall tally. The tally must reflect only the latest valid votes from eligible voters and is computed through transparent, publicly verifiable processes.

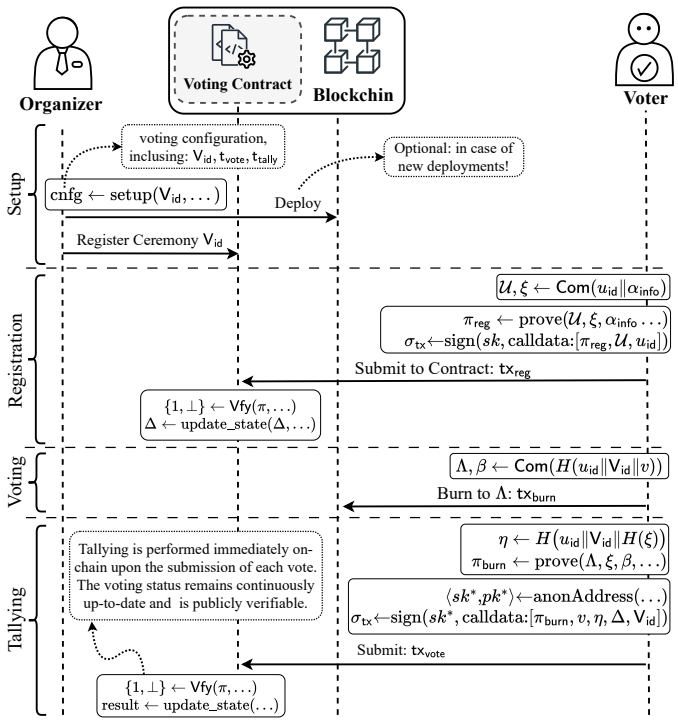

Fig. 1: Protocol Overview

- **Vote Validity:** Only syntactically well-formed and semantically valid votes submitted by authenticated voters are accepted. Malformed or duplicate submissions are rejected during verification.
- **Censorship Resistance:** The voting protocol should not depend on a central authority or trusted intermediary to relay or include votes. Any voter with access to the blockchain should be able to submit their transactions directly, without risk of being blocked or ignored.

IV. Burn to Vote Protocol

We begin this section by presenting a high-level overview of the protocol and the role of each actor in its respective phase. We then provide a detailed breakdown of each phase.

A. High-Level Overview

Figure 1 illustrates the four phases of the protocol: *setup*, *registration*, *voting* (which includes token burning and proof generation), and *tallying*.

- **Setup Phase:** The *organizer* sets the voting parameters, such as the available options to vote for, voting duration (t_{vote}: voting period, t_{tally}: tally period, can be set based on block numbers), and the statement for the proof-of-burn circuit. The *organizer* may either deploy a new contract or configure an existing one.
- **Registration Phase:** Each voter locally generates a commitment using their unique public identifier and some auxiliary data that binds their digital identity to a real-world identity. This commitment is submitted along with a ZKP attesting to its correct formation (Equation 2). Finally, the *organizer*, or the contract itself (operating autonomously), aggregates all valid commitments into a vector commitment structure (e.g., a Merkle tree) and registers the root of this structure on-chain.
- **Voting Phase:** Each voter locally computes a unique, unspendable burn address and transfers a specified amount of tokens to it.
- **Tallying Phase:** The voter generates a ZKP attesting to the correctness of the burn and to their eligibility, and submits this proof along with their vote to the smart contract. 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. In contrast to prior approaches [13], [15], which restrict anonymity to a fixed set of pre-registered voters, our mechanism supports an anonymity set that includes all externally owned accounts (EOAs) involved in any token transfer during the voting period. Even after the registration phase, the actual number of participating voters remains hidden, as voting occurs through indistinguishable burn transactions. This design significantly enhances voter privacy and resists adversarial attempts to infer voter participation. Importantly, voters are not required to interact with the blockchain directly; instead, they may delegate the burn transaction to any relayer or even a centralized exchange (CEX). A notable benefit of this approach is that neither the relayer nor the CEX can distinguish burn transactions from standard token transfers (see Definition II.9: *hiding/uncensorability*). This flexibility is especially valuable in large-scale voting scenarios, where many participants may rely on custodial wallets or centralized platforms to perform the burn transaction.

B. Setup Phase

During the setup phase, the *organizer* either deploys a new smart contract or configures an existing one that supports multiple instances of similar voting ceremonies. The configuration parameters may include:

- **Voting ID:** A unique identifier of this voting instance.
- **Voting Options:** The set of choices available for voters.
- **Registration and Voting Period:** Timing parameters for the voting process, including the start/end block numbers for *registration*, *voting*, and *tallying* phases.
- **Proof-of-Burn Statement:** A formal statement defining the logic and constraints of the voting. In practice, this is realized by linking the voting contract to a `zk_verifier` contract, which is compiled with the verification key corresponding to the desired *proof-of-burn* statement.
- **[Optional] Reusing Prior Registration:** Instead of initiating a new registration process, the organizer may specify a precomputed Merkle root corresponding to an allowlist of eligible voters derived from a previous registration phase.

C. Registration Phase

In this section, we show how even the most transparent form of registration can be used anonymously in our voting scheme. To that end, we present the most open and non-anonymous registration method possible: voters publicly reveal their identities and use verified, KYC-compliant wallets to register. This also applies to cases where only holders of certain on-chain NFT collections (and therefore completely exposed addresses) are allowed to vote, e.g., in *Nouns DAO*, where only *Nouns* NFT holders can participate in governance [36]. During registration, each voter commits to their identity as follows:

$$(\mathcal{U}, \xi) \leftarrow \mathsf{Com}(u_{\mathsf{id}} \,\|\, \alpha_{\mathsf{info}}) \tag{1}$$

Here, u_{id} denotes the public user identity, and α_{info} is auxiliary information required for KYC and authentication. This data is authenticated via the function f_{auth}, which depends on contextual factors, such as proof of possession of a valid passport [37]. The user submits the commitment $\mathcal{U}$ to the smart contract, along with a proof π_{reg} that reveals their identity as a public input for the following relation:

$$\begin{aligned}\mathcal{R}_{\mathsf{reg}} := \big\{ \big((\mathcal{U}, u_{\mathsf{id}}), (\alpha_{\mathsf{info}}, \xi)\big) : \\ f_{\mathsf{auth}}(\alpha_{\mathsf{info}}) = 1 \ \wedge \ \mathsf{Open}\big((u_{\mathsf{id}} \,\|\, \alpha_{\mathsf{info}}), \mathcal{U}, \xi\big) = 1 \big\}\end{aligned} \tag{2}$$

After collecting all submissions from eligible users, a VC (e.g., a Merkle tree) can be constructed over the list of valid commitments, each accompanied by a valid proof π_{reg}. The exact construction of the VC is beyond the scope of this work, and we note that the proposed voting method is agnostic to the choice of VC structure. For the remainder of the paper, we assume that a simple Merkle tree is built using these commitments as its leaves, which serves as an allow-list. We also adopt the classical setting in which each individual is permitted to vote at most once[3].

D. Voting Phase

We begin by defining the core burning mechanism, namely *Burn-to-Vote*, in which voters commit to a specific vote as part of the burn transaction. We then show how, with a minimal modification to the burn process, voters can delegate their vote to a designated trustee (*Burn-to-delegate* mechanism). Next, we describe the details of the deterministic nullifier derivation, which can later be proven within a ZK statement. Finally, we formally define the NIZK relation that the voter must satisfy as their valid *proof-of-burn* to authenticate their vote submission in the tally phase.

Definition IV.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. We use the same construction from Definition II.9. Here, the tag t consists of the following values:

- 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.

The inner function f can be defined as a simple concatenation and hashing function, e.g., $f(t) = H(u_{\mathsf{id}} \,\|\, \mathsf{V}_{\mathsf{id}} \,\|\, v)$ (where H is a random oracle). Accordingly, the $\mathsf{GenBurnAddress}$ used for voting is defined as:

$$(\Lambda, \beta) \leftarrow \mathsf{Com}(H(u_{\mathsf{id}} \,\|\, \mathsf{V}_{\mathsf{id}} \,\|\, v)) \tag{3}$$

Depending on the application, some components of the tag t may be omitted, set to null values, or extended with additional fields based on the requirements of the voting scheme. For example, if voter eligibility is not restricted, u_{id} may be set to a null or default value.

Definition IV.2. Burn-to-Delegate: In this mechanism, the voter's choice is not embedded in the computation of the burn address. Instead, the voter irreversibly uses their eligibility to participate in the voting ceremony without committing to any specific vote. The resulting *proof-of-burn* can then be transmitted to a designated trustee, who casts a vote on behalf of the voter. In this setting, the actual vote is cast at the vote submission transaction, where the transaction sender provides the *proof-of-burn* and specifies the voting choice[4]. The burn address Λ' is computed as follows:

$$(\Lambda', \beta) \leftarrow \mathsf{Com}(H(u_{\mathsf{id}} \,\|\, \mathsf{V}_{\mathsf{id}})) \tag{4}$$

where, u_{id}, V_{id} and β are same as previous mechanism.

Definition IV.3. Nullifier Construction. The nullifier is a critical component of the protocol that ensures a single registered entity cannot use multiple burn addresses to cast multiple votes. It prevents double voting under the same identity while fully preserving voter anonymity. To achieve this, the nullifier must be uniquely tied to: (1) the user identity, (2) the registration ceremony, (3) and the voting ceremony. At the same time, it must not reveal any information about, or link to, the corresponding burn address Λ or the user's commitment $\mathcal{U}$ submitted during registration. Such a nullifier can be constructed using any *collision-resistant* hash function H under our adversarial model, as follows:

$$\eta = H(u_{\mathsf{id}} \,\|\, \mathsf{V}_{\mathsf{id}} \,\|\, H(\xi)) \tag{5}$$

[3]As discussed in Section IV-F, the registration phase may vary significantly depending on the application. For instance, in economically incentivized scenarios (e.g., promoting a specific token), the registration phase may be entirely omitted. In such cases, eligibility is determined solely by the act of burning the token, allowing a single entity to burn multiple times and thereby cast multiple votes.

[4]We remark that the delegated party discovers v and some u^*_{id} (which may leak partial information about the original u_{id}) for each party. Nonetheless, the associated α_{info} remains hidden, protecting the real identity of the users.

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 ξ is the opening values corresponding to $\mathcal{U}$.

Proof Generation. After burning tokens, user generates a *proof-of-burn* to demonstrate following properties:

- **User Eligibility:** The voter can successfully prove membership in the VC generated during the registration phase by opening a valid commitment $\mathcal{U}$ corresponding to their ID u_{id} and registration witness ξ.
- **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 $a > 0$. 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 positive balance 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 following the Definition IV.3 Equation 5 using the exact opening values ξ for commitment $\mathcal{U}$, thereby binding the nullifier uniquely to the user identity without leaking identifying information.
- **Vote Binding:** 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 relation $\mathcal{R}_{\mathsf{vote}}$ that a voter must prove they hold the correct statement-witness pair to 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.

Definition IV.4. Let β be the opening value of the burn address Λ, u_{id} the voter's unique identifier, V_{id} the voting ceremony identifier, and v the vote value. Let $\mathtt{sr}$ 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). Let δ_k denote the opening (e.g., Merkle path) of the k-th commitment (i.e., $\mathcal{U}$) in the VC Δ corresponding to the prover, who holds the opening value ξ. The prover claims knowledge of values that the pair $\big(\phi, w\big)$ with the statement $\phi = (\mathtt{sr}, \eta, \mathsf{V}_{\mathsf{id}}, v, \Delta)$ and witness $w = (\xi, \Lambda, \pi_\Lambda, \mathsf{rlp}_\Lambda, \mathcal{U}, k, \delta_k, \alpha_{\mathsf{info}}, \beta, u_{\mathsf{id}}, a)$ belongs to the relation $\mathcal{R}_{\mathsf{vote}}$ as follows:

$$
\begin{aligned}
&\mathcal{R}_{\mathsf{vote}} := \\
&\Big\{\big((\mathtt{sr}, \eta, \mathsf{V}_{\mathsf{id}}, v, \Delta), (\xi, \Lambda, \pi_\Lambda, \mathsf{rlp}_\Lambda, \alpha_{\mathsf{info}}, \mathcal{U}, k, \delta_k, \beta, u_{\mathsf{id}}, a)\big) : \\
&\quad \mathsf{Open}\big((u_{\mathsf{id}} \,\|\, \alpha_{\mathsf{info}}), \mathcal{U}, \xi\big) = 1 \ \wedge \ \mathcal{V}_\Delta(\Delta, \mathcal{U}, k, \delta_k) = 1 \\
&\quad \wedge \ \mathsf{Open}\big(H(u_{\mathsf{id}} \,\|\, \mathsf{V}_{\mathsf{id}} \,\|\, v), \Lambda, \beta\big) = 1 \ \wedge \ a > 0 \wedge \\
&\quad \mathsf{rlp}_\Lambda = \mathsf{rlp}(0, a, \phi, \phi) \ \wedge \ \eta = H(u_{\mathsf{id}} \| \mathsf{V}_{\mathsf{id}} \| H(\beta)) \\
&\quad \wedge \ \mathcal{V}_{MPT}(\mathtt{sr}, \pi_\Lambda, \Lambda, \mathsf{rlp}_\Lambda) = 1\Big\} \quad (6)
\end{aligned}
$$

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

E. Tally

After the voting phase concludes, users submit their votes along with the corresponding burn proofs to the smart contract for verification. Each vote submission transaction includes the vote value v, a zero-knowledge proof π_{vote} attesting to the relation $\mathcal{R}_{\mathsf{vote}}$ (see Definition IV.1), and the required public inputs (e.g., η, V_{id}, $\mathtt{sr}$, etc.). Upon receiving a vote submission, the smart contract performs the following checks:

1) Verifies that the nullifier has not been previously used (to prevent double voting), records it as spent, and confirms that it belongs to the registered vector commitment (Δ) from the registration phase.
2) Calls the verifier contract to validate the burn proof (Definition IV.4).
3) Checks the vote value v to comply with the rules of the voting (e.g., valid option, range, or binary value).
4) If the `revote` flag is enabled, permits overwriting a previous vote using the same nullifier.
5) Verifies that the vote is submitted within the permitted voting period.

Since votes are submitted in plaintext, the smart contract can directly apply the tally logic as each vote is verified. This enables support for a wide range of autonomous and fully trustless tallying strategies, which we discuss next.

F. Overview of Supported Voting Scenarios

Across all voting scenarios, the core components of our protocol (burn address generation, nullifier derivation, and vote submission) remain unchanged, ensuring modularity and extensibility. Each supported model relies on the same *proof-of-burn* and a unique nullifier. Variations appear only in vote integrity checks (e.g., range assertions), which impose negligible overhead on prover performance, since the majority of circuit constraints are consumed by verifying the burn address's state in blockchain (e.g., MPT inclusion on the EVM).

Our plaintext-based tally design directly supports a wide range of voting models that are difficult or infeasible under encryption-based approaches. In contrast, systems such as MACI [12] and Kite [18] embed voting semantics within

TABLE III: Supported Voting Schemes

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

* B2D = Burn-to-Delegate, B2V = Burn-to-Vote, Both = supports both.
+ Perm. = permutation; Arbitrary = string/number. ♦ Set of allowed vote values: m: Number of candidates, w: Maximum voting power, $\pi(S)$: Set of permutations of S. ○ Wght. = Weighted, Freq. Cnt. = frequency count.

encrypted payloads or rigid contract logic, which restricts their adaptability.

Table III summarizes the supported models and their associated costs. Concretely, our protocol supports: (1) Majority Voting (First-Past-the-Post), (2) Token-Weighted Voting, (3) Quadratic Voting [24], (4) Ranked-Choice Voting (RCV), and (5) Open-Content Voting. The latter allows voters to submit arbitrary content (strings, numbers, or free-form textual proposals) thus generalizing majority voting to arbitrary data values. To the best of our knowledge, this is the first blockchain-based design to enable such a flexible voting mechanism. Detailed descriptions of each model are provided in Appendix E.

V. Soundness and Security

In this section, we demonstrate that the proposed voting protocol satisfies the desired soundness and security properties. To this end, we first justify the *hiding* and *binding* guarantees of our burn address constructions with respect to the security of the employed commitment scheme. We then reduce the knowledge soundness of the NIZK relations $\mathcal{R}_{\mathsf{reg}}$ and $\mathcal{R}_{\mathsf{vote}}$ to that of the underlying ZKP system.

Theorem 1. *The proposed method of generating a burn address satisfies both the* hiding *and* binding *properties, given that the underlying commitment scheme is secure. Specifically, let H be a collision-resistant hash function. Then the commitment function* Com *can be constructed as follows:*

$$\mathsf{Com}(m):\ \beta \leftarrow\!\$\ \mathbb{Z}_q, \quad \Lambda \leftarrow H(m\|\beta).$$

Proof sketch. **Hiding:** A burn address is indistinguishable from a standard address unless the tag is known since both types of addresses are generated by applying a random oracle to unpredictable inputs (tags for burns, public keys for standard addresses), making them identical to any observer without knowledge of the secret values. Therefore, no computational adversary can distinguish or censor burn transactions, achieving uncensorability/hiding. The security relies on the random oracle model and the unpredictability of the underlying inputs. The formal proof is identical to the proof of Theorem 4 in [38] therefore we defer to it for details.

Binding: The *binding* property is guaranteed by the binding property of the commitment, indeed it must be computationally infeasible to find two distinct inputs that produce the same burn address. Let us assume that there exists an adversary $\mathcal{A}$ that breaks the binding property of the burn address with non-negligible probability ϵ. We define an adversary $\mathcal{B}$ that using $\mathcal{A}$ breaks the binding of Com with the same probability ϵ. We use the following experiment executed between the challenger and $\mathcal{B}$:

$$\Pr\left[m_0 \neq m_1 \;\middle|\; \begin{array}{c} pp \leftarrow \mathrm{Gen}(1^\lambda) \\ \mathsf{ret} = (\Lambda, (u_{\mathsf{id}:0}\|\mathsf{V}_{\mathsf{id}:0}\|v_0), (u_{\mathsf{id}:1}\| \\ \mathsf{V}_{\mathsf{id}:1}\|v_1), \beta_0, \beta_1) \leftarrow \mathcal{A}(pp) \\ (\Lambda, m_0, m_1, \beta_0, \beta_1) \leftarrow \mathcal{B}(pp, \mathsf{ret}) \\ \mathsf{Open}(\Lambda, m_0, \beta_0) = 1 \\ \mathsf{Open}(\Lambda, m_1, \beta_1) = 1 \end{array} \right] = \epsilon$$

where $m_0 = H(u_{\mathsf{id}:0}\|\mathsf{V}_{\mathsf{id}:0}\|v_0)$ and $m_1 = H(u_{\mathsf{id}:1}\|\mathsf{V}_{\mathsf{id}:1}\|v_1)$. This captures the probability that if $\mathcal{A}$ can find two distinct openings such that both are accepted as valid openings of the same burn address Λ then $\mathcal{B}$ can exploit it to break the Com binding property using that values in the messages. If ϵ is non-negligible, it would violate the binding property: $\mathsf{Com}(H(u_{\mathsf{id}}\|\mathsf{V}_{\mathsf{id}}\|\beta_0\|v_0)) = \mathsf{Com}(H(u_{\mathsf{id}}\|\mathsf{V}_{\mathsf{id}}\|\beta_1\|v_1))$. Therefore, under our assumptions, adversary's advantage ϵ is negligible. □

Theorem 2. *The probability that a PPT adversary successfully breaks the soundness of the statement in Definition IV.4 is negligible.*

Proof sketch. To break the soundness of the statement in Definition IV.4, 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 rlp_Λ, 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 $\mathtt{sr}$. 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 succeeds in opening the commitment Λ' without prior knowledge of its corresponding opening value β'. Successfully doing so would violate the *hiding* property of the underlying commitment scheme, contradicting our assumptions. (3) Finally, the adversary may attempt to bypass the nullifier mechanism and vote more times than allowed. This would imply that they are able to open the same commitment $\mathcal{U}'$ or the aggregated VC created during the registration phase Δ using multiple opening values, which violates the *binding* property of $\mathcal{U}$ or the *position-binding* of the Δ.

We calculate the overall probability of the adversary's successfully breaking the soundness of the statement in Defi-

nition IV.4 as follows:

$$\Pr\left[\begin{array}{c|c} \mathsf{Vfy}(R,\sigma,\phi,\pi)=1 \wedge \\ \Big(\mathsf{Open}\big((u_{\mathsf{id}}\|\alpha_{\mathsf{info}}),c_{\mathsf{reg}},\xi\big)\neq 1 \vee \\ \mathcal{V}_\Delta(\Delta,\mathcal{U},k,\delta_k)=0 \ \vee \ a=0 \vee \\ \mathsf{Open}\big(H(u_{\mathsf{id}}\,\|\,\mathsf{V}_{\mathsf{id}}\,\|\,v),\Lambda,\beta\big)\neq 1 \\ \vee\ \mathsf{rlp}_\Lambda \neq \mathsf{rlp}(0,a,\phi,\phi) \\ \vee\ \eta \neq H(u_{\mathsf{id}}\|\mathsf{V}_{\mathsf{id}}\|H(\beta)) \vee \\ \mathcal{V}_{MPT}(\mathtt{sr},\pi_\Lambda,\Lambda,\mathsf{rlp}_\Lambda)\neq 1\Big) \end{array} \middle| \begin{array}{c} (R,z)\leftarrow\mathcal{R}(1^\lambda) \\ (\sigma,\tau)\leftarrow\mathsf{Setup}(R) \\ ((\phi,\pi),w)\leftarrow(\mathcal{A}\|\mathcal{X}_{\mathcal{A}})(R,z,\sigma) \\ w:(\xi,\Lambda,\pi_\Lambda,\mathsf{rlp}_\Lambda,\alpha_{\mathsf{info}},\mathcal{U}, \\ k,\delta_k,\beta,u_{\mathsf{id}},a) \\ \phi:(\mathtt{sr},\eta,\mathsf{V}_{\mathsf{id}},v,\Delta) \end{array}\right] = \mathrm{negl}(\lambda)$$

□

Theorem 3. *Under the coercion setting defined in Definition III.1, the* burn-to-vote *mechanism is resistant to coercion.*

Argument. The *proof-of-burn* mechanism enables submitting multiple unlinkable votes. If a voter is coerced to cast a specific vote, they may comply by executing a burn transaction that commits to the coercer's preference. However, under our assumption (Definition III.1) the coercion period ends before the voting period concludes. The voter can subsequently generate a new burn address (based on their true preference) and burn tokens to it. Due to the unlinkability of burn address and the non-revealing nature of π_{vote}, the adversary cannot determine which vote was ultimately counted, or whether the voter complied with coercion. Therefore, for any coercion strategy, the adversary's advantage in distinguishing whether a voter complied is negligible. □

Receipt-freeness: This is a stronger notion than coercion resistance. Our system does not achieve full receipt-freeness. Since votes are publicly verifiable upon submission, voters can construct partial receipts, which may be misused in scenarios where a voter wishes to prove how they voted, even without revealing their full identity. Preventing such behavior would require additional protocol mechanisms that obfuscate the link between the vote and the voter, even post-submission. This could involve incorporating a trusted or a trustless MPC-based third party, into the vote verification process. We leave the design of such extensions to future work.

VI. Details of the Supported Voting Scenarios

1) Majority Voting (First-Past-the-Post): This is a binary majority voting system in which the outcome is determined by whether the number of affirmative votes exceeds 50%. It is implemented as an equal-weight scheme, where the amount of tokens burned does not affect the voting weight. Let N denote the total number of voters. The tally is as follows:

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

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

2) Token-Weighted Voting: Each voter casts a vote for exactly one candidate by burning a chosen amount of tokens. The vote weight assigned to a candidate is proportional to the number of tokens burned. In their $\mathcal{R}_{\mathsf{vote}}$, voters prove that the balance of the burn address Λ is $a > \tau_i$, where τ_i is part of the public statement ϕ. The tally is computed as follows:

$$\forall\, k \in [1,m], \quad \text{score}_k = \sum_{i=1}^{N} \tau_i \times \delta_{k,v_i}$$

where $\delta_{k,j}$ is the *Kronecker delta* function, which returns 1 when $k = j$ and 0 otherwise.

3) Quadratic Voting [24]: This scheme allows voters to express the intensity of their preferences by allocating votes such that the cost increases quadratically with the number of votes cast. Let each voter's ballot be represented as a vector $B_i = (b_{i,1}, \ldots, b_{i,m})$, where the following holds:

$$\forall i \in \mathbb{Z}_n : \ \sum_{j=1}^{m} b_{i,j}^2 \le \tau_i, \quad \forall j \in \mathbb{Z}_m : \ \text{score}_j = \sum_{i=1}^{N} b_{i,j},$$

where τ_i denotes the budget of voter i, determined by the amount of tokens they have burned.

4) Ranked-Choice Voting (RCV) voters rank candidates in order of preference. Each voter submits a ranking vector R_i, with $R_{i,0}$ in the top choice, $R_{i,1}$ the second choice, and so on. Two common tally methods are:

- **Borda Count [39], [40]:** Each candidate receives points inversely proportional to their ranking position:

$$\forall j \in \mathbb{Z}_m \mid \text{score}_j = \sum_{i=1}^{N} (m - \mathsf{R}_i.\mathsf{IndexOf}(j)).$$

- **Instant-Runoff Voting (IRV) [41]:** 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.

5) Open-Content Voting: This scheme allows voters to submit votes containing **arbitrary content**, such as strings, numbers, or textual proposals—essentially, any data the voter wishes to express. It is suitable for open-ended decisions, such as selecting a protocol name, slogan, or other customizable inputs where predefined options may be insufficient. If the vote value exceeds the size of a field element (e.g., larger than 254 bits), the voter may instead use a hash $H(v)$ in the construction of Λ. During the vote submission phase, the contract can compute $H(v)$ on-chain and verify it against the submitted proof π_{vote} in a trustless manner. This scheme can be viewed as a generalization of majority voting, extended to support arbitrary vote values. To the best of our knowledge, this is the first design to enable such a voting mechanism in a blockchain setting.

VII. Implementation

In this section, we first overview the proof generation process and circuit-level complexity. We then describe 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

TABLE IV: 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.

TABLE V: Performance Metrics

Operation	Time (s)	Size (MB)	Gas Cost (k)
Circuit Compilation	$1{,}500 \pm 100$	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

costs for verification and tallying, and compare our results with the state-of-the-art voting protocols.

A. Proof Generation

Following the relation defined in Definition IV.1, voters generate a proof attesting to three key properties: (1) the correctness of the burn address Λ and the corresponding nullifier η; (2) the inclusion of the burn address with a positive balance in the latest block's state root, i.e., a valid MPT proof; and (3) the voter's membership in Δ via a Merkle proof.

Burn Proof Statement: To cast a valid vote, user must prove knowledge of ξ, δ_k such that $\mathcal{V}_\Delta(\Delta, \mathcal{U}, k, \delta_k) = 1$ with the Merkle tree root Δ, and that the burn address Λ satisfies the following conditions within the Ethereum state trie root `sr` at the last voting block: (1) The leaf node for Λ must contain a $\langle$`key`,`val`$\rangle$ pair with `key` being the remaining nibbles of $\mathsf{keccak}(\Lambda)$ after the shared prefix (k_u) and `val` with $\mathsf{rlp}_\Lambda = \mathrm{RLP}(0, \texttt{balance}, \texttt{storageRoot}, \texttt{codeHash})$. (2) For each node in the path from the leaf to the root we compute $h_i = \mathsf{keccak}(\mathrm{RLP}(node_i))$ and verify that h_i matches the corresponding child reference in the parent node. (3) Confirm that the final hash h_n equals the state root `sr`. (4) Enforce that the recovered account state satisfies:$\texttt{nonce} = 0$, $\texttt{balance} > 0$, $\texttt{storageRoot} = \texttt{NULL}$, $\texttt{codeHash} = \texttt{NULL}$, ensuring the address is unused and has no private key or deployed code.

Circom Implementation: We implemented the above statement in `Circom`, as formalized in Definition IV.4. The circuit verifies vote validity and generates a nullifier uniquely tied to the voter's secret and burn address. Table IV reports the constraint counts for our circuit embedding RLP and MPT logic. A simplified version of the `Circom` implementation is provided in Appendix B.

B. Voting Smart Contract

For each voting ceremony, the appropriate voting contract is instantiated by a factory (generator) contract using the `CREATE2` opcode. Depending on the selected voting scheme (e.g., majority voting, quadratic voting), the factory deploys a specialized implementation accordingly. We implemented all core functionalities of the voting contract in Solidity, including `submitVote`, `submitRevote`, and `tally`. Appendix D provides details of the contract's algorithms (Algorithms 1, 2, 3, and 4). To evaluate the system using real-world data, we deployed[5] the implemented contract system on the Sepolia testnet. We also successfully executed the `submitVote`[6] and `tally`[7] transactions on-chain.

C. 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 benchmark its performance against existing solutions. Since our system uses Groth16, both the proof size and verification time remain constant, regardless of the number of votes or the specific voting scenario. On the other hand, the prover complexity is also low, allowing the prover to run on a lightweight laptop (e.g., MacBook Air). Table V reports the performance metrics of our prototype implementation.

In terms of on-chain costs, the contract performs a single Groth16 proof verification and updates the vote tally accordingly. Specifically, the `submitVote` transaction consumes approximately 300k gas, which is significantly lower than in related work [13], [18]. The final `tally` transaction incurs a cost of only around 50k gas, making the protocol highly efficient. Table VI compares the gas costs of our design to those of [18] and [13] in a simple Yes/No voting scenario. Our protocol achieves up to 25% lower gas consumption compared to Cicada [13], and up to 21% lower compared to Kite [18].

To further demonstrate the scalability of our system, we evaluate its performance against Cicada [13] 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, while the cost of [13] increases sharply and becomes impractical at scale.

The primary reason for our improved efficiency is that the on-chain logic consists of simple vote count aggregation after the voting deadline. Notably, our design can support over one million voters with tally costs equivalent to those in a

[5]Deployed contract: https://sepolia.etherscan.io/address/0xbf36ae91d220ca3f387ee63979eb433ed9b1d0be#code

[6]`submitVote`: https://sepolia.etherscan.io/tx/0x7acda7e3677372246dd523e6fd039f61110f7f8d970af36606bbc30dd60f08cf

[7]`tally` transaction: https://sepolia.etherscan.io/tx/0xeb07a4fae6115a6eaba390aedac47dfcc67454df5b95db38667ee1f6c460db46

TABLE VI: Yes/No Voting Gas Cost Comparison

System	Gas Cost	Δ vs. Cicada [13]	Δ vs. Kite [18]
This work	312 927	-25.2 %	-21.0 %
Kite [18]	396 323	-5.3 %	—
Cicada [13]	418 358	—	+5.6 %

setting with just 100 voters. Furthermore, the gas cost of vote submission remains constant, regardless of the number of votes or candidates. Table VII summarizes the qualitative comparison between our design and the other two schemes.

VIII. Related Work

Table I summarizes key trade-offs among different e-voting protocols. Compared to rest, our protocol eliminates the need for off-chain coordinators or dealers and supports fully autonomous on-chain tallying performed by the smart contract. To achieve this, we avoid the use of encryption and threshold decryption entirely, instead leveraging NIZKs and burn-based commitments to achieve anonymity through unlinkability.

Kite [18] 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 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 [15] uses distributed homomorphic encryption (threshold Paillier) [17] for vote aggregation and threshold decryption, blind signatures [16] for anonymous authentication, and ZKP for vote correctness. It also employs a hybrid data structure (Bloom filter [42] + Merkle tree [43]) 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 [44], [45] for threshold decryption, which introduces coordination complexity and may pose deployment challenges in high-coercion or adversarial settings.

Cicada[13] uses HTLPs[14] to enable private, one-round vote submission. Voters encrypt their ballots so that they can only be decrypted after a preset time. While this design avoids interaction, it results in post-tally vote disclosure, as all ballots are eventually revealed. In contrast, our protocol achieves vote confidentiality through a burn-based mechanism. Although votes are revealed after the tally, they remain unlinkable to the corresponding burn addresses, ensuring that voter anonymity is preserved at all times.

Semaphore [21] is a general-purpose anonymity tool that allows users to prove Merkle membership and submit anonymous signals using zkSNARKs. Semaphore supports unlinkability through ZKPs allowing users to prove group membership and send signals without revealing their identity. Because an adversary/coercer cannot verify which identity sent a specific signal, they cannot confirm compliance with their demands, thereby [21], similar to our approach achieves a certain level of coercion resistance. However, [21] is not a complete voting protocol and lacks constructions for eligibility check, vote validity enforcement, and tallying mechanism.

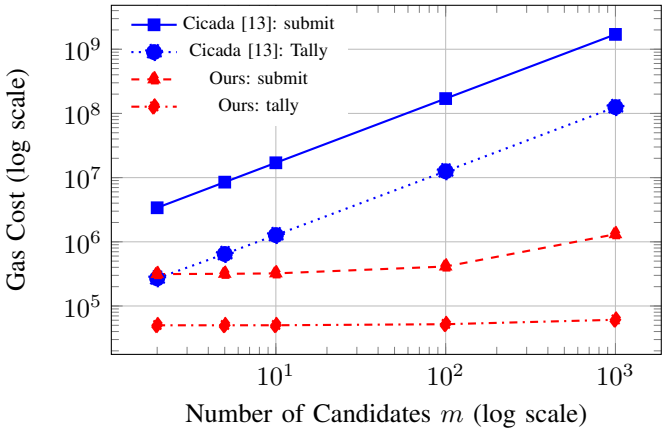

Fig. 2: Gas cost comparison of cicada vs. our design for *vote submission* and *tally* over varying number of candidates.

McCorry et al. [46] propose a self-tallying voting protocol in which voters publish encrypted binary votes on-chain, and the final tally is obtained by aggregating ciphertexts. Their main limitation is that it only supports binary voting due to its reliance on homomorphic encryption over votes. The scheme achieves vote privacy as an individual ballot remains hidden unless all other voters collude. It further enables verifiable, decentralized tallying without relying on a trusted authority. However, the design requires a global public key setup and per-voter key management.

Dimitriou [5] introduces a coercion-resistant protocol using a tamper-resistant token randomizer. [32] 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) [47] and external decryption, which limits decentralization. Our protocol does not rely on trusted hardware to achieve coercion resistance.

PPVSS [27] 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 $\mathcal{O}(nm)$ for n talliers and m voters, making it suitable for scalable applications like e-voting. 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 [12] Minimal Anti-Collusion Infrastructure (MACI) is an open-source protocol for private on-chain voting that

TABLE VII: Qualitative Comparison of Voting Schemes

Characteristic	TLP-based (Cicada [13])	FHE-based (Kite [18])	Burn Your Vote (Ours)
Voter Complexity	High (HTLP setup + ZK proof generation)	High (AHE+ZK proof generation)	Low (ZK proof generation only)
Vote On-chain Cost	Medium (HTLP + NIZKs)	Medium-High (ZK + AHE vote)	Low (ZK Verify + 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	Decryption by trusted authority	Fully decentralized, no coordinator
Maximum Voters	Medium (HTLP aggregation bound)	Medium (HE aggregation bound)	High (scales with EVM throughput)

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.

IX. 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 [25] DSL and the Groth16 [26] 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.

Acknowledgment

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 partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 innovation program (grant PROCONTRA-885666).

References

[1] V. Butarin, "Blockchain voting is overrated among uninformed people but underrated among informed people," 2021.

[2] A. Juels, D. Catalano, and M. Jakobsson, "Coercion-resistant electronic elections," in *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, ser. WPES '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 61–70. [Online]. Available: https://doi.org/10.1145/1102199.1102213

[3] G. S. Grewal, M. D. Ryan, S. Bursuc, and P. Y. Ryan, "Caveat coercitor: Coercion-evidence in electronic voting," in *2013 IEEE Symposium on Security and Privacy*, 2013, pp. 367–381.

[4] J. Benaloh and D. Tuinstra, "Receipt-free secret-ballot elections (extended abstract)," in *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, ser. STOC '94. New York, NY, USA: Association for Computing Machinery, 1994, p. 544–553. [Online]. Available: https://doi.org/10.1145/195058.195407

[5] T. Dimitriou, "Efficient, coercion-free and universally verifiable blockchain-based voting," *Computer Networks*, vol. 174, p. 107234, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128619317414

[6] "Rethinking voter coercion: The realities imposed by technology," 2013. [Online]. Available: https://www.usenix.org/system/files/conference/evtwote13/jets-0101-benaloh.pdf

[7] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *STOC*, 2009.

[8] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene, "A homomorphic lwe based e-voting scheme," in *PQC*, 2016.

[9] R. Del Pino, V. Lyubashevsky, G. Neven, and G. Seiler, "Practical quantum-safe voting from lattices," in *ACM CCS*, 2017.

[10] P. Bogetoft, I. Damgård, T. Jakobsen, K. Nielsen, J. Pagter, and T. Toft, "A practical implementation of secure auctions based on multiparty integer computation," in *Financial Crypto*, 2006.

[11] J. Alwen, R. Ostrovsky, H.-S. Zhou, and V. Zikas, "Incoercible multiparty computation and universally composable receipt-free voting," in *CRYPTO*, 2015.

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

[13] N. Glaeser, I. A. Seres, M. Zhu, and J. Bonneau, "Cicada: A framework for private non-interactive on-chain auctions and voting," *Cryptology ePrint Archive*, 2023.

[14] G. Malavolta and S. A. K. Thyagarajan, "Homomorphic time-lock puzzles and applications," Cryptology ePrint Archive, Paper 2019/635, 2019. [Online]. Available: https://eprint.iacr.org/2019/635

[15] S. Zhang, L. Wang, and H. Xiong, "Chaintegrity: blockchain-enabled large-scale e-voting system with robustness and universal verifiability," *International Journal of Information Security*, vol. 19, no. 3, pp. 323–341, 2020. [Online]. Available: https://doi.org/10.1007/s10207-019-00465-8

[16] D. Chaum, "Blind signatures for untraceable payments," in *Advances in Cryptology*, D. Chaum, R. L. Rivest, and A. T. Sherman, Eds. Boston, MA: Springer US, 1983, pp. 199–203.

[17] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology — EUROCRYPT '99*, J. Stern, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238.

[18] K. Nazirkhanova, V. Gunjur, X. Jesus, and D. Boneh, "Kite: How to delegate voting power privately," *arXiv preprint arXiv:2501.05626*, 2025.

[19] K. Karantias, A. Kiayias, and D. 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*. Springer, 2020, pp. 523–540.

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

[21] A. Pertsev, R. Semenov, and R. Storm, "Tornado cash privacy solution version 1.4," 2019.
[22] A. Gupta and K. Gurkan, "PLUME: An ECDSA nullifier scheme for unique pseudonymity within zero knowledge proofs," Cryptology ePrint Archive, Paper 2022/1255, 2022. [Online]. Available: https://eprint.iacr.org/2022/1255
[23] S. P. Lalley and E. G. Weyl, "Quadratic voting: How mechanism design can radicalize democracy," *AEA Papers and Proceedings*, vol. 108, p. 33–37, May 2018. [Online]. Available: https://www.aeaweb.org/articles?id=10.1257/pandp.20181002
[24] S. Park and R. L. Rivest, "Towards secure quadratic voting," *Public Choice*, vol. 172, no. 1/2, pp. pp. 151–175, 2017. [Online]. Available: https://www.jstor.org/stable/48719874
[25] M. Bellés-Muñoz, M. Isabel, J. L. Muñoz-Tapia, A. Rubio, and J. Baylina, "Circom: A circuit description language for building zero-knowledge applications," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–18, 2022.
[26] J. Groth, "On the size of pairing-based non-interactive arguments," in *Advances in Cryptology – EUROCRYPT 2016*, M. Fischlin and J.-S. Coron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 305–326.
[27] K. Baghery, N. Knapen, G. Nicolas, and M. Rahimi, "Pre-constructed publicly verifiable secret sharing and applications," Cryptology ePrint Archive, Paper 2025/576, 2025. [Online]. Available: https://eprint.iacr.org/2025/576
[28] "Eip-7503: Zero-knowledge wormholes," 2023. [Online]. Available: https://eips.ethereum.org/EIPS/eip-7503
[29] I. Damgård, "Commitment schemes and zero-knowledge protocols," in *School organized by the European Educational Forum*. Springer, 1998, pp. 63–86.
[30] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "Poseidon: A new hash function for zero-knowledge proof systems." in *USENIX Security Symposium*, vol. 2021, 2021.
[31] W. Lueks, I. Querejeta-Azurmendi, and C. Troncoso, "{VoteAgain}: A scalable coercion-resistant voting system," in *29th USENIX security symposium (USENIX Security 20)*, 2020, pp. 1553–1570.
[32] B. Lee and K. Kim, "Receipt-free electronic voting scheme with a tamper-resistant randomizer," in *Information Security and Cryptology — ICISC 2002*, P. J. Lee and C. H. Lim, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 389–406.
[33] M. Hirt and K. Sako, "Efficient receipt-free voting based on homomorphic encryption," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2000, pp. 539–556.
[34] A. Juels, D. Catalano, and M. Jakobsson, "Coercion-resistant electronic elections," in *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, 2005, pp. 61–70.
[35] A. Kiayias, T. Zacharias, and B. Zhang, "End-to-end verifiable elections in the standard model," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 468–498.
[36] "One noun, every day, forever." https://nouns.wtf/, 2025.
[37] "Private identity verification," https://zkpassport.id/, 2025.
[38] K. Karantias, A. Kiayias, and D. Zindros, "Proof-of-burn," *IACR Cryptol. ePrint Arch.*, 2019.
[39] J. Fraenkel and B. G. and, "The borda count and its real-world alternatives: Comparing scoring rules in nauru and slovenia," *Australian Journal of Political Science*, vol. 49, no. 2, pp. 186–205, 2014. [Online]. Available: https://doi.org/10.1080/10361146.2014.900530
[40] D. R. Woodall, "The original borda count and partial voting," *Social Choice and Welfare*, vol. 40, no. 2, p. 353–358, 2013. [Online]. Available: https://www.researchgate.net/publication/257405174_The_Original_Borda_Count_and_Partial_Voting
[41] L. Harrison, S. Bag, and F. Hao, "Camel: E2E verifiable instant runoff voting without tallying authorities," Cryptology ePrint Archive, Paper 2023/1920, 2023. [Online]. Available: https://eprint.iacr.org/2023/1920
[42] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, 2000.
[43] R. C. Merkle, "Protocols for public key cryptosystems," in *1980 IEEE Symposium on Security and Privacy*, 1980, pp. 122–122.
[44] D. Boneh and M. Franklin, "Efficient generation of shared rsa keys," *J. ACM*, vol. 48, no. 4, p. 702–722, Jul. 2001. [Online]. Available: https://doi.org/10.1145/502090.502094
[45] M. Chen, C. Hazay, Y. Ishai, Y. Kashnikov, D. Micciancio, T. Riviere, abhi shelat, M. Venkitasubramaniam, and R. Wang, "Diogenes: Lightweight scalable RSA modulus generation with a dishonest majority," Cryptology ePrint Archive, Paper 2020/374, 2020. [Online]. Available: https://eprint.iacr.org/2020/374
[46] P. McCorry, S. F. Shahandashti, and F. Hao, "A smart contract for board-room 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*. Springer, 2017, pp. 357–375.
[47] "Trusted platform module (tpm) summary," accessed: 2025-05-11. [Online]. Available: https://trustedcomputinggroup.org/resource/trusted-platform-module-tpm-summary/
[48] D. Catalano and D. Fiore, "Vector commitments and their applications," in *Public-Key Cryptography–PKC 2013: 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26–March 1, 2013. Proceedings 16*. Springer, 2013, pp. 55–72.
[49] E. N. Tas and D. Boneh, "Vector commitments with efficient updates," *arXiv preprint arXiv:2307.04085*, 2023.
[50] D. Boneh, B. Bünz, and B. Fisch, "Batching techniques for accumulators with applications to iops and stateless blockchains," in *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part I 39*. Springer, 2019, pp. 561–586.

APPENDIX A
VECTOR COMMITMENT

A vector commitment (VC) is a cryptographic primitive that functions as an accumulator, enabling a committer to bind to an ordered sequence of values. This allows for selective openings of the commitment at specific positions, providing compact proofs. The VC scheme offers public verifiability, meaning a verifier can confirm the authenticity of individual elements within the committed vector without the need for having access to the entire sequence [48], [49], [50].

Definition A.1. A VC scheme can be defined as a quadruple of PPT algorithms $VC = (\mathcal{KG}_\Delta, \mathcal{C}_\Delta, \mathcal{O}_\Delta, \mathcal{V}_\Delta)$ as follows:

- $pp \leftarrow \mathcal{KG}_\Delta(1^\lambda, n)$: The generator outputs public parameters pp based on the security parameter λ, and message space size of n.
- $\Delta_M \leftarrow \mathcal{C}_\Delta(pp, m_1, \ldots, m_n)$: Commits to elements $\{m_1, \ldots, m_n\}$.
- $\delta_k \leftarrow \mathcal{O}_\Delta(pp, m_k, k, \text{aux})$: Produces a proof for opening the commitment corresponding to the kth element in the sequence, where $k \in \mathbb{N}_n$. The opening usually requires knowledge of some auxiliary value aux.
- $b \leftarrow \mathcal{V}_\Delta(pp, \Delta_M, m_k, k, \delta_k)$: The verifier accepts the proof δ_k iff m_k is indeed the kth value of the accumulated Δ_M.

VC schemes can also support advanced features such as updatability and batch proofs [50], [49]. However, for simplicity, we focus on the basic definition while noting that these features remain compatible with our constructions. A VC scheme satisfies ***position-binding*** if, for every index i in the range $[n]$ and for any PPT adversary $\mathcal{A}$, the probability that $\mathcal{A}$ successfully opens the commitment at index i with two different values is negligible in the security parameter λ [48].

APPENDIX B
CIRCOM VOTE CIRCUIT

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

APPENDIX C
SYSTEM-SETUP

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

A. 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)

B. 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. 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

APPENDIX D
VOTING SMART CONTRACT

Algorithm 1: Initialize Voting Contract

Input: $T_{\texttt{submission}}, T_{\texttt{tally}}$, system parameters: `merkle_root`, `ceremony_id`, `state_root`

Output: Voting contract initialized

1 Set verifier contract;
2 Set submission deadline $T_{\texttt{submission}}$ and tally deadline $T_{\texttt{tally}}$;
3 Set `merkle_root`, `ceremony_id`, `state_root`;
4 Initialize vote counters: `yesVotes` $\leftarrow 0$, `noVotes` $\leftarrow 0$;
5 Initialize `tallyCompleted` $\leftarrow$ `false`;
6 Initialize empty mapping `usedNullifiers`;

Algorithm 2: Vote Submission

Input: Vote value v, burn proof π_{vote}, public inputs $(\eta, \mathsf{V}_{\mathsf{id}}, \mathtt{sr}, \ldots)$
Output: Vote recorded or rejected

if *Voting period has ended* **then**
  reject vote submission;
if *Nullifier η already recorded as spent* **then**
  reject vote submission;
if *Nullifier does not belong to correct VC root Δ* **then**
  reject vote submission;
if *π_{vote} is not valid for $\mathcal{R}_{\mathsf{vote}}$* **then**
  reject vote submission;
if *v is not a valid option under the voting rules* **then**
  reject vote submission;
Record η as spent;
Store vote value v for η;
return success;

Algorithm 3: Revote Handling (Optional)

Input: New vote value v', same nullifier η, new burn proof $\pi_{\mathsf{vote}'}$
Output: Vote updated or rejected

if *Revoting is disabled* **then**
  reject revote;
if *Voting period has ended* **then**
  reject revote;
if *$\pi_{\mathsf{vote}'}$ is not valid for $\mathcal{R}_{\mathsf{vote}}$* **then**
  reject revote;
Overwrite previously stored vote for η with v';
return success;

Algorithm 4: Finalize Tally

Input: Current time t
Output: Vote result computed or rejected

if $t < T_{\mathtt{tally}}$ **then**
  reject tallying (TallyingNotAllowed);
if $\mathtt{tallyCompleted} = \mathtt{true}$ **then**
  reject tallying (AlreadyTallied);
set $\mathtt{tallyCompleted} \leftarrow \mathtt{true}$;
Compute result:
  $\mathtt{passed} \leftarrow (\mathtt{yesVotes} > \mathtt{noVotes})$;
emit `VotingResults` event;

APPENDIX E
DETAILS OF THE SUPPORTED VOTING SCENARIOS

1) Majority Voting (First-Past-the-Post): This is a binary majority voting system in which the outcome is determined by whether the number of affirmative votes exceeds 50%. It is implemented as an equal-weight scheme, where the amount of tokens burned does not affect the voting weight. Let N denote the total number of voters. The tally is as follows:

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

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

2) Token-Weighted Voting: Each voter casts a vote for exactly one candidate by burning a chosen amount of tokens. The vote weight assigned to a candidate is proportional to the number of tokens burned. In their $\mathcal{R}_{\mathsf{vote}}$, voters prove that the balance of the burn address Λ is $a > \tau_i$, where τ_i is part of the public statement ϕ. The tally is computed as follows:

$$\forall\, k \in [1, m], \quad \text{score}_k = \sum_{i=1}^{N} \tau_i \times \delta_{k, v_i}$$

where $\delta_{k,j}$ is the *Kronecker delta* function, which returns 1 when $k = j$ and 0 otherwise.

3) Quadratic Voting [24]: This scheme allows voters to express the intensity of their preferences by allocating votes such that the cost increases quadratically with the number of votes cast. Let each voter's ballot be represented as a vector $B_i = (b_{i,1}, \ldots, b_{i,m})$, where the following holds:

$$\forall i \in \mathbb{Z}_n : \sum_{j=1}^{m} b_{i,j}^2 \leq \tau_i, \quad \forall j \in \mathbb{Z}_m : \text{score}_j = \sum_{i=1}^{N} b_{i,j},$$

where τ_i denotes the budget of voter i, determined by the amount of tokens they have burned.

4) Ranked-Choice Voting (RCV) voters rank candidates in order of preference. Each voter submits a ranking vector R_i, with $R_{i,0}$ in the top choice, $R_{i,1}$ the second choice, and so on. Two common tally methods are:

- **Borda Count [39], [40]:** Each candidate receives points inversely proportional to their ranking position:

$$\forall j \in \mathbb{Z}_m \mid \text{score}_j = \sum_{i=1}^{N} (m - \mathsf{R}_i.\mathsf{IndexOf}(j)).$$

- **Instant-Runoff Voting (IRV) [41]:** 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.

5) Open-Content Voting: This scheme allows voters to submit votes containing **arbitrary content**, such as strings, numbers, or textual proposals—essentially, any data the voter wishes to express. It is suitable for open-ended decisions, such as selecting a protocol name, slogan, or other customizable inputs where predefined options may be insufficient. If the vote value exceeds the size of a field element (e.g., larger than 254 bits),

the voter may instead use a hash $H(v)$ in the construction of Λ. During the vote submission phase, the contract can compute $H(v)$ on-chain and verify it against the submitted proof π_{vote} in a trustless manner. This scheme can be viewed as a generalization of majority voting, extended to support arbitrary vote values. To the best of our knowledge, this is the first design to enable such a voting mechanism in a blockchain setting.