State of Private Voting 2026

An in-depth analysis of private voting protocols in the Ethereum ecosystem

privacy stewards of ethereum

shutter

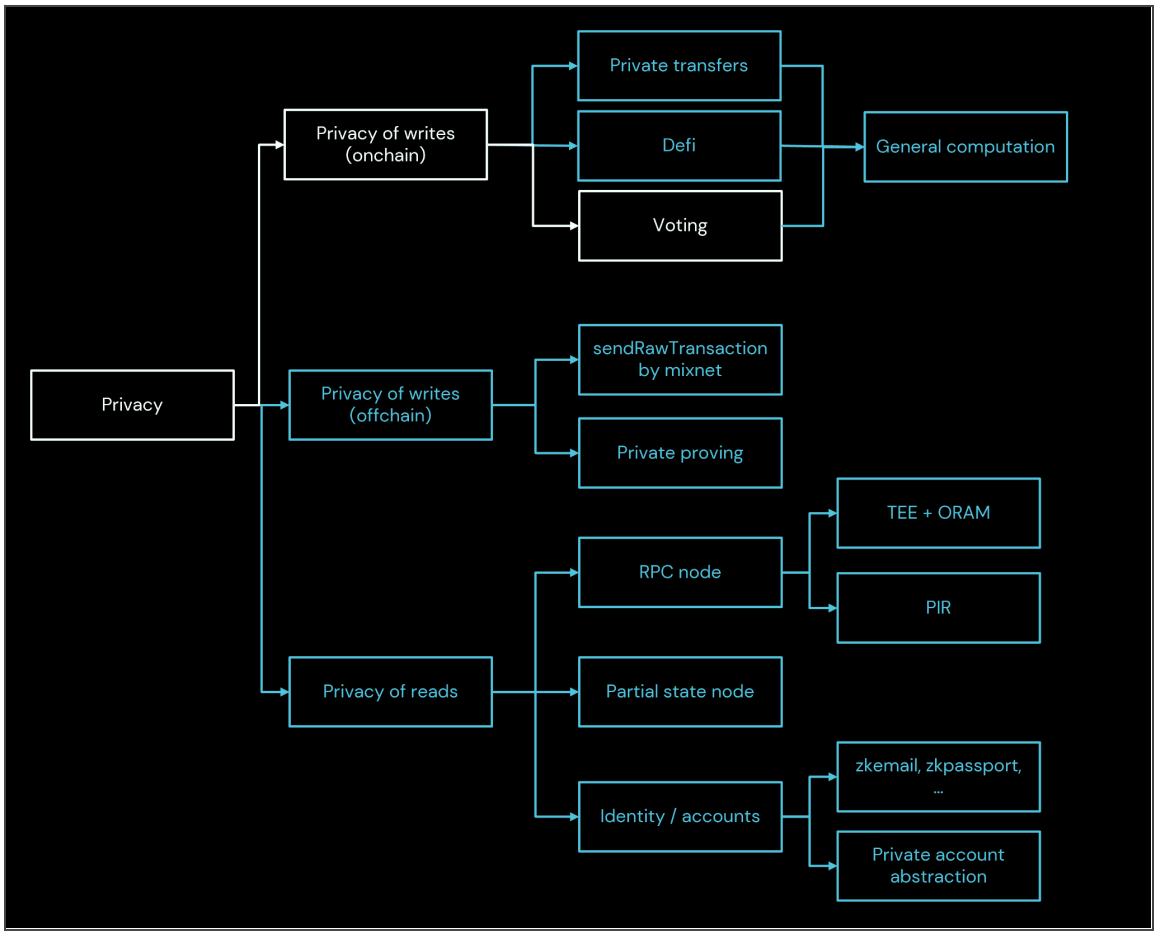
Table of Contents

Introduction

This report examines private voting projects and protocols built on Ethereum and presents the State of Private Voting 2026. We present a set of criteria to examine voting protocols and tools, compare different solutions according to the criteria, make recommendations for a selection of desirable properties, and outline future work in the field.

In 2025, privacy awareness has risen significantly in the blockchain ecosystem. For a long time Ethereum has been home to many important privacy projects and initiatives. That being said, the status quo for onchain activities remains fully transparent and public. One helpful way of understanding privacy in a blockchain context is to break it down into the following three categories:

1. Privacy of writes (onchain)
2. Privacy of writes (offchain)
3. Privacy of reads



Voting is best described as inside the category “Privacy of writes (onchain)”. Onchain voting has unlocked novel organizational structures, applications, and capital allocation mechanisms, such as DAOs (Decentralized Autonomous Organizations), funding mechanisms such as quadratic funding (QF) and retroactive public goods funding (RPGF). Voting can also be encapsulated within the privacy of writes (offchain) category.

The Need for Private Voting

Private voting protects individuals and collective decision making processes. There is significant consensus that democratic processes must include a secret ballot in order to function most effectively. Private voting in democratic elections means being able to vote without anyone finding out who you voted for, and making it difficult for voters to prove beyond reasonable doubt how they voted.

Fully public and transparent voting creates information asymmetries and side-games that can be exploited to influence voting choices. Information asymmetry exists when participants can see private information before others, leading to unfair advantages. For example, seeing the result is favoring one direction mid-vote. Side-games include voter bribery, coercion and social pressure. Privacy restores information symmetry so that each voter can truthfully decide without pressure how to vote. Side-game incentives are also strongly discouraged with privacy guarantees.

1. **Bandwagon Effects -** When votes and tallying are public, interim votes can encourage bandwagon effects, whereby voters defer to vote for the perceived winner, rather than vote for their preferred candidate. This can also contribute to voter apathy, where voters disengage from the voting process as they feel like they don’t have a say in the result. This herd mentality is restricted when voters cannot see who has voted for what and what the real-time vote result is.
2. **11th Hour Voting and Whale Intervention -** Public real-time voting leads to information asymmetries that permit two strategies that can be employed by whales or a coordinated group. At the beginning of a vote, a whale or coordinated group can submit a large number of votes in favour of their preferred option. This can stifle further votes and incentivise the bandwagon effects described above, as the vote result appears more inevitable. Secondly, 11th hour voting allows entities to take advantage of the public tally total and their ability to deploy large amounts of voting power. This strategy involves waiting until the end of the vote as a whale or coordinated group before deciding to flip the vote in a certain direction.
3. **Social Pressure -** Public votes make it easy to identify individuals, which can lead to voters fearing community retaliation based on how they vote. Leaders and the community can reward or punish these individuals based on their past choices, exclude them from opportunities, or shame them. This nudges voters and delegates to vote in their best interests, rather than vote honestly. Private voting reduces these social incentives.

4. **Coercion and Voting Markets -** Vote verification enables problematic markets. In a public governance setting, anyone can verify how a specific vote was cast and the chosen option. This creates conditions where governance and voting power can be openly traded, as buyers (e.g., bribers) can verify their bought vote was submitted. Although bribery is prohibited by law in most elections, markets currently exist on Ethereum where anyone can buy and sell their voting power because of the relative simplicity and scalability of providing verifiable receipts.

The Challenges of Private Voting

Despite the proposed benefits of private voting, transparency has long been a core feature in DAOs, and full transparency can be seen as a feature worth keeping. DAOs were born by mixing democracy and onchain properties such as decentralization, immutability, and transparency. They theoretically provide a great solution for democratic governance and have seen practical usage for many years. There are a number of challenges to consider when introducing private voting.

1. **Transparency & Efficiency -** The full transparency of DAOs is an important feature to some participants. Some users like being able to see who votes for what, how many votes are assigned to different options in real time. By adding privacy, DAO votes lose these features, and some feel that privacy does not provide the value that transparency provides. A notable example is that the votes of other voters and delegates are used as a proxy for thoughtful and well-reasoned voting decisions. The friction of understanding every proposal in full, and coming to a well-reasoned independent conclusion is significant. As a voter, I can take a cognitive shortcut and examine how different delegates have voted as I trust that those individuals have done the work to inform themselves correctly and they align with my views. The public reputations of certain voters and transparent voting increase the decision-making efficiency of other voters. This is especially true in cases when voting on a topic that you are not familiar with. For example, it may be better to defer to the opinion of a core protocol dev when voting on technical proposals.

2. **User Experience -** There are other trade-offs to consider when enabling private voting which do not exist with transparent onchain voting today. An ideal private voting protocol should be as easy and as cheap to use as onchain voting is today. Private voting introduces additional complexity, trust assumptions and gas costs. Some view the complexity and additional gas costs as too prohibitive given the advantages of private voting.

3. **DAO Management & Whipping Votes -** DAO leadership and influential delegates can be apprehensive to enable private voting because it can harm their ability to direct the

DAO in a specific direction. In the worst case, this can be framed as malicious manipulation in order to actively benefit these specific participants. In other cases, it is worth noting that DAOs have undergone a certain amount of experimentation to get where we are today—there was a larger focus on embarking on a credible path to protocol decentralization in the past. This has resulted in some DAOs decentralizing too much, or too early. It could be argued that influential voters can play an important role here in directing proposals efficiently for the long term benefit of the DAO.

4. **Inability to Cover Every Edge Case –** Although the transparency of DAO votes is presented as an argument for why private voting is required, there also exists an interesting edge case where privacy may lead to a specific attack. An example of this scenario is the Compound governance attack in 2024 where the community was actively called to vote "no" during the voting period in order to suppress a malicious proposal that would have affected the DAO. We would contend that this edge case is not sufficient to meaningfully argue against private voting. In addition to privacy protecting governance processes from a variety of other threats, arguments can be made against the underlying token voting model which favours entities with significant capital to deploy. A running private tally helps reduce information asymmetry. In the case of a proposal made by a single malicious actor, there will always be information asymmetry regardless of whether there is a private running tally. (The malicious actor always knows how much voting power they have across one or more wallets. The other DAO members are never privy to this same information.) As a result, having a private running tally does not actually aid malicious proposals—it merely fails to prevent information asymmetry in this extreme case.

5. **Principal-Agent Problems –** Although private delegation is not used much today, there exists a principal-agent problem when supporting fully private delegation. I.e. privacy of voters delegation to delegates, and privacy for the delegates voting themselves. Users want individual privacy when submitting votes and when delegating votes. This guarantees nobody can judge the delegator by how they are acting or asking a third-party to act on their behalf. The problem arises when delegates can submit private votes and their delegators cannot verify that their voting power went to support or reject a particular vote option. Delegates can deceive delegators by publicly endorsing one choice and privately supporting another one. A reasonable approach that favors the transparency of representative governance while enabling privacy could enable privacy for voters delegating their voting power, but delegates are simply not given privacy, and are forced to vote transparently—the same way things work in representative democracies today.

Protocols & Projects

We analyzed the following private voting protocols in the Ethereum ecosystem in 2026. We considered established protocols, published research with upcoming prototypes and promising protocols in development. The following list is ordered by rank of the “implementation maturity” property (see definition below) and then in alphabetical order.

High Maturity

1. Freedom Tool
2. MACI V3
3. Semaphore V4
4. Shutter - Shielded Voting
5. SIV

Medium Maturity

6. Incendia
7. DAVINCI

Low Maturity

8. Aragon/Aztec
9. Cicada
10. Enclave
11. Kite
12. Shutter - Permanent Shielded Voting

Properties Definition

Integrity & Correctness

1. **Correct Execution** - No one can produce a false tally of votes.
2. **Robustness** - The system can tolerate a certain degree of malfunction and still deliver correct results.
3. **Censorship Resistance & Liveness** - No one can censor a vote or stop a vote from being tallied.
4. **Coercion Resistance** - Receipt freeness: voters cannot provide a verifiable receipt that they have voted a certain way to satisfy a coercer.

Privacy

5. **Private Vote** - Individual votes are not visible, or are not linked to any identity—i.e. it is infeasible to link a voter with a choice.
6. **Running Tally Privacy** - The running tally is not revealed until the end of a vote. No one can obtain in-progress results, which would give later voters more information than earlier voters.

Verifiability

7. **Quorum Status** - Supports display of quorum status (total voting weights cast) during the vote period.
8. **Individual Verifiability** - A voter can verify that their vote is included in the set of all cast votes.
9. **Universal Verifiability** - Anyone can check that the vote outcome corresponds to the sum of all cast votes.

Voter Authentication & Integrity

10. **Eligibility -** Only legitimate voters can vote.
11. **Uniqueness** - No eligible voter can contribute to the final tally with more than one vote.

Voting Methods

12. **Multiple Choice, Fractional Voting** - Supports voting methods in which there is more than one choice and/or in which a voter can assign fractions of their voting power to different choices.
13. **Weighted Voting** - Supports voting methods in which votes can vary in strength depending on specified criteria—e.g. for quadratic voting.

Flexibility

14. **Vote Updatability** - A voter can update their vote choice during the voting period.

Delegation & Representation

15. **Vote Delegation** - An eligible voter may delegate their voting power to a delegate, privately or non-privately.
16. **Delegation Privacy** - The system does not leak voter delegation choices—i.e. which delegate a voter delegated to.
17. **Engagement Disclosure** - Delegates must disclose whether or not they voted—but not how they voted to the public.
18. **Vote Disclosure** - Delegates must disclose how they voted—but only to voters who delegate voting power to them.
19. **Delegation Updatability** - Voters can re-delegate their voting power during the voting period.

Practical Considerations

20. **Implementation Simplicity** - How simple is the implementation of the project from a technical perspective.
21. **Ease of Use** - How straightforward is it for end users to interact with the system.
22. **Gas Efficiency** - How cheap is it to create proposals and cast votes.
23. **Ease of Integration/Deployment** - How easily the protocol can be deployed, adopted or embedded into existing governance systems, dApps, or infrastructure.
24. **Implementation Maturity** - How developed and battle-tested the protocol is, measured by its version history, deployments, audits, production readiness, and amount of use.

Evaluations

Freedom Tool

Overview

Freedom Tool is a private voting protocol built on top of the Rarimo protocol, a permissionless and generalized zero-knowledge proof registry and verifier. Designed primarily for real-world elections, it has already been deployed successfully in Georgia, Russia, and Iran. The system leverages the tools within the Rarimo ecosystem to deliver a simple, private, and robust voting mechanism.

- Website: https://freedomtool.org/
- GitHub: https://github.com/rarimo/passport-voting-contracts
- Whitepaper: https://freedomtool.org/#/whitepaper
- Testnet: Yes
- Mainnet: Yes

Explanation

The process begins with voter profile creation. A biometric passport is used to ensure the three fundamental properties of a secure voting system: authenticity, uniqueness, and eligibility. The voter scans their passport, verifies the data locally, and stores the necessary information on their device.

During voter registration, the user generates a keypair for identity management and creates a zero-knowledge proof demonstrating eligibility, covering factors such as citizenship, age, passport validity, and the authority's signature. This proof is then linked to the user's digital identity, while the keys are used to confirm the user's actions on smart contracts.

Next comes the voting round creation. Anyone can initiate a voting round by setting custom parameters that deploy three smart contracts. The first, the investment contract (INV), gathers funds to compensate relayer nodes. The second, the registration contract (REG), verifies that users control the appropriate keys. The third, the voting contract (VOT), receives anonymous votes and manages the counting of results.

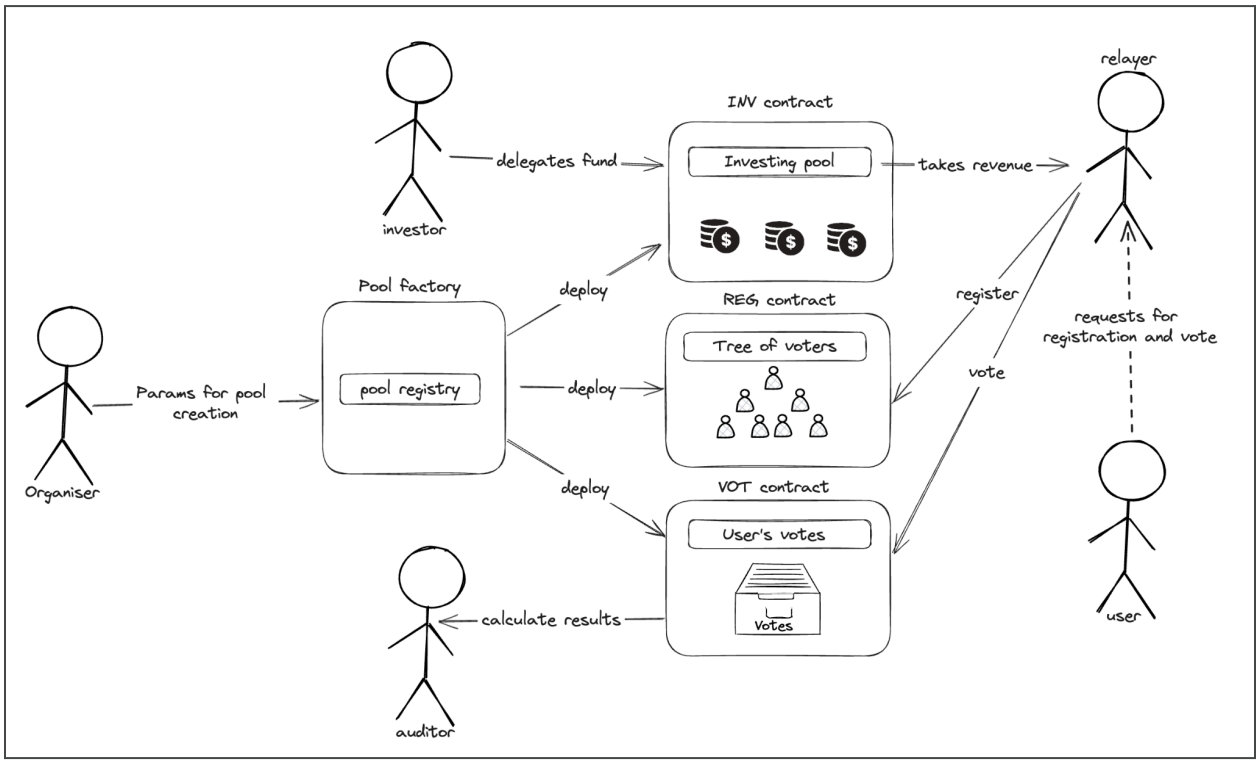
When a voter registers for a specific round, the procedure ensures that their vote remains anonymous and cannot be linked to their registration transaction. The voter generates a secret and computes a corresponding public key, along with a nullifier, which helps preserve anonymity.

The voter then provides proof of eligibility and adds their public key to a Merkle tree list, with the contract storing hashes to prevent double registration.

During the voting phase, the voter sends their chosen option along with a Merkle inclusion proof and the nullifier to a relayer. The relayer then packages this data into a transaction and submits it to the network. The VOT contract verifies the proof before recording the vote.

Tallying depends on how voting options are transmitted to the VOT contract. According to the whitepaper, options can either be sent in plaintext or encrypted using a group public key with threshold encryption. In the latter case, votes are decrypted individually after the round ends to produce the final tally.

Two upcoming features are Private Delegation and Proof of Liveness. The former will allow users to delegate their voting power to another user without revealing their identity. The latter will enable users to prove their physical presence and match with the passport photo.



Analysis

1. **Correct Execution** - Yes. No one can produce false results because votes are registered in a smart contract, which can only be submitted by authorized voters. The registration step prevents vote submission from users that do not pass the round criteria.

Freedom Tool uses the Rarimo ZK rollup so all messages submitted onchain are immutable as long as the underlying security assumptions hold (zero-knowledge soundness or encryption security). Anyone with access to the chain can check the smart contract state and recompute the results from the submitted votes. No entity can modify the submitted votes, not even the round creator or the system administrator.

2. **Robustness** - High. The Freedom Tool system can be considered robust because it mainly depends on onchain liveness of the Rarimo ZK rollup. There is a network of relayers that allow gasless transactions but in case all these relayers go offline, voters can still submit their vote directly to the smart contract. One scenario where the robustness of the system can be broken is if identity providers (usually authorities) are issuing extra passports and sending them to malicious actors in order to submit votes. In this case the vote results will differ from the actual result because there are more virtual voters than real ones.
3. **Censorship Resistance & Liveness** - Yes. A Freedom Tool voting round is censorship resistant and will be active as long as the contracts system is deployed in the Rarimo ZK rollup on top of a permissionless blockchain like Ethereum. Freedom Tool offers a network of relayers to send gasless transactions. If all relayers go offline, the voter can submit the vote directly onchain by paying the gas fees. The transaction can be submitted from another wallet using the registration zero-knowledge proof in order to avoid privacy compromises.
4. **Coercion Resistance** - No. Users generate two private values: a Nullifier and a Secret, which are used to register their public identity. When they submit a vote, they locally generate a zero-knowledge proof to guarantee that their identifier is located in the onchain Merkle tree. The vote is submitted in plaintext alongside this proof. During/after voting, voters could recreate this proof and show third-parties how they voted. The whitepaper describes a mode of hiding the votes by having a set of participants create a threshold encryption public key that would be used to encrypt the votes until the process finishes. After the voting process ends, the threshold participants will come together to decrypt the results. This addition could impact the robustness property because the results will depend on n threshold participants coming together to decrypt the votes.
5. **Private Vote** - Yes. The cast vote is public. There is no way to link the cast vote to a specific user because the vote is sent with a proof of inclusion without explicitly indicating which user it is. The hiding votes mode allows for private votes during the voting process, but it does not influence the individual private vote property because the submission process still requires the proof of inclusion zero-knowledge proof to send the vote onchain which provides privacy. Instead, this mode mainly affects the running tally privacy property
6. **Running Tally Privacy** - No. In the normal mode scenario, the cast votes are public onchain. Anyone can tally the votes during the voting process and afterwards. In the hiding votes mode, the submitted votes are encrypted and the results are unveiled after the voting process ends. A set of threshold encryption participants is required to come together to decrypt the votes and publish the final tally.

7. **Quorum Status** - Yes. Freedom Tool supports display of quorum status (total votes) during vote period because the cast votes are public onchain. Anyone can see the votes and count them. There is no vote updatability feature therefore every submitted vote is unique and can be counted to get the quorum status. In the hiding votes mode the votes are encrypted but they are still unique per voter so they could be counted in order to get the quorum status.
8. **Individual Verifiability** - Yes. When a voter submits a vote, they create a local zero-knowledge proof to show their identifier is located at the registration Merkle tree. The voter submits their vote choice with the proof through a relayer or directly onchain. The voter can see their cast public vote and their inclusion proof onchain as long as they have access to read onchain data.
9. **Universal Verifiability** - Yes. In the normal mode, all users can check individual votes and compute the final tally corresponding to the sum of all cast votes because the submitted votes are in plaintext. In the hiding votes mode, the final results are decrypted by a set of threshold encryption participants therefore there is no universal verifiability unless these threshold participants submit a zero-knowledge proof of correct execution with their partial decryption share onchain.
10. **Eligibility** - Yes. Only users that passed the passport eligibility criteria (e.g. has a specific citizenship) can cast a vote. This step is executed in the registration smart contract that gets configured by the round creator. The gatekeeping process is executed using the zkPassport solution developed by the Rarimo Protocol team. After a voter has successfully passed the gatekeeping mechanism, their identifier (hash of nullifier and secret) is registered on the smart contract and they can create a proof of inclusion zero-knowledge proof to submit a vote.
11. **Uniqueness** - Yes. Users can only register to a poll once per poll to prevent user's duplication. This is guaranteed by the use of the zkPassport solution and the registration smart contract. Freedom Tool assumes that one user will only have one passport. This property can be compromised if malicious identity providers (e.g. authorities) create extra fraudulent passports and share them with malicious parties that will submit votes for a specific choice.
12. **Multiple Choice, Fractional Voting** - No. Freedom Tool focuses on elections so the concept of fractional voting is not supported in the current product. To achieve fractional voting, Freedom Tool would need to add a functionality to break voting power into smaller chunks and verify that the total amount has not been spent before casting a vote.
13. **Weighted Voting** - No. Freedom Tool does not explicitly have weighted voting because it is focused on real world elections rather than onchain governance. But it could be adapted to support weighted voting due to the cast votes being public. The tally process can be modified to account for a different weighted voting mode and all users will be able to reproduce the final results by recomputing the tally using the modified process.

14. **Vote Updatability** - No. Users cannot vote more than once because there is no way to identify which one is the valid vote. It is worth noting that the Freedom Tool is used for elections where there is no option to change the vote.
15. **Vote Delegation** - No. Freedom Tool does not currently allow for delegation. However, the Freedom Tool team reports that a planned future feature will allow users to delegate their voting power to another user without revealing their identity.
16. **Delegation Privacy** - No. Freedom Tool does not allow delegation. However, this feature is on their roadmap.
17. **Engagement Disclosure** - No. Freedom Tool does not allow delegation. However, this feature is on their roadmap.
18. **Vote Disclosure** - No. Freedom Tool does not allow delegation. However, this feature is on their roadmap.
19. **Delegation Updatability** - No. Freedom Tool does not allow delegation. However, this feature is on their roadmap.
20. **Implementation Simplicity** - Medium/High. The components for Freedom Tool include NFC passport scanning logic, zero-knowledge proof generation for passport data, smart contracts for registration and voting, Merkle-tree registries, relayer infrastructure, and optional threshold encryption. Freedom Tool uses solutions and infrastructure developed by the Rarimo protocol team.
21. **Ease of Use** - High. The voter only needs to scan their passport chip and select an option. The organizer needs to define the relayers compensation and set up the relayers before actually deploying a poll. Freedom Tool provides user-friendly mobile apps.
22. **Gas Efficiency** - Medium-High. The poll creation is assumed by the election authorities. Votes can be published directly onchain (cheap enough depending on L1, L2) but there is also the option to use relayers to submit gasless transactions.
23. **Ease of Integration/Deployment** - High. The Freedom Tool website lets you create a poll interactively (it is deployed on ipfs, so just reload the page if any error occurs, but you should be able to create one poll). It is also a stand alone voting platform for non-Web3 users, so the team has not considered integration.
24. **Implementation Maturity** - High. The Freedom Tool has been used in 3 national elections (Georgia, Russia and Iran) which shows a clear trust record of different third-party entities. It should be noted that these elections are shadow elections or parallel governance projects advanced by opposition parties, to advance freedom and democracy where it is needed most. The team has a roadmap with new features and improvements which shows development support and maintenance of the project.

MACI V3

Overview

MACI (Minimal Anti-Collusion Infrastructure) is a private voting protocol developed by Privacy Stewards of Ethereum (PSE). Messages are stored in Merkle trees and processed by a trusted coordinator who decrypts votes off-chain, applies state transitions, and tallies the result. MACI provides collusion resistance, receipt-freeness and correct execution. However, individual ballots are visible to the coordinator. The protocol is relatively mature as it has gone through multiple iterations. The main shortcomings of MACI are the centralized coordinator and complexity of using the protocol.

- Website: https://maci.pse.dev/
- GitHub: https://github.com/privacy-scaling-explorations/maci
- Ethresearch: https://ethresear.ch/t/minimal-anti-collusion-infrastructure/5413
- Testnet: No
- Mainnet: No

Explanation

Users begin by generating a MACI key pair based on an EdDSA key and registering with the main MACI contract. This contract includes a gatekeeping mechanism,such as a proof of personhood check, to ensure eligibility before a user's public key is recorded onchain.

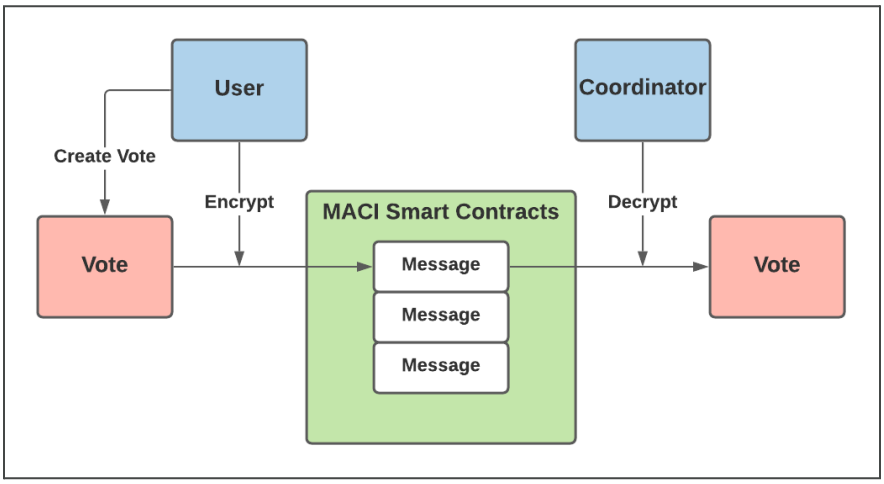
After registration, users must join the specific poll in which they intend to vote. Each poll can enforce its own gatekeeping requirements, allowing organizers to tailor access controls. To join a poll, the user must also provide a zero-knowledge proof confirming they possess the private key corresponding to their registered MACI public key.

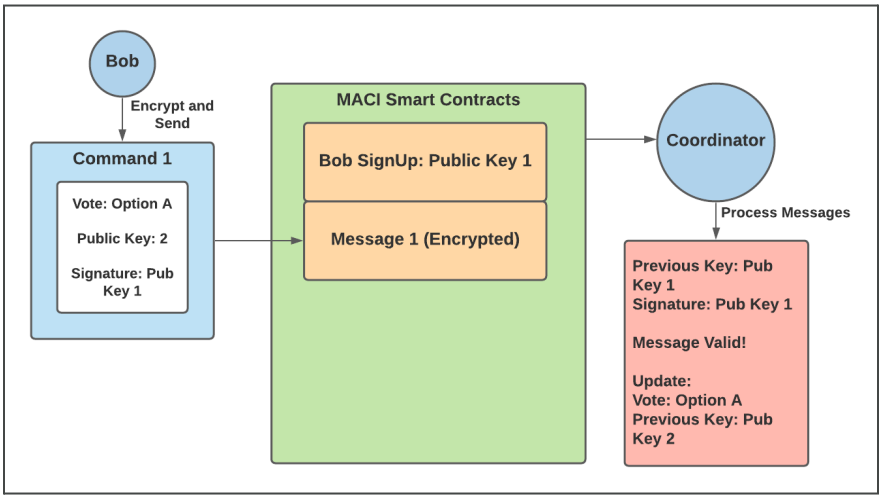
Once enrolled in a poll, the user submits a vote command. This command is a signed message containing the user's public key, their vote, and several additional parameters, all signed with their MACI key pair. To preserve privacy, the vote is encrypted using a shared key known only to the user and the coordinator.

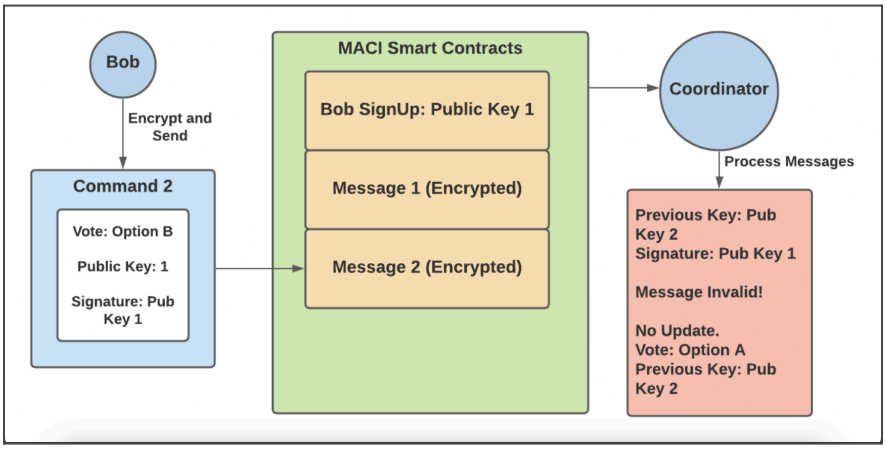
Users can also perform a key change by publishing a key change message onchain. This mechanism protects against bribery, as it prevents any third party from verifying whether their bribed vote was ultimately overwritten.

When the voting round concludes, the coordinator finalizes the process in three main steps: merging the state trees, decrypting and tallying the votes, and generating zero-knowledge proofs

that confirm the correct tallying and message processing. These proofs are then posted onchain for public verification. The final tally results are published in aggregate form, revealing the overall outcome without disclosing how any individual voted.







Analysis

1. **Correct Execution** - Yes. No one, not even the trusted coordinator, should be able to produce a false tally of votes. Individual messages (encrypted votes) are submitted onchain by authorized voters using events to save gas. A hash chain on the smart contract keeps the record of the emitted events so no messages can be censored or pass unnoticed. The coordinator tallies the votes by fetching all vote messages, decrypting them and summing them up. This process happens in a circuit so a zero-knowledge proof of correct execution is generated alongside the tallied results. The coordinator publishes the results on the smart contract using the proof to guarantee correct execution. Anybody can check the tallied results values and verify the posted proofs.
2. **Robustness** - Medium. MACI has a medium level of robustness, if you trust the coordinator, the system has strong censorship, privacy, and anti-collusion guarantees. However, the coordinator is a single trusted entity which can break certain assumptions like tallying completion. If the coordinator does not like the output of the results, they can refuse to publish the results and halt the tallying process.
3. **Censorship Resistance & Liveness** - Partial. No one—not even the trusted coordinator, should be able to censor a vote. The coordinator can threaten the liveness of the result. The coordinator can calculate the results locally because they have the decryption key of the votes value. If the coordinator doesn't like the result, they could choose to not publish the results and halt the election process. Another reason for this could be that the coordinator loses its private key and loses the ability to decrypt votes. If the key was not shared or backed up, the election process will be halted and no results will be able to be published.
4. **Coercion Resistance** - Yes. No one except a trusted coordinator should be certain of the validity of a vote, reducing the effectiveness of bribery. No one can prove (besides to the coordinator) which way they voted. This property is obtained by implementing a key change mechanism that allows voters to switch voting public keys and fool a potential briber. The briber does not have any guarantee that the submitted vote with a particular key is the final vote and the voter has not changed it.
5. **Private Vote** - Yes. MACI encrypts each vote offchain before submission, so the raw vote choice is not publicly visible onchain. It uses zero-knowledge proofs so the tally can be verified without revealing which user cast which vote. However, it relies on a semi-trusted coordinator for decryption and still links votes to signup keys, leaving minor metadata correlation risks.
6. **Running Tally Privacy** - Yes. The tally remains hidden until results are revealed. Once the voting period is over, the coordinator decrypts votes, tallies them, verifies the result onchain, and publishes the tally results. The coordinator performs the tallying process locally and generates a zero-knowledge proof of correct execution that needs to be submitted with the results to the smart contract function.

7. **`Quorum Status`** - No. Votes are encrypted during the voting period and only the coordinator can decrypt and tally after the voting period closes. Votes, updated votes, and key changes are sent as messages onchain, from an outsider point of view it is impossible to know how many valid votes have been truly submitted. Final results (with proofs) are then published onchain. This design prevents any in-period totals like a quorum status from being revealed.
8. **`Individual Verifiability`** - Yes. Voters can see their `publishMessage` transaction was included onchain. The encrypted message can be seen to exist in the message tree (but the contents are encrypted). The coordinator must process all messages in the tree and prove this via a zero-knowledge proof.
9. **`Universal Verifiability`** - Yes. MACI's tallying process produces a zero-knowledge proof that publicly verifies that the final result corresponds exactly to the encrypted votes, without revealing individual ballots. Anyone can independently verify this proof. The only limitation is that verifiability assumes the coordinator behaves honestly when processing messages. The coordinator could omit or alter messages before generating the proof, but this could be detected by monitoring onchain data.
10. **`Eligibility`** - Yes. Only the owner of a user's private key must cast a vote tied to its corresponding public key. MACI presumes an identity system where each legitimate member controls a unique Ethereum private key. MACI allows the implementation of different gatekeeping mechanisms like ERC20 token holders, ERC20 votes, ERC721 (NFTs) token holders, Gitcoin passport, etc. The robustness of the eligibility property depends on the soundness of the gatekeeping mechanism (e.g. one NFT equals one vote).
11. **Uniqueness** - Yes. Only the owner of a user's private key may cast a vote tied to its corresponding public key. A user can only vote once. Note they are permitted to change their vote or key, but this doesn't allow them to vote twice as stale votes/votes from old keys are not counted.
12. **Multiple Choice, Fractional Voting** - Yes. It is possible to assign voting power to multiple vote options, and give those options different voting weights.
13. **Weighted Voting** - Yes. MACI supports full credits voting (you vote fully for one option), quadratic voting, and non-quadratic voting. Quadratic voting means voice credits (voting power) becomes the square root of the sum, so a vote weight of 5 costs 25 credits. Votes can be split across multiple parties. Non-qv means that votes can still be spread across parties, but the voice credits (voting power) is linear to the end voice credits received. For example, 5 votes cost 5 voice credits. Full voice credits means that you can only vote for one option, and all of your voting power goes to a chosen entity.
14. **Vote Updatability** - Yes. Each voter can submit multiple encrypted votes during the voting period, and only the latest valid message is counted in the final tally. This design allows participants to freely change their minds without revealing earlier choices.
15. **`Vote Delegation`** - No. MACI does not have features for delegation.
16. **`Delegation Privacy`** - No. MACI does not have features for delegation.
17. **`Engagement Disclosure`** - No. MACI does not have features for delegation.

18. **`Vote Disclosure`** - No. MACI does not have features for delegation.
19. **`Delegation Updatability`** - No. MACI does not have features for delegation.
20. **Implementation Simplicity** - Low. The MACI protocol is relatively complex and encompasses many different parts. There are smart contracts, circuits, an sdk, optional offchain infra (relayer and coordinator service).
21. **`Ease of Use`** - Medium-High. For the end user, MACI can be relatively straightforward to use, most of the complexity comes with running the MACI infrastructure, not signing up and voting. The current MACI does assume a user has access to an Ethereum account that can be used to sign a message to generate their MACI keys. In the frontend, this is handled by an extension wallet prompt to sign a message.
22. **`Gas Efficiency`** - Medium. MACI is cost-prohibitive to run on L1, but it can be run relatively cheaply on L2s. When submitting a batch size of 89 messages (votes) onchain, it costs around 2,800,000 gas.
23. **`Ease of Integration/Deployment`** - Low. MACI is not straightforward to implement. You need to deploy contracts with correct initialization parameters that do not hide all of the complexity of zero-knowledge proofs. You need an entity to act as the coordinator that can secure its private key, tally the votes, and publish the results onchain.
24. **`Implementation Maturity`** - High. MACI was first proposed in 2019, and the protocol is now on version 3. So it is a relatively mature implementation. For a zero-knowledge proof project, it uses established tooling and cryptography.

Semaphore V4

Overview

Semaphore, another project developed by Privacy Stewards of Ethereum (PSE), is a protocol that can be used for anonymous signalling within a group. Participants register an identity commitment in a Merkle group and later publish signals (votes) with a zero-knowledge proof of membership. A nullifier prevents double-signalling. While not a dedicated private voting protocol like MACI, Semaphore can be used if a DAO needs a simpler protocol for anonymous voting with fewer features. It can also be combined with other protocols to provide additional desired properties. For example, after tallying, user ballots can also be decrypted using Cicada (another project assessed in this report). Semaphore can be used to provide anonymous voter eligibility, so a decrypted ballot would only reveal that an eligible voter voted that way, not the who actually voted. This voter anonymity gives the Cicada indefinite ballot privacy.

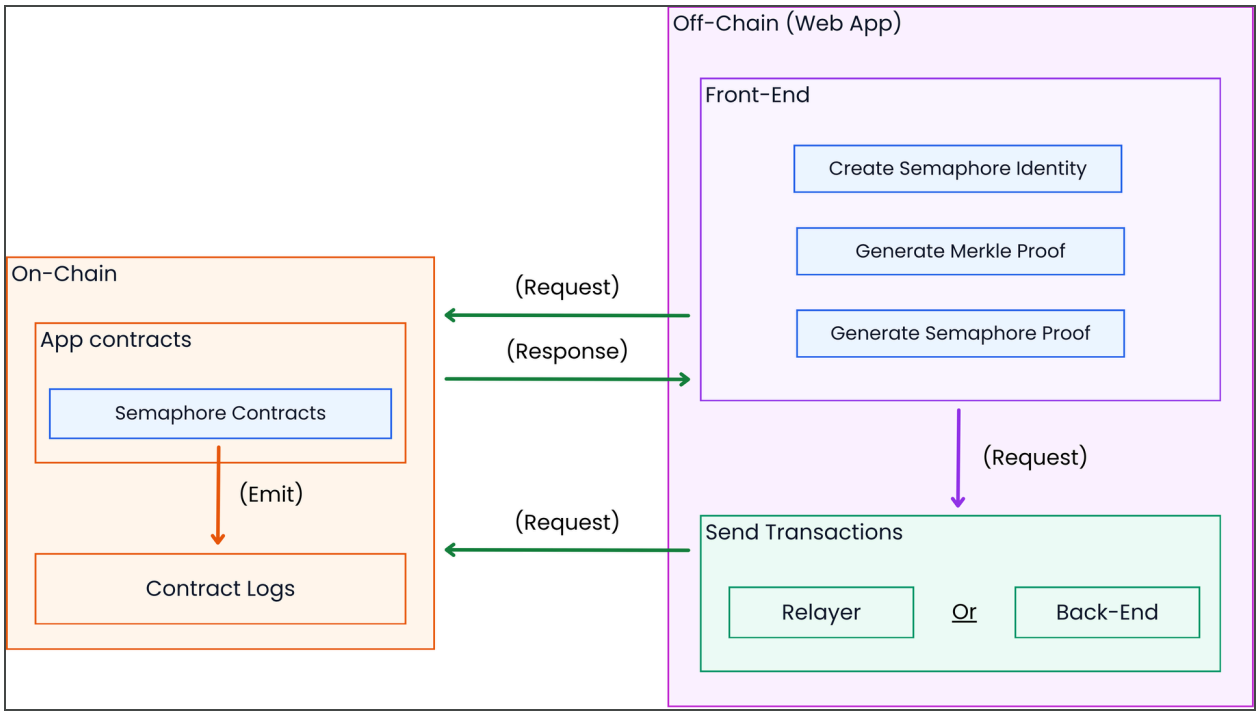
- Website: https://semaphore.pse.dev/
- GitHub: https://github.com/semaphore-protocol
- Testnet: Yes
- Mainnet: Yes

Explanation

A user begins by creating a Semaphore identity, which consists of an EdDSA private key, a nullifier, and another random value. Once the identity is created, it must be registered by inserting the corresponding identity commitment into Semaphore's identity tree.

When the user wishes to send a signal—such as casting a vote—they must broadcast it onchain along with a zero-knowledge proof and several additional values. The zero-knowledge proof confirms that the user is a valid member of the group and ensures that they have not previously broadcast a signal using the same nullifier, effectively preventing double-signalling.

Once the signal and proof are broadcast, the smart contracts verify the zero-knowledge proof onchain. Meanwhile, offchain libraries handle operations such as identity creation and group management, while onchain contracts oversee group membership and the validation of proofs.



Analysis

1. **`Correct Execution`** - Partial. Semaphore does not enforce "Correct Execution" of vote tallies. It focuses on ensuring that each vote comes from a valid group member and that no one votes twice. However, Semaphore doesn't define how votes are tallied or provide a cryptographic proof that the tally matches all submitted votes. That responsibility lies with the application layer built on top of Semaphore (e.g., a voting smart contract). If implemented correctly, each valid proof would contribute deterministically to a public onchain count and the correct execution would be achieved.
2. **`Robustness`** - High. Semaphore is relatively simple for a zero-knowledge proof protocol. Valid zero-knowledge proofs verified onchain ensure the protocol works as expected. Semaphore groups have group admins which are responsible for admitting and managing members. Proper use of the admin feature should be upheld. The admin can be an immutable smart contract that could only admit valid members for example.
3. **`Censorship Resistance & Liveness`** - Yes. Semaphore messages are published onchain directly. A badly implemented admin could threaten this property as admins can remove members.
4. **`Coercion Resistance`** - No. Semaphore does not have features for coercion resistance and receipt freeness. It is possible to prove that you have voted a certain way by revealing your circuit inputs which can be used to reconstruct the fact that you generated a specific nullifier with a specific message.

5. **Private Vote** - Partial. Semaphore allows a user to prove membership of a group without revealing their identity, and then to send a vote anonymously. It uses zero-knowledge proofs to ensure that the user belongs to the group and has not yet voted (via a nullifier), while the actual identity and link to the vote remain hidden. However, the group membership registration step and leaked metadata introduce some de-anonymisation risk.
6. **Running Tally Privacy** - No. All signals emit an event onchain with the message visible, making the running tally visible in real-time. It would be possible to search for onchain events, and verify that the votes in those messages sum up to the end tally.
7. **Quorum Status** - Yes. Since signals are submitted onchain during the vote period, and the votes would be unencrypted, it would be trivial to infer a running quorum status.
8. **Individual Verifiability** - Yes. It is possible for a user to verify their vote has been included by verifying proof verification was successful onchain. This is possible by checking onchain events.
9. **Universal Verifiability** - Yes. Because tallying and votes are public, anyone can verify the results have been tallied correctly
10. **Eligibility** - Yes. Each Semaphore identity commitment is associated with a private key. Only members added to the group can generate valid Merkle proofs of membership, enforced by the circuit's Merkle root verification.
11. **Uniqueness** - Yes. Semaphore employs a simple mechanism using nullifiers to prevent double signalling. Any signal re-using nullifiers with the same scope is rejected.
12. **Multiple Choice, Fractional Voting** - Possible. The protocol treats all messages equally, but application logic built on top could interpret Semaphore messages differently to account for different voting weights, including multiple choice voting with fractional weighting.
13. **Weighted Voting** - Possible. The protocol treats all messages equally, but application logic built on top could interpret Semaphore messages differently to account for different voting weights, including weighted voting such as QV.
14. **Vote Updatability** - Possible. Once a nullifier has been used once with the same scope, it cannot be re-used. Application logic could be built on top that interprets a new message from the same identity with a different scope as an updated vote, where the old message can be negated in tallying.
15. **Vote Delegation** - Possible. Semaphore does not support vote delegation. However, a custom vote delegation system can be built on top of Semaphore by adding an external smart contract or identity layer that tracks delegated relationships before generating Semaphore proofs. This would be an application-level extension, not a feature of Semaphore itself.
16. **Delegation Privacy** - NA. Depends on custom vote delegation logic.
17. **Engagement Disclosure** - NA. Depends on custom vote delegation logic.
18. **Vote Disclosure** - NA. Depends on custom vote delegation logic.
19. **Delegation Updatability** - NA. Depends on custom vote delegation logic.

20. **Implementation Simplicity** - High. Semaphore is a relatively simple zero-knowledge proof protocol. Even though Semaphore abstractions are clean, some understanding is still required of zero-knowledge proofs, Merkle trees, and cryptographic primitives.
21. **Ease of Use** - Medium-High. Semaphore's flow (generating an identity commitment, joining a group, and producing a zero-knowledge proof) is conceptually simple for developers but still complex for non-technical end users. Most users need a web UI or wallet integration to hide the cryptographic steps.
22. **Gas Efficiency** - Medium. Creating an empty group costs approximately 91,596 gas, adding 30 members 177,557 gas, and verifying a proof for 30 members costs 285,541 gas.
23. **Ease of Integration/Deployment** - High. Semaphore is easy to integrate if using existing deployments, and also has ready to use deployment scripts for deploying on new networks.
24. **Implementation Maturity** - High. Semaphore is currently on the 4th version of the protocol so is a mature implementation. It is one of the most popular ZK protocols used on Ethereum.

Shutter - Shielded Voting

Overview

Shutter Shielded Voting is a threshold encryption system for temporary privacy in blockchain governance. Votes are encrypted during the voting period and decrypted only after the vote concludes. It's the most widely used encrypted voting system in DAO governance (notably on Snapshot).

- Website: https://www.shutter.network/shielded-voting
- GitHub/Docker: https://github.com/shutter-network/rolling-shutter/tree/snapshot/livetest/docker
- Mainnet:
    - Shielded Voting on on Snapshot, Snapshot Announcement
    - Shielded Voting available on Shutter API for dApp integration
- Updates/Planned Integrations:
    - Shielded Voting coming soon to Kleros v2
    - Onchain shielded voting PoC

Explanation

Shutter Shielded Voting provides temporary ballot secrecy for DAO governance by encrypting each ballot to a per-proposal public key that is generated by a decentralized "Keyper" committee via threshold cryptography.

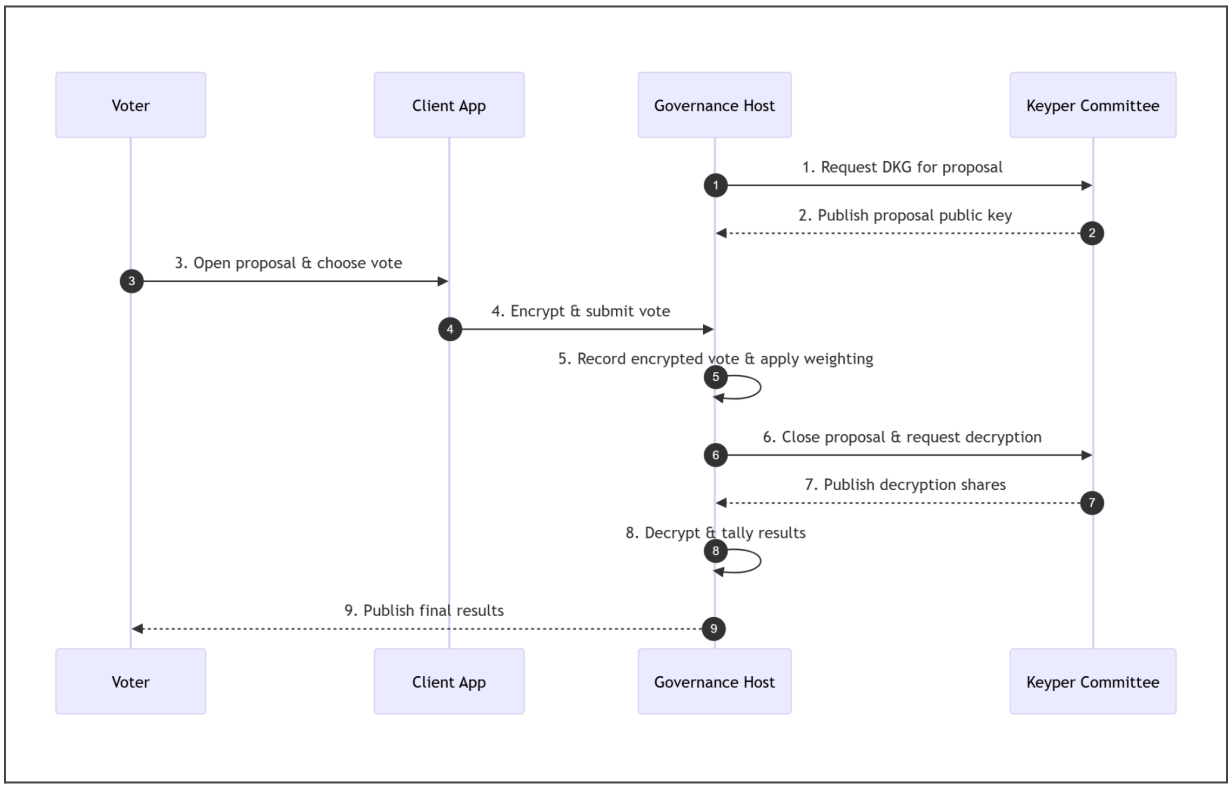
When a proposal opens, the public encryption key is published alongside the proposal metadata (space, proposal ID, strategy configuration), and voters cast ballots exactly as they do today, except the choice payload is encrypted client-side before being submitted and authenticated by the host system (for example, Snapshot) using the voter's wallet signature and whatever eligibility or weighting rules the space already enforces.

During the voting window, no one, including Keypers, can read individual choices, so there is no running tally leakage. After the deadline, the Keypers run a decryption ceremony: each Keyper releases a decryption share, and once the threshold is reached, the proposal-specific decryption key can be derived and applied to all ciphertexts. The host tallies the now-plaintext choices and publishes the result.

Anyone can independently fetch the ciphertexts, the Keyper shares, and the final plaintexts to rederive the key and verify that the tally matches the set of valid ballots. This design composes with existing voting UX and strategies, adds essentially zero friction for voters, and confines new

trust to Keyper liveness and non-collusion during the vote window. Colluding Keypers could decrypt early or a stalled committee could delay decryption, but they cannot forge votes or alter eligibility and weights enforced by the host platform.

A proof of concept is available for onchain shielded voting using a modified OpenZeppelin Governor contract, demonstrating full lifecycle encryption and decryption. Kleros is also testing a beta version of Shielded Voting for juror decisions, scheduled for full production release in November 2025.



Analysis

1. **Correct Execution** - Yes. A dishonest threshold of Keypers cannot produce a false tally—they can (at worst) decrypt early or refuse to release keys. The actual risk lies with whoever performs the decryption and tallying (e.g., Snapshot Hub or a smart contract). Ideally, Shutter - Shielded Voting would be combined with a zero-knowledge proof of correct decryption and tallying. The system functions reliably in practice, and verifiable decryption proofs are planned for future versions.
2. **Robustness** - Medium–High. As long as a quorum of Keypers participate, the system remains operational. Failure or non-cooperation from too many Keypers can delay or block tallying. Has been live for ~3 years on Snapshot with no privacy failures reported.

3. **Censorship Resistance & Liveness** - Yes. Decentralized Keyper setup prevents any single party from censoring votes or blocking the generation of the decryption key, assuming threshold availability. Also, after the Keypers generate the decryption key, anyone may use it to decrypt the votes and reveal the tally.
4. **Coercion Resistance** - No. Voters cannot prove or reveal their choice until the election ends. Voters can also change their encrypted vote during the voting period if the host platform allows it. However, after the voting period ends, all votes are decrypted and revealed publicly - effectively providing coercers with a receipt.
5. **Private Vote** - Partial. Votes are encrypted during the voting period, However, after the voting period ends, all votes are decrypted and revealed publicly, thus tying voter identities to votes.
6. **Running Tally Privacy** - Yes. Votes are encrypted during the voting period and decrypted after the voting period ends. As a result, no early results can be obtained which could influence the remaining voters.
7. **Quorum Status** - Yes. Although the contents of the votes are encrypted during the voting period, the identities of the voters and the voting power deployed are both visible throughout the voting period. As a result, the quorum status is visible throughout the voting period.
8. **Individual Verifiability** - Yes. Each voter can verify their encrypted ballot was recorded and included in the final tally.
9. **Universal Verifiability** - Yes. Anyone can audit the election to confirm that the final results correspond to all valid encrypted votes.
10. **Eligibility** - Yes. Eligibility enforcement is delegated to the host governance system, such as Snapshot's strategy engine, which defines which wallets can vote and how much voting power each one has. Shutter does not modify or extend the eligibility mechanism; it operates at the encryption layer and relies on the host system's existing authentication and weighting logic.
11. **Uniqueness** - Yes. Each eligible voter can submit only one encrypted vote per proposal. The host governance framework prevents duplicate submissions or overwriting unless explicitly supported (e.g., vote updating within the allowed period). Since each encryption is tied to a unique voter signature, double voting or replaying ciphertexts is rejected.
12. **Multi Choice, Fractional Voting** - Yes. Supports multiple choice voting, fractional voting and other weighted voting schemes used in DAOs.
13. **Weighted Voting** - Yes. Supports quadratic voting and other weighted voting schemes used in DAOs.
14. **Vote Updatability** - Yes. Voters can change their encrypted vote during the voting period if the host platform allows it.
15. **Vote Delegation** - Yes. Shutter relies on the host governance platform (such as Snapshot or OZ Governor) for delegation functionality. Delegates can vote on behalf of others, and their votes are encrypted in the same way as any other voter's ballot.

16. **Delegation Privacy** - No. Delegation relationships (who delegates to whom) are visible and can be proven via the state onchain (ERC-20 delegate).
17. **Engagement Disclosure** - Yes. Although the contents of the votes are encrypted during the voting period, the identities of the voters and the voting power deployed are both visible throughout the voting period. As a result, delegate engagement (or lack thereof) is publicly visible.
18. **Vote Disclosure** - No. Votes are public after decryption, and there is no selective sharing with delegators.
19. **Delegation Updatability** - No. Voters cannot re-delegate voting power during an active voting period.
20. **Implementation Simplicity** - Medium. Minimal changes to voting logic; encryption handled transparently.
21. **Ease of Use** - High. Identical to standard voting UX for end users.
22. **Gas Efficiency** - High. As there's no computation over encrypted data or other expensive data storage/computation needed.
23. **Ease of Integration/Deployment** - High. API and SDK integrations are straightforward for dApps and governance frameworks.
24. **Implementation Maturity** - High. Stable and widely used in production on Snapshot and via Shutter API. Onchain PoC for OZ Governor available.

SIV

Overview

Secure Internet Voting (SIV) is an offchain voting system that can be used in government & non-governmental elections as an additional alternative to in-person voting or mail voting. The system guarantees three main security requirements:

1. Authenticated voters: only legitimately registered voters are allowed to vote
2. Private voting: votes are private/hidden
3. Verifiable tallies: results can be independently audited

- Website: https://siv.org
- Production: Yes

Explanation

SIV employs threshold encryption, ElGamal encryption, and zero-knowledge proofs to ensure secure and verifiable elections.

The process begins with voter registration, where the election administrator creates a list of valid voters along with a contact option, such as an email address, mailing address, or phone number.

Next, privacy protectors register for the election. Any entity may apply to act as a privacy protector, but they must first be approved by the administrator. Each approved protector generates a share of the unlocking key, which will later be required for the tallying phase.

Once registration is complete, each voter receives a unique voter authentication token that serves as their identifier for the election. This token can only be used once and must remain private, stored securely on the voter's local device.

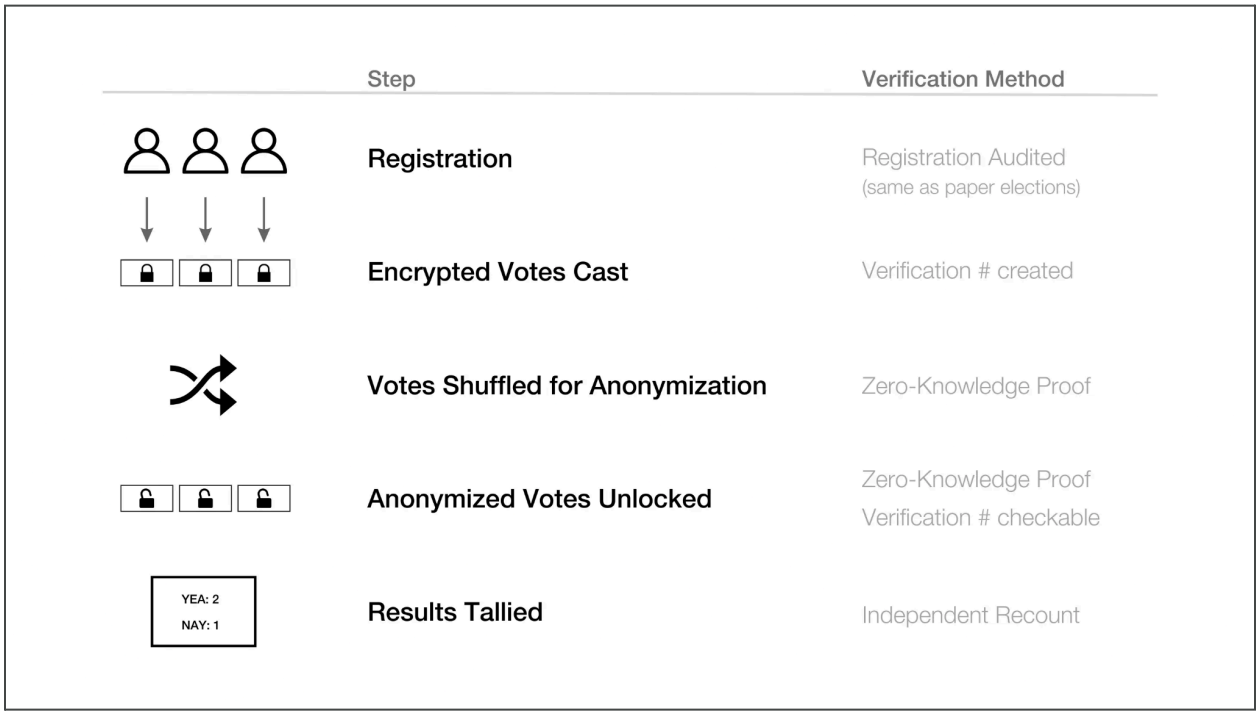
When it is time to vote, the voter's device generates a verification number. The voter then selects an option for each question on the ballot, appends the verification number to each choice, encrypts the resulting votes, and submits them to the SIV system together with their voter authentication token. These submissions are added to the votes list.

To preserve anonymity, a central system removes all voter authentication tokens from the list, leaving only the encrypted votes. At this stage, each privacy protector sequentially processes the list by shuffling the order of the votes, re-encrypting each one using their local public key share,

generating a zero-knowledge proof of their operation, and publishing the updated shuffled list for the next protector to continue the process.

After the shuffling phase concludes, all privacy protectors combine their key shares to decrypt the final list of votes. The decrypted data contains each vote option along with its corresponding verification number. These results are published openly so that anyone can verify and tally the final outcome.

For auditability, external auditors can conduct Risk-Limiting Audits to confirm that every vote was counted without manipulation. This involves selecting a random sample of voters and asking them to verify that their vote and corresponding verification number appear correctly in the published list.



Analysis

1. **`Correct Execution`** - Yes. After the election is over, anyone can check the results and verify its correctness by using "Risk-Limiting Audits". This method consists of checking a random sample of votes and verifying with the voters that the chosen option is the one published (without compromising vote privacy). The auditor can be statistically confident that the submitted votes are correct depending on the size of the analyzed sample.
2. **`Robustness`** - Medium. SIV depends on centralized storage to keep the votes list available at all times. The vote list could be distributed across multiple redundant servers but the vote entry point will still be a single central server that needs to share a new

addition to the different servers. The decentralized network of privacy protector entities guarantee a correct tallying computation even if *n-1* protectors go rogue or offline. There are plans to decentralize the storage using redundant data backups (e.g. Storj or Filecoin) or incentives-based systems.

3. **Censorship Resistance & Liveness** - No. The SIV system administrator could censor votes by blocking append requests based on the voter auth tokens. The votes entry point is a single centralized point-of-failure. If this happens, it can be caught: a) By voters checking on their own using their Verification # against the published tally, or b) with high confidence as part of a statistical post-election audit. This property could be improved if votes were signed with private keys instead of the current symmetric auth tokens, for more decentralized submission channels.
4. **Coercion Resistance** - Yes. Previously, the voter could prove how they voted by showing anyone their verification number and checking out the decrypted votes list, which contains their vote option next to their verification number. There is a newly added feature for anti-bribery/coercion that consists of the user working with the poll authority to secretly cancel their coerced vote and resubmitting a new one. The authority produces a zero-knowledge proof to guarantee that the old vote was discarded and the new vote will be counted. More info here. There is also a "Vote Seller's Dilemma" game-theory solution too, for law-enforceable votes.
5. **Private Vote** - Yes. No one can see voters' choices—not even the centralized server receiving the votes if there is a set of privacy protectors active for that particular election. The encrypted vote is published for anyone to look at, but the content of the vote is not visible until it is decrypted. The decrypt-shuffle process ensures a voter cannot be linked to choice - even after decryption.
6. **Running Tally Privacy** - Yes. The tally remains hidden until results are revealed. Once the voting period is over, the privacy protectors decrypt votes and publish the votes list. From there anyone can tally the results. During the voting processes, the encrypted votes list is public so anyone could see how many votes have been cast.
7. **Quorum Status** - Yes. SIV allows anyone to verify how many votes have been published (quorum status) during the voting process because anyone can check the number of submitted votes (even though the values are encrypted). In the case of an election with the coercion resistance feature, there are two lists: one for published votes and one for override votes. Therefore the quorum status is still visible in elections using this feature.
8. **Individual Verifiability** - Yes. Voters can check that their vote was included by using their verification number and the public votes list. This also allows external auditors to verify correct execution by asking individual voters to verify that their choice is next to their verification number in the final decrypted vote list.
9. **Universal Verifiability** - Yes. The privacy protectors shuffle the encrypted votes list and generate a zero-knowledge proof of correct execution. Anyone could download the zero-knowledge proofs, verify them individually and sequentially, and be certain that the final list is a shuffled + re-encryption of the initial list..

10. **Eligibility** - Yes. Only authorized voters can submit their vote by using the voter auth token provided in their invitation. The system expects the election authority to define the voters list, with no duplicates or ghost records that could be used by a specific party to submit more than one vote. To mitigate an illegitimate voter roll: it can be efficiently audited with random sampling. The SIV Protocol is also compatible with a list of public keys too, if available.
11. **Uniqueness** - Yes. Voters can only cast their vote once using their unique vote auth token.
12. **Multiple Choice, Fractional Voting** - Yes. SIV can support multiple voting methods (normal voting, weighted voting, rank-based voting, quadratic voting, etc) because the ballot questions and answers are open. The final votes list is published so the results calculation can be performed in different ways, in plaintext.
13. **Weighted Voting** - Yes. SIV can support multiple voting methods because the ballot questions and answers are open. The final votes list is published so the results calculation could be performed in different ways. SIV is currently all designed around One-Person-One-Vote, not Token-Weighted Voting.
14. **Vote Updatability** - Yes. Voters can get their original vote invalidated, and new credentials issued to re-vote, with the election admin's help. This public list of invalidated auth tokens is auditable, and useful for benign mistakes, lost credentials, etc. There is a separate option of Verifiable Private Overrides, for voters facing more serious coercion or bribery threats. These Overrides instead use zero-knowledge proofs, hiding which auth token they correspond to, to balance privacy & integrity. More info here.
15. **Vote Delegation** - Possible. SIV does not have native features for delegation, although it is possible to manually forward vote credentials to people to vote on your behalf (assuming election rules allow it). For a small number of received votes this is easy, but it would become tedious without automation if sent many votes.
16. **Delegation Privacy** - Yes. Manual delegation would be mostly private, although without additional safeguards, the server could see many encrypted votes coming from the same IP & device user-agents.
17. **Engagement Disclosure** - Yes. A voter who manually forwards their vote credentials could check whether that auth token ended up being used or not.
18. **Vote Disclosure** - No. The original voter would not be able to see how the delegate chose to cast their forwarded vote.
19. **Delegation Updatability** - Yes. Once a vote credential is sent to one delegate, the same token could be sent to another, but it would be a race-condition for who uses it first. The original voter could ask the election administrator to invalidate their first auth token, and be re-issued a new one for their new choice.
20. **Implementation Simplicity** - Medium. SIV requires three components to run correctly: a central system to store votes (it can be decentralized using Storj or Filecoin as described above), a set of privacy protectors to shuffle the votes list and a client application for users to vote. The votes encryption scheme and the shuffling

zero-knowledge proofs are well-established and secure. SIV works as a SaaS so election authorities do not need to worry about implementation. The privacy protector can download the service locally or use a browser web application that does not require custom installation. The future roadmap focuses on more detailed documentation and ease of use.

21. **Ease of Use** - High. Anyone can create an admin account in the SIV website and wait for the team to approve it and help run an election. SIV voters can submit their vote in seconds by following their custom link or using an election's public link, no installs needed. In scenarios where people are already together in person, the election authority can print unique QR codes for each voter to scan from their devices and submit their votes.
22. **Gas Efficiency** - N/A. SIV does not use any blockchain network so there is no gas to be paid.
23. **Ease of Integration/Deployment** - Low. SIV is a non-Web3 voting system. There are no active plans for integrations or modularity features, but SIV is very open to new ideas and collaborations. The shuffle mechanism which provides privacy to the system cannot be truly replicated onchain because every time a voter submits a vote, the data gets registered on the onchain transaction. So after N shuffles, there is still the possibility of linking voters to their votes. An external coprocessor, a private rollup or an offchain service mechanism could be implemented to mitigate this challenge.
24. **Implementation Maturity** - High. According to their blog you can see that SIV has been used in multiple local US elections. Approx 30,000 votes cast so far in binding elections, including for electing a US Congressperson in a special election. SIV also held a big "HACK SIV" Challenge at DEFCON, where many professional and academic election security experts looked for vulnerabilities in the protocol & implementation. It's also being adopted by organizations like ZCash for governance voting.

Incendia

Overview

Incendia (“Burn Your Vote”) is a recently proposed protocol for on-chain anonymous voting system without heavy cryptography such as homomorphic encryption or MPC. The way it works is voters “burn” tokens to a pseudorandom address and prove via a zero-knowledge proof that the burn corresponds to a legitimate vote. The vote is stored in plaintext in the contract. The system is fully on-chain: proof verification and vote aggregation occur in smart contracts. Votes are unlinkable to identities, but the tally is visible in real time and individual votes are plaintext. The unique feature of Incendia is the simplicity from not using expensive cryptography and coming up with a scheme that tallies via smart contracts, therefore negating the need for a centralised or decentralized tallying authority. One main downside of Incendia is that it does not provide ongoing tally privacy as votes are tallied in the open.

- Website: http://incendia.tech/
- Paper: https://eprint.iacr.org/2025/1022
- Ethresearch: https://ethresear.ch/t/incendia-onchain-anonymous-voting-via-proof-of-burn/22749
- GitHub: https://github.com/zero-savvy/burn-to-vote
- Testnet: No
- Mainnet: No

Explanation

Incendia is a novel use of the proof-of-burn mechanism: voters irreversibly burn tokens onchain and generate a zero-knowledge proof attesting to a valid burn (tokens sent to an un-spendable address). To prevent double voting, each submission includes a nullifier derived deterministically from private data, 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 onchain, allowing any observer to independently audit the outcome.

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.

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.

The public and transparent tallying of plaintext votes enables support for a wide range of voting models, including majority, token-weighted, quadratic and ranked-choice voting.

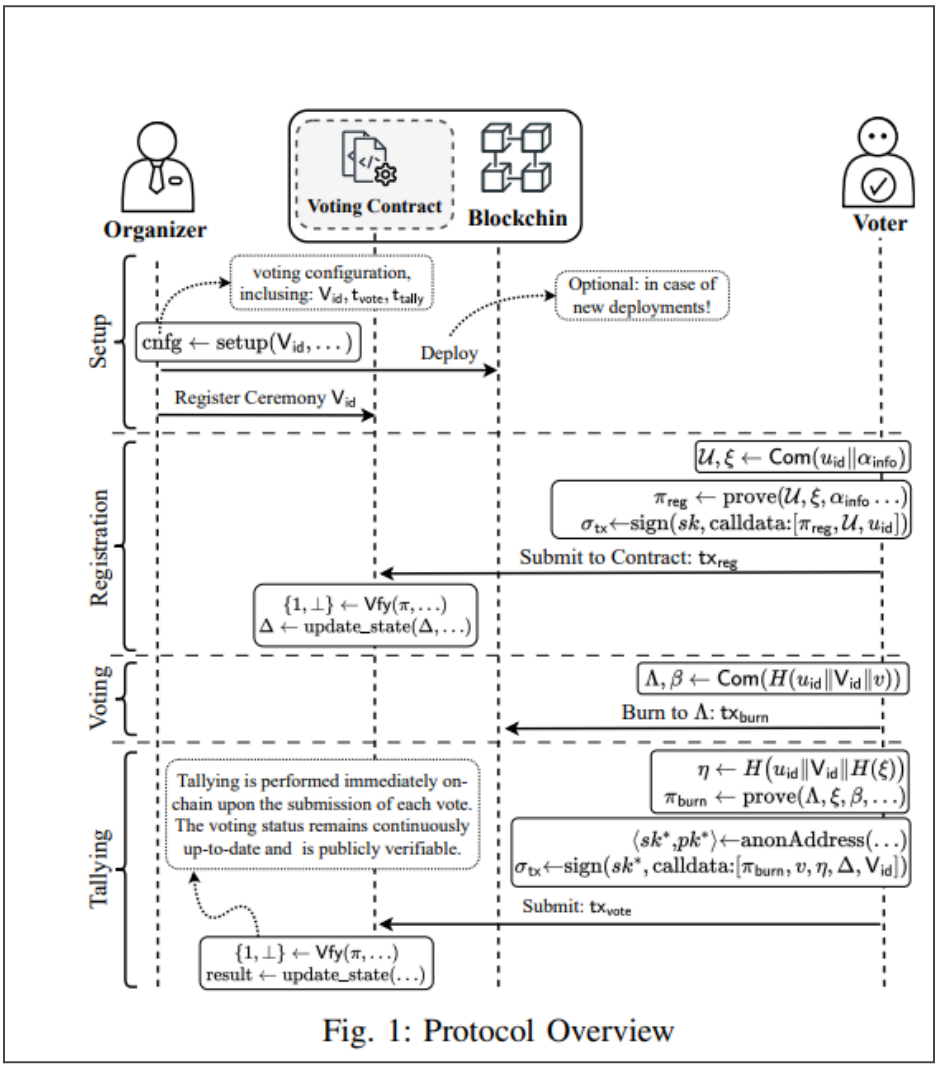


Fig. 1: Protocol Overview

Analysis

1. **Correct Execution** - Yes. The cast votes are public so anyone can tally them and verify the correct results.
2. **Robustness** - High. The system is highly robust since it relies on established tooling for zero-knowledge proofs (circom + groth16) and vote posting and tallying occurs directly onchain so is secured by smart contracts only. There is minimal heavy cryptography and no offchain infra that can be compromised
3. **Censorship Resistance & Liveness** - Yes. High censorship resistance since the protocol relies on onchain tallying using smart contracts and no external infra or networks

4. **Coercion Resistance** - Partial. The proof-of-burn mechanism allows each voter to perform multiple independent burns to random, unlinkable addresses. This means they can submit 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. This doesn't lead to double voting since the nullifier derivation means that they can submit at most one vote. A post-submission re-voting feature can be enabled which permits the overwriting of previously submitted ballots, further increasing coercion resistance. 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
5. **Private Vote** - Partial. Before tallying starts, the votes are not visible, it is during tallying that votes become visible as plaintext. Votes are in plaintext, but are unlinkable to identities. Proof of burn benefits from having a large anonymity set. Even after the registration phase, the number of participating voters remains hidden, as voting occurs through indistinguishable burn transactions. Voters can submit their votes through a relayer, or even a CEX. While votes are submitted in plaintext alongside proofs, identities are not linked to these votes
6. **Running Tally Privacy** - No. After the voting phase concludes, users submit their votes along with the corresponding burn proofs to the smart contract for verification. Tallying happens onchain and is not encrypted, so the tally result becomes visible once tallying commences
7. **Quorum Status** - Yes. Because the cast votes are submitted as plaintext, it is trivial to support a running quorum status with smart contract logic.
8. **Individual Verifiability** - Yes. All votes and associated proofs are published and checked onchain. A voter can see and verify that their corresponding proof-of-burn + vote option was correctly added to the cast votes.
9. **Universal Verifiability** - Yes. It is possible to verify that all votes came from a valid proof of burn and they were correctly tallied onchain. This is possible by examining zero-knowledge proof verification events and checking the tally was counted correctly onchain
10. **Eligibility** - Yes. Voters can register before a vote starts. Each user must provide a proof of inclusion in an allow-list Merkle tree. The anonymity set is larger than the set of registered voters
11. **Uniqueness** - Yes. Double spending is prevented to guarantee uniqueness. To prevent double voting, each submission includes a nullifier derived deterministically from private data, which the smart contract checks for uniqueness. The secrets used for inclusion for registration are also used to define the burn address, ensuring a binding relationship between eligibility and voting.
12. **Multiple Choice, Fractional Voting** - Yes. Because the cast votes and the final results are public, any voting model can be implemented (e.g. majority, token-weighted,

quadratic and ranked-choice voting). The protocol supports any tallying logic expressible in smart contract's logic.

13. **Weighted Voting** - Yes. Because the cast votes and the final results are public, any voting model can be implemented (e.g. majority, token-weighted, quadratic and ranked-choice voting). The protocol supports any tallying logic expressible in smart contract's logic.
14. **Vote Updatability** - Yes. Post submission revoting allow overwriting previously submitted ballots
15. **Vote Delegation** - Yes. Incendia supports delegation through a burn-to-delegate mechanism. When an eligible voter performs a burn, 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 can cast a vote on behalf of the voter. Front-running can be prevented by including the address of the desired delegate in the proof.
16. **Delegation Privacy** - Yes. When the proof-of-burn is transferred to a delegate, the delegate does not learn the identity of the voter.
17. **Engagement Disclosure** - Yes. Since votes in Incendia are submitted in plaintext, when a delegate uses a burn-to-delegate to cast a vote, the transaction discloses which way the delegate votes. There is no separate mechanism to selectively disclose engagement, so delegate participation is automatically visible.
18. **Vote Disclosure** - No. When delegating, the actual vote is cast when the vote submission transaction is executed, where the transaction sender provides the proof-of-burn and specifies the voting choice. The voting choice of the delegate is publicly visible—not visible exclusively to delegators.
19. **Delegation Updatability** - No. In the proposed implementation, it is not possible to revoke the proof-of-burn that was sent to someone else earlier. Therefore, delegation updateability is not supported. If it was to be implemented, the nullifier design is such that the second delegate would use the same nullifier as the first delegate. This would preserve the anonymity of the second delegate, but someone other than the original burner may be able to figure out the vote value of the second delegate
20. **Implementation Simplicity** - Medium. The implementation requires zero-knowledge proofs (currently written in Circom but is being migrated to Noir) and user interfaces (CLI, backend server or a frontend website)
21. **Ease of Use** - Medium. Voters will need to perform 3 actions: burn tokens, generate proof and submit a vote option with proof. The organizer will have to integrate Incendia into its system (e.g. DAO). While this is relatively straightforward, 3 actions is higher than some other protocols.
22. **Gas Efficiency** - High. Because no heavy cryptography or significant zero-knowledge proof verification is involved, Incendia is an affordable option compared to other projects. Benchmarks for a simple yes/no vote submission indicate the function to submit a vote:

submitVote, would consume approximately 300k gas. For the burn itself, a burn transaction is implemented as a standard token transfer with minimal gas costs (21K gas). Additionally, the burn amount can be small (e.g. 0.000001 ETH)

23. **Ease of Integration/Deployment** - Medium. Incendia can be used with governance tokens, which most DAOs use. Given the project is at research stage, the tooling around it is not well developed.
24. **Implementation Maturity** - Medium. Proof-of-burn is relatively new and there have been some potential risks. That said, it is relatively simple in many respects.

DAVINCI Protocol

Overview

Created by Vocdoni, DAVINCI is a voting protocol designed to provide secure, verifiable, and anonymous digital voting. It is envisioned DAVINCI can be utilised outside of the Web3 sphere, for also progressing civil society. The Vocdoni protocol is implemented as an L2 for verifiable e2e elections. It combines zero-knowledge proofs and threshold homomorphic encryption (ElGamal) to encrypt ballots and aggregate them privately. Sequencers run distributed key generation (DKG) to create a public key and store votes. They publish zero-knowledge proofs that updates are valid. The final tally is decrypted via threshold decryption.

- Website: https://davinci.vote/
- Whitepaper: https://whitepaper.vocdoni.io/
- Testnet: No
- Mainnet: No

Explanation

DAVINCI is a zero-knowledge proof based voting rollup built for privacy, scalability, and verifiability. Operating as a Layer-2 system, it uses decentralized sequencers for coordination and Ethereum for settlement and data availability, ensuring transparent and tamper-evident elections.

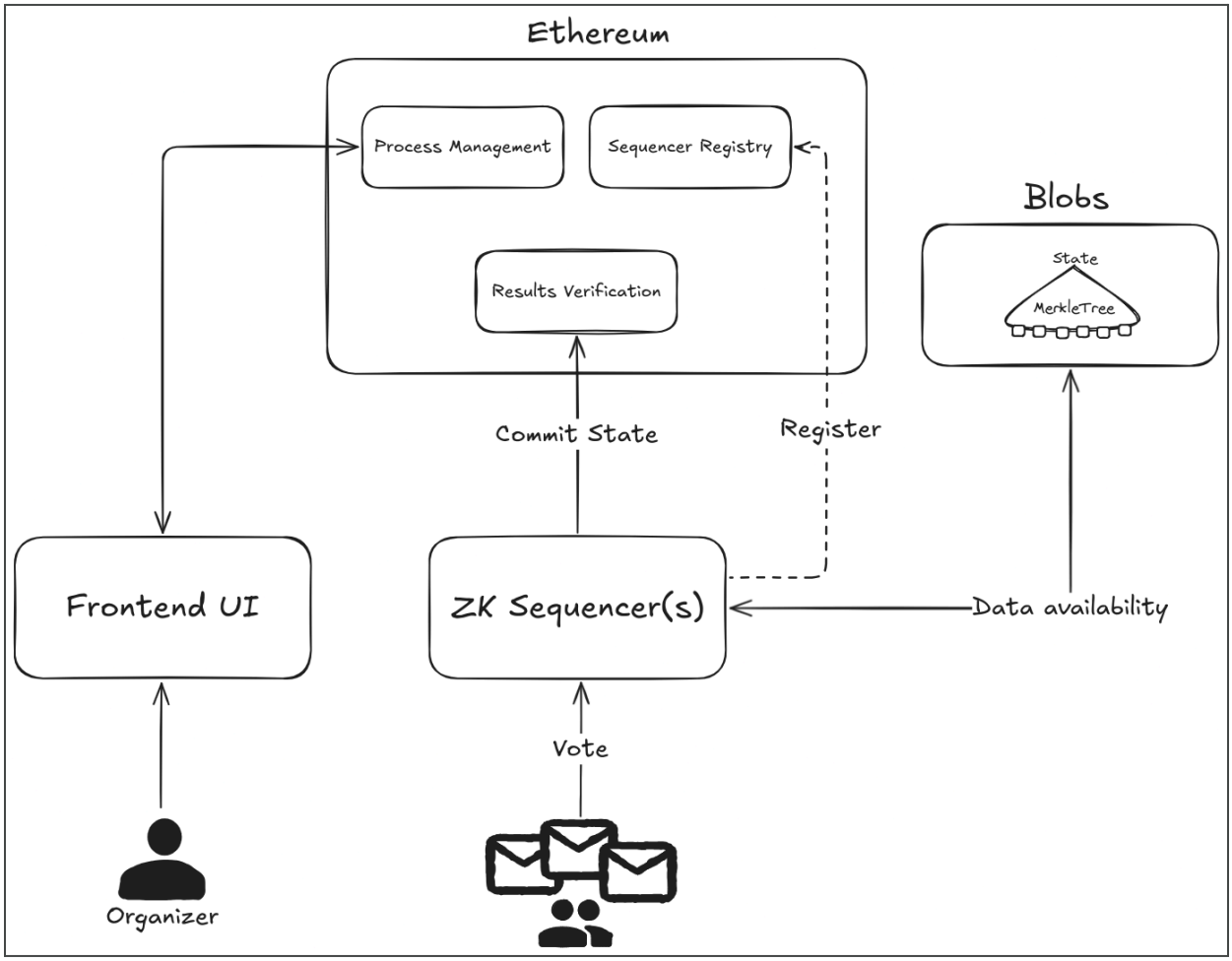
The process begins when an organizer creates a new voting event. They define the voter registry (the “census”), parameters such as duration and options, and submit them to Ethereum smart contracts that manage lifecycle and verification. Both the census and process states are stored as Merkle trees whose roots are anchored onchain.

Sequencers jointly perform a distributed key generation to produce a threshold encryption public key. Each sequencer holds a share of the decryption key, ensuring that no single participant can decrypt votes. DAVINCI uses ElGamal-based homomorphic encryption so tallies can be computed without exposing individual ballots.

When voting opens, a voter retrieves a Merkle proof of eligibility, encrypts their choice with the election’s public key, and generates two zero-knowledge proofs—one proving ballot validity and another proving eligibility without revealing identity. The encrypted ballot and proofs are submitted to a sequencer.

Sequencers verify proofs, update the election state, and generate zero-knowledge proofs that attest to correct state transitions, single-vote compliance, and proper accumulation of encrypted votes. These updates are committed onchain along with data stored in Ethereum blobs under EIP-4844, where smart contracts verify each proof before finalization.

After voting closes, sequencers publish decryption shares. Once the threshold is met, the collective key is derived, encrypted totals are decrypted, and results are verified against the onchain state. DAVINCI's design achieves receipt-freeness, censorship resistance, and auditability while supporting frequent, low-cost voting at scale.



Analysis

1. **Correct Execution** - Yes. Every step is accompanied by zero-knowledge proofs verified onchain, ensuring only valid votes are counted and the final tally is mathematically correct. Sequencers can't falsify or alter results since any invalid proof would be rejected by the smart contract. A false tally is only possible if the underlying security assumptions (such as zero-knowledge proof soundness or encryption security) are broken.
2. **Robustness** - High. Multiple sequencers, threshold encryption, and Ethereum settlement provide strong fault tolerance, but liveness still depends on enough sequencers remaining active to complete decryption and state updates.
3. **Censorship Resistance & Liveness** - Yes. Multiple sequencers can independently verify and post proofs to Ethereum, preventing any single party from censoring votes or

halting tallying, as long as the minimum threshold of sequencers required for key generation and decryption stays alive.

4. **Coercion Resistance** - Yes. DAVINCI achieves coercion resistance through receipt-freeness by combining ElGamal re-encryption and stealth vote overwriting. Sequencers re-randomize each ballot so voters cannot produce a verifiable record of what-how they voted, and voters can resubmit new ballots that overwrite previous ones. It is not possible for a third party observer to differentiate between an overwrite and a random reencryption. This design makes it impossible for a briber to confirm whether a coerced vote was kept or later replaced.
5. **Private Vote** - Yes. Each vote is encrypted with threshold ElGamal and proven valid via zero-knowledge proofs, so no one can see an individual's choice. The system decouples voter identities from their encrypted ballots using cryptographic hashes and nullifiers. Even if ballots are later decrypted, they cannot be linked back to individual voters.
6. **Running Tally Privacy** - Yes. Every vote is encrypted using threshold homomorphic encryption (ElGamal). Until the voting period ends and a threshold of sequencers publishes their decryption shares, the contents of any individual or aggregate ballot remain completely hidden.
7. **Quorum Status** - Yes. Weights can be extracted from the census Merkle-tree based on the votes' addresses.
8. **Individual Verifiability** - Yes. Every voter can verify that their vote has been correctly recorded and included in the election tally. Instead of locating their encrypted ballot directly (which may be re-encrypted during processing), voters use a unique voteID to confirm that their submission is part of the set of valid cast votes, ensuring their participation is registered without revealing their choice
9. **Universal Verifiability** - Yes. The system publishes zero-knowledge proofs and state roots on Ethereum. Anyone can independently verify that the final decrypted tally matches the encrypted ballots committed onchain, confirming that the results follow directly from all recorded votes.
10. **Eligibility** - Yes. The system enforces census-membership through a Merkle-based census or verifiable credentials. Each voter must provide a valid Merkle proof of inclusion or credential during voting, and zero-knowledge proof verification ensures only registered voters in the approved census can cast a ballot.
11. **Uniqueness** - Yes. DAVINCI allows voters to overwrite their previous votes, but the system maintains uniqueness by detecting existing nullifiers. When a new vote is submitted, the Sequencer subtracts the old encrypted ballot from the tally, adds the new one, and updates the state Merkle tree, ensuring only the latest vote counts.
12. **Multiple Choice, Fractional Voting** - Yes. The Vocdoni Ballot Protocol supports a variety of voting methods such as multiple choice, quadratic voting, range, rank, approval. Each vote defines how a ballot should be formed. Each ballot contains one or more fields which a voter must fill out. The votes choices are represented as a sequence of numbers, with each value corresponding to the voter's choice for a respective

field. Therefore a voter can submit multiple choices and assign fraction weights to different options.

13. **Weighted Voting** - Yes. The Vocdoni Ballot Protocol supports various voting models, including ranked choice, quadratic voting, and budget voting, by allowing configurable ballot fields and weighting parameters that let votes vary in strength based on predefined criteria.
14. **Vote Updatability** - Yes. DAVINCI allows voters to overwrite their previous votes, but the system maintains uniqueness by detecting existing nullifiers. When a new vote is submitted, the Sequencer subtracts the old encrypted ballot from the tally, adds the new one, and updates the state Merkle tree, ensuring only the latest vote counts.
15. **Vote Delegation** - Possible. DAVINCI protocol doesn't hard-code delegation, but it's built to be modular and schema-free, meaning organizers can define custom vote delegation logic (private or non-private) using smart-contract or offchain rules that the zero-knowledge proof circuits will enforce.
16. **Delegation Privacy** - NA. Depends on custom vote delegation logic.
17. **Engagement Disclosure** - NA. Depends on custom vote delegation logic.
18. **Vote Disclosure** - NA. Depends on custom vote delegation logic.
19. **Delegation Updatability** - NA. Depends on custom vote delegation logic.
20. **Implementation Simplicity** - Low. DAVINCI's architecture is highly complex, involving multiple zero-knowledge proof circuits, distributed key generation, threshold encryption, and Ethereum smart contracts.
21. **Ease of Use** - Medium. DAVINCI inherits Vocdoni's emphasis on accessibility and user-friendly design, and its MVP demonstrates a functional voting interface; however, the final user experience across all platforms remains untested and not fully defined. Vocdoni is implementing a typescript SDK which will enable the easy integration of DAVINCI into its existing voting service. Once this is done, using DAVINCI will have the same UX as https://vocdoni.app
22. **Gas Efficiency** - High. A state transition currently batches 60 votes and costs approximately 350k gas.
23. **Ease of Integration/Deployment** - Medium-High. The system is modular, consisting of interchangeable components that can be integrated with external systems via adaptable interfaces, allowing seamless third-party integration through voting-as-a-service APIs, but setup still requires coordination with DAVINCI's sequencer, DKG, and zero-knowledge proof infrastructure. Vocdoni also provides a typescript SDK that makes integration relatively straightforward. The network is autonomous, so anyone using or building on it, does not need to worry about deployment or maintenance of network components. Using the SDK and the API abstracts away a lot of complexity..
24. **Implementation Maturity** - Medium-Low. While DAVINCI's foundation builds on the mature Vocdoni platform, the new zkRollup version is only deployed on testnet, and has not yet launched a production deployment, indicating limited real-world maturity so far.

The full system is deployed on Sepolia, however, the DKG is still missing it is not fully public. Public testnet with incentives, is expected by December or January 2026.

Aragon / Aztec Private Voting Research

Overview

Aragon and Aztec Research explored a system for ballot-secret voting using storage proofs and time-locked cryptography in collaboration with Nouns DAO. The research question was whether a decentralized, fair, weighted, ballot-secret voting system could be achieved on Ethereum. The approach relies on storage proofs and time-lock services to ensure that votes remain secret until a deadline. Participants prove membership using zero-knowledge proofs and cast encrypted votes. A time-locked service prevents early decryption. The system aims to provide anonymity, vote confidentiality and outcome confidentiality, but the complexity of time-lock services and the absence of full receipt-freeness are challenges

- General Report: https://research.aragon.org/nouns.html
- Technical Report: https://research.aragon.org/nouns-tech.html
- GitHub: https://github.com/aragonzkresearch/nouns-anonymous-voting
- Testnet: No
- Mainnet: No

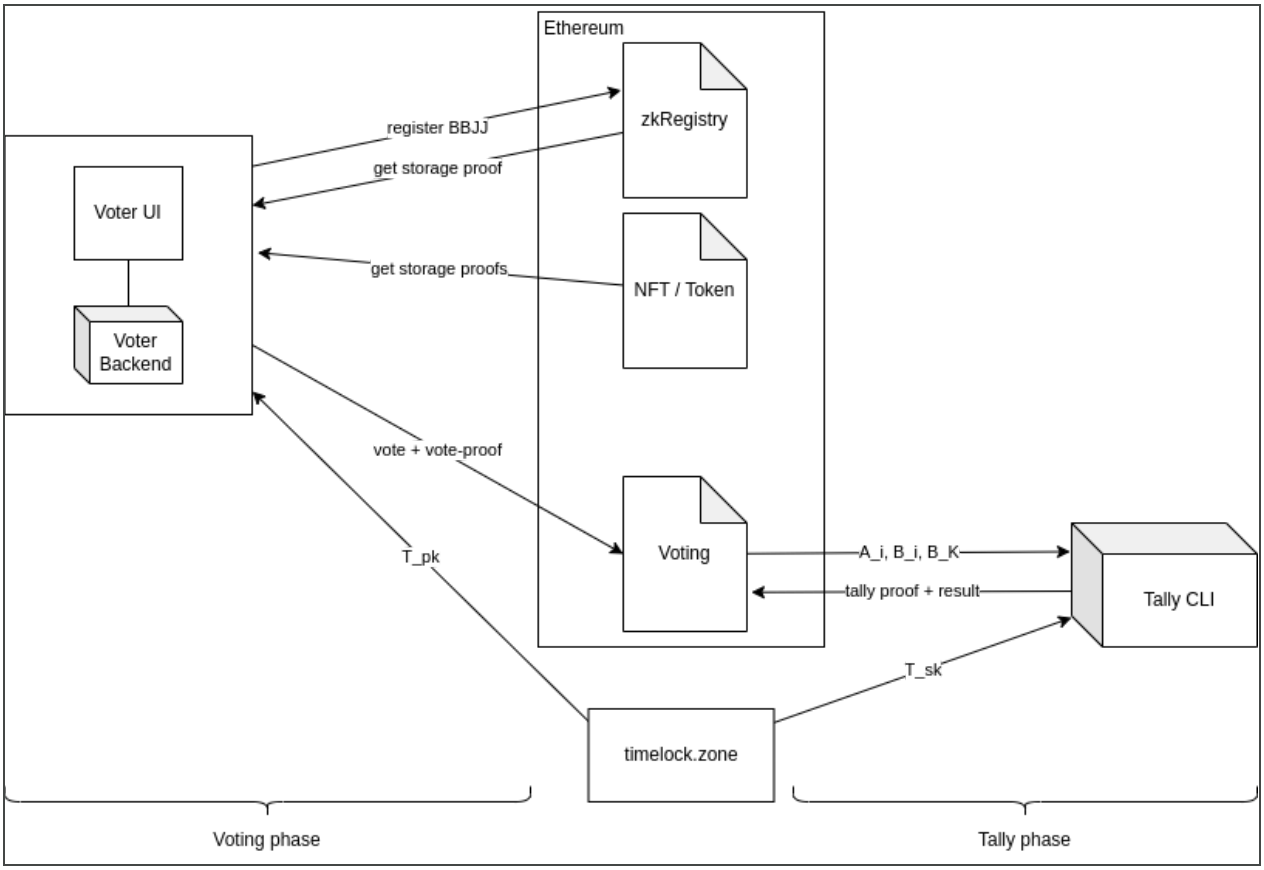
Explanation

Before the voting process begins, every wallet must be registered. To qualify as an eligible voter, a wallet must hold an NFT, which can be owned directly or through delegation. The wallet owner does not need to maintain any additional state, as the keypair is generated deterministically. Registration involves connecting MetaMask to the zkRegistry web app, registering the public key, and copying the private key, which will later be required for casting a vote.

Proposal creation is entirely permissionless. Anyone can create a new voting process, or proposal, using the command-line interface (CLI). This allows participants to initiate governance actions without centralized approval or intervention.

During the voting phase, registered wallets can generate their votes along with the corresponding zero-knowledge proofs. These votes are then submitted directly to the Nouns Voting Smart Contract (VSC). Both the generation and submission steps are handled through the CLI, ensuring a streamlined process for participants.

Once the voting period concludes, the decryption key is released by timelock.zone. At that point, anyone can verify the outcome by independently tallying the results. This final step, also executed through the CLI, ensures that the entire voting process remains transparent, auditable, and verifiable by any observer.



Analysis

1. **`Correct Execution`** - Yes. The use of zero-knowledge proofs ensures that votes are tallied correctly without revealing individual votes. If the protocol is followed correctly, no participant can falsify the tally.
2. **`Robustness`** - Medium. All ballots and proofs are submitted directly to the voting smart contract. There is no off-chain coordinator or final aggregator responsible for tallying. Once the voting period ends, any third-party can run the time-lock solver to decrypt the aggregated ballots and submit the zero-knowledge tally proof. We must assume the drand functionality powering timelock.zone works as expected. The League of Entropy are known organizations.
3. **`Censorship Resistance & Liveness`** - Yes. Anyone can create, vote, and tally directly on Ethereum without central control. However, its resilience is limited by dependencies on timelock.zone and missing tooling.
4. **`Coercion Resistance`** - No. The authors of the report state that allowing a vote recast and/or making proofs non-deterministic to handle coercion resistance falls under future work.

5. **Private Vote** - Yes. Fully private. Ballots are encrypted using the timelock.zone public key. This time-lock encryption scheme hides vote contents until the designated time, ensuring ballot secrecy Only the tallying process (via zero-knowledge proof) reveals the final outcome without exposing individual votes.
6. **Running Tally Privacy** - Yes. The tally remains hidden until decrypted and proven via zero-knowledge proof. The protocol can prove correctness of tally without exposing individual votes.
7. **Quorum Status** - Yes. The implementation allows voters to vote for 3 options: yes, no, abstain. Since one address would have one vote each, it is possible to infer the quorum by examining the number of votes submitted onchain. Note that it is not an explicit feature.
8. **Individual Verifiability** - Yes. Voters can verify their vote was submitted (e.g. via nullifier or commitment), but cannot see the contents.
9. **Universal Verifiability** - Yes. The final tally is backed by a zero-knowledge proof showing correct computation from all valid encrypted votes..
10. **Eligibility** - Yes. Only eligible voters (via identity commitments or whitelisting mechanisms like Semaphore, ERC-4337 identity, etc.) can cast a vote. Aztec supports custom identity schemes.
11. **Uniqueness** - Yes. Nullifier logic prevents double voting. Each user can vote once, and only the latest vote counts (if revoting is enabled). Double-voting attempts can be filtered or invalidated.
12. **Multiple Choice, Fractional Voting** - No. The PoC supports a simple multiple choice of 3 options: Yes, No, Abstain. However, fractional voting is not possible. Weighted voting was identified as future work.
13. **Weighted Voting** - No. This feature was not added, but it is another feature that would be possible to add on. Voting weights could be tied to token balances or identity-based allocation. The cost of vote strength can follow square root functions or other custom logic.
14. **Vote Updatability** - No. This feature was not included in the implementation. Only the latest vote is counted. However, it would be feasible by submitting a new encrypted vote with the same nullifier.
15. **Vote Delegation** - Yes. Delegation is supported—any eligible voter may delegate their vote to a delegate. This is done by delegating your NFT to a chosen delegate.
16. **Delegation Privacy** - No. Delegation status is stored in the Nouns token contract.
17. **Engagement Disclosure** - No. Delegates submit ballots anonymously like any voter. The system does not record or publish whether a particular delegate has participated.
18. **Vote Disclosure** - No. Ballots are encrypted, and the tally proof reveals only aggregated counts. Delegates cannot prove to their delegators how they voted, and delegators cannot verify the delegate's individual vote. The protocol deliberately avoids any receipt or proof of individual vote content to maintain ballot privacy.

19. **Delegation Updatability** - No. The delegation state is captured at registration. After the voting process begins, a voter cannot change their delegate, so there is no mechanism to re-delegate mid-vote
20. **Implementation Simplicity** - Low. The system relies on advanced cryptographic tooling, zero-knowledge proofs, custom identity verification, and encrypted state. Requires familiarity with Noir for circuit design.
21. **Ease of Use** - Medium. For users, voting can be simple if a good UI is provided. Users only need to register their wallet and cast their vote, which is relatively straightforward.
22. **Gas Efficiency** - Low. Gas costs are tied to the cost of verifying an Noir UltraPlonk proof, which are expected to decrease as Noir and the proof system are optimized. Registering a wallet in the zkRegistry costs about 45k gas. Creating a voting process costs roughly 700k gas. Submitting a vote (i.e., sending the encrypted ballot and proof) consumes about 690k gas. Finally, submitting the tally proof costs around 522k gas.
23. **Ease of Integration/Deployment** - Low. Timelock.zone is a time-locked cryptography service that would need deploying in order to use the protocol. The codebase and components also need auditing and deploying on a chosen network.
24. **Implementation Maturity** - Low. Aztec and Aragon's zero-knowledge proof voting work is experimental. Not production-grade. It was implemented as a research sprint, so the implementation would require productionizing before it could be used safely.

Cicada

Overview

Cicada is a protocol proposed by a16z that utilises homomorphic time-lock puzzles to achieve private on-chain voting without trusted tallying authorities. Voters encode their votes as time-lock puzzles that can be homomorphically combined. The final puzzle is solved after the election to reveal the total while individual ballots remain hidden. The scheme provides ballot privacy, running tally privacy and voter anonymity without threshold decryption. However, solving a single time-lock puzzle for each option can be computationally intensive, and the scheme lacks a mechanism for revoting or receipt-freeness.

- GitHub: https://github.com/a16z/cicada
- Paper: https://eprint.iacr.org/2023/1473.pdf
- Blog Post: https://a16zcrypto.com/posts/article/building-cicada-private-on-chain-voting-using-time-lock-puzzles/
- Ethresearch: https://ethresear.ch/t/cicada-private-on-chain-voting-from-homomorphic-time-lock-puzzles/15748
- Testnet: No
- Mainnet: No

Explanation

Cicada is a private onchain voting protocol built around homomorphic time-lock puzzles and zero-knowledge proofs.

It begins with the election organizer deploying an election contract onchain, which defines public parameters including an RSA-modulus (or similar trapdoor-resistant setting), a time parameter T for the time-lock puzzles, and the allowed vote domain (for example “0” = no, “1” = yes).

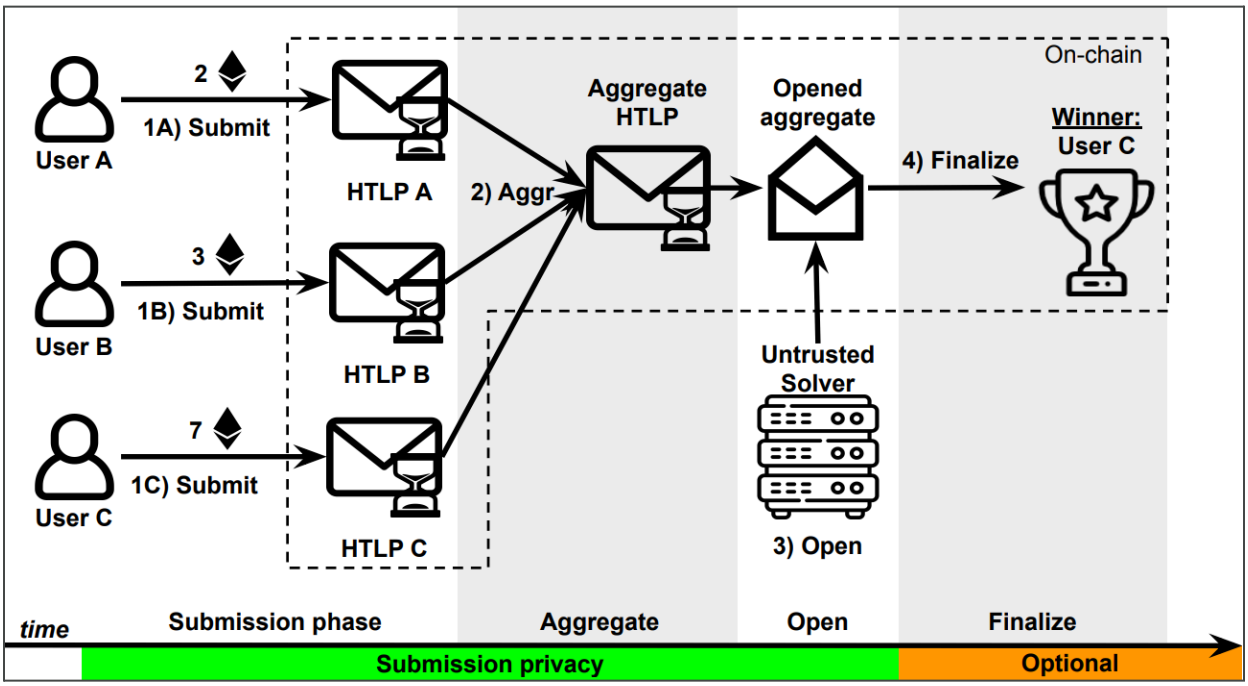
When a voter casts a ballot, they generate a time-lock puzzle encoding their choice. They produce a ciphertext under the election’s public parameters using an exponential ElGamal (or equivalent) scheme. They also supply a zero-knowledge proof that their puzzle encodes a valid vote (for example either 0 or 1) so that they cannot inflate their vote or encode invalid values.

During the voting period the tally remains hidden: each time-lock puzzle is homomorphically added into a running encrypted tally without revealing interim results. Because the puzzles require

sequential computation T steps to decrypt, no party learns the intermediate tally before the deadline. This protects against running-tally leakage and strategy manipulation.

Once voting closes and the time parameter T has elapsed, any party can solve the accumulated puzzle and decrypt the final tally. The result can then be published and audited against the onchain commitments. Because the system uses homomorphic operations, only one decryption is needed for the sum rather than decrypting each ballot individually.

In this way, Cicada offers ballot privacy (depending on integration), running-tally privacy, and minimal trust assumptions: there is no need for dedicated decryptors or trusted authorities. The voter proofs ensure validity, and the time-lock puzzle ensures confidentiality until the end of voting.



Analysis

1. **`Correct Execution`** - Yes. Cicada is designed so that no one can produce a false tally that would be accepted by the smart contract—provided standard cryptographic assumptions hold. If 1) the cryptographic primitives (time-lock puzzle hardness, ElGamal security, and proof system soundness) remain intact, and 2) the smart contract logic is implemented correctly, then it is mathematically impossible to forge a tally that passes verification.
2. **`Robustness`** - Medium. Cicada is quite robust under partial system malfunction. It can handle missing voters, failed transactions, or invalid proofs without breaking. It

guarantees that if a result appears onchain, that result is correct. However, a real failure mode here is liveness—if no one completes the time-lock computation, the result remains encrypted. But correctness (integrity) is never lost. In conclusion, Cicada is robust in correctness, but not robust in liveness. Ideally, redundancy or incentives (to ensure someone finishes decryption) would make Cicada more robust.

3. **Censorship Resistance & Liveness** - Yes. Cicada inherits its censorship resistance primarily from the blockchain layer it's deployed on (typically Ethereum or another EVM-compatible chain). So unless the entire blockchain is censored, the result will appear. However, a malicious contract deployer could introduce logic that rejects certain voters. This is prevented only if the contract code is open-source and verified onchain (as Cicada's reference contracts are). As noted above, liveness is a real failure mode. No trusted entity can prevent the vote from being tallied, but if no one completes the time-lock computation, then the result remains encrypted.
4. **Coercion Resistance & Receipt Freeness** - No. Cicada provides temporary ballot secrecy through time-lock puzzles—each vote is encrypted until the puzzle is solved after the voting window ends. As a result, 1) No one (including coercers) can see the vote during the election, but 2) the vote content is public and directly linkable to the sender's address after decryption—unless an anonymity layer is added. So by itself, Cicada offers zero coercion resistance and no receipt-freeness once the tally is revealed.
5. **Private Vote** - Partial. Cicada provides temporary privacy of individual ballots, but it does not guarantee permanent unlinkability between a voter and their choice. Its design focuses on hiding the running tally, not on fully anonymizing voters.
6. **Running Tally Privacy** - Yes. Cicada provides temporary privacy of individual ballots—i.e. hiding the running tally.
7. **Quorum Status** - Yes. The onchain Cicada contracts store a `numVotes` variable which is incremented for each submitted ballot. This can be used to infer the quorum status while a vote is in progress.
8. **Individual Verifiability** - Yes. When a voter submits their ballot: the vote is recorded directly onchain as a transaction to the Cicada smart contract, the contract verifies the voter's zero-knowledge proof of a valid vote (0 or 1), and (once accepted), the ballot becomes part of the public onchain state (the set of all valid ciphertexts contributing to the homomorphic tally). As a result, a voter can always look up their transaction hash, verify that it is accepted by the Cicada contract, and confirm that their encrypted ballot (time-lock puzzle) is included in the set used to compute the aggregate tally.
9. **Universal Verifiability** - Yes. After the voting period: all encrypted ballots are public onchain, the smart contract maintains the aggregated homomorphic ciphertext (the "sum" of all encrypted votes), and anyone can recompute the aggregation independently—the result must match the onchain tally ciphertext. Then, when the final tally is revealed: the solver provides a proof of correct decryption, and this proof is verified onchain by the smart contract. If the proof passes, it's cryptographically guaranteed that the revealed tally equals the decryption of the homomorphic sum of all valid ballots. As a

result, anyone (a voter, auditor, or observer) can verify that the final tally equals the correct decryption of all accepted ballots. There's no trusted tallying authority, no hidden computations, and no private keys required for verification.

10. **Eligibility** - Yes. Cicada's cryptography enforces vote validity by requiring each submitted ballot to include a valid zero-knowledge proof that the vote is either 0 or 1. However, eligibility (who can vote) depends on how the smart contract integrates with an external identity or membership system. Cicada deliberately leaves eligibility open-ended for implementers to define.
11. **Uniqueness** - Yes. Cicada ensures uniqueness through onchain and cryptographic safeguards. At the contract level, every ballot submission is verified for validity and recorded against the sender's address, so any attempt to vote again is automatically rejected. In anonymous configurations using zero-knowledge membership systems like Semaphore, uniqueness is preserved via nullifier hashes, which allow one vote per verified identity without revealing who voted. These mechanisms make double voting or ballot duplication cryptographically impossible.
12. **Multiple Choice, Fractional Voting** - No. Cicada is implemented to support first-past-the-post (FPTP) binary (yes/no) voting, each ballot can be 0 or 1. However, the underlying cryptography is more flexible, so Cicada can be extended to support more complex voting schemes. Section 3 of the Cicada paper demonstrates how to adapt the protocol to support approval, range, cumulative or even quadratic voting. These extensions require modifications which are not yet supported.
13. **Weighted Voting** - No. As mentioned, Cicada's design focuses on binary, equal-weight ballots. However, the underlying protocol could be extended to weighted or even nonlinear voting systems with some modification.
14. **Vote Updatability** - No. Votes are not updatable once submitted—a ballot is final when it's accepted by the smart contract. The protocol enforces a strict append-only, single-vote rule to guarantee tally integrity and prevent double voting.
15. **Vote Delegation** - Possible. Cicada does not currently support delegated voting. However, a custom vote delegation system could be layered on top of Cicada through smart-contract or cryptographic extensions.
16. **Delegation Privacy** - NA. Depends on custom vote delegation logic.
17. **Engagement Disclosure** - NA. Depends on custom vote delegation logic.
18. **Vote Disclosure** - NA. Depends on custom vote delegation logic.
19. **Delegation Updatability** - NA. Depends on custom vote delegation logic.
20. **Implementation Simplicity** - Low. Cicada relies on time-lock cryptography (linearly homomorphic time-lock puzzles (HTLPs)), custom solidity libraries, zero-knowledge proofs, and the actual running of votes requires the set up of offchain infrastructure to solve the timelock puzzles.
21. **Ease of Use** - Medium. Cicada is easy for wallet-native users. The UX is comparable to other onchain votes. However, the cost of submitting a ballot (~400–450k gas) on Ethereum L1 can be expensive during busy periods.

22. **Gas Efficiency** - Medium-High. Submitting a binary vote in the reference implementation costs approximately 418k gas. More complex ballots significantly increase the cost—a cumulative vote with two candidates would cost 3.39M gas to submit, with the gas cost scaling linearly with the number of candidates. Finalization (verifying the time-lock solution) costs approximately 101k gas.
23. **Ease of Integration/Deployment** - Medium. Cicada composes well with common building blocks (token/NFT gating, Merkle allowlists, zero-knowledge-membership like Semaphore, L1/L2 deployments). But there's no turnkey plugin for popular governance frameworks (e.g., OpenZeppelin Governor), so DAOs need to do a custom integration. Cicada contracts and reference flow are public and production-minded. However, a DAO would need to wire up eligibility, host a simple client to build ballots, choose parameters, and decide who (or how) the final timelock computation is run.
24. **Implementation Maturity** - Low. Cicada is research-backed and implementation-ready for pilots, but not a turnkey, audited, production framework yet.

Enclave (CRISP)

Overview

Enclave proposes Encrypted Execution Environments (E3)s that combine fully homomorphic encryption (FHE), zero-knowledge proofs, and distributed threshold cryptography. CRISP (Coercion-Resistant Impartial Selection Protocol) is the name of the voting protocol built on top of Enclave's E3 infrastructure. In a voting flow, voters register a keypair, encrypt votes under a threshold FHE key and provide zero-knowledge proofs that votes are valid. The "compute provider" homomorphically tallies encrypted votes, and key holders decrypt only the final result. The protocol supports re-voting and key switching.

- Website: https://www.enclave.gg/
- GitHub: https://github.com/gnosisguild/enclave
- Ethresearch: https://ethresear.ch/t/collusion-resistant-impartial-selection-protocol-crisp/18804
- Testnet: No
- Mainnet: No

Explanation

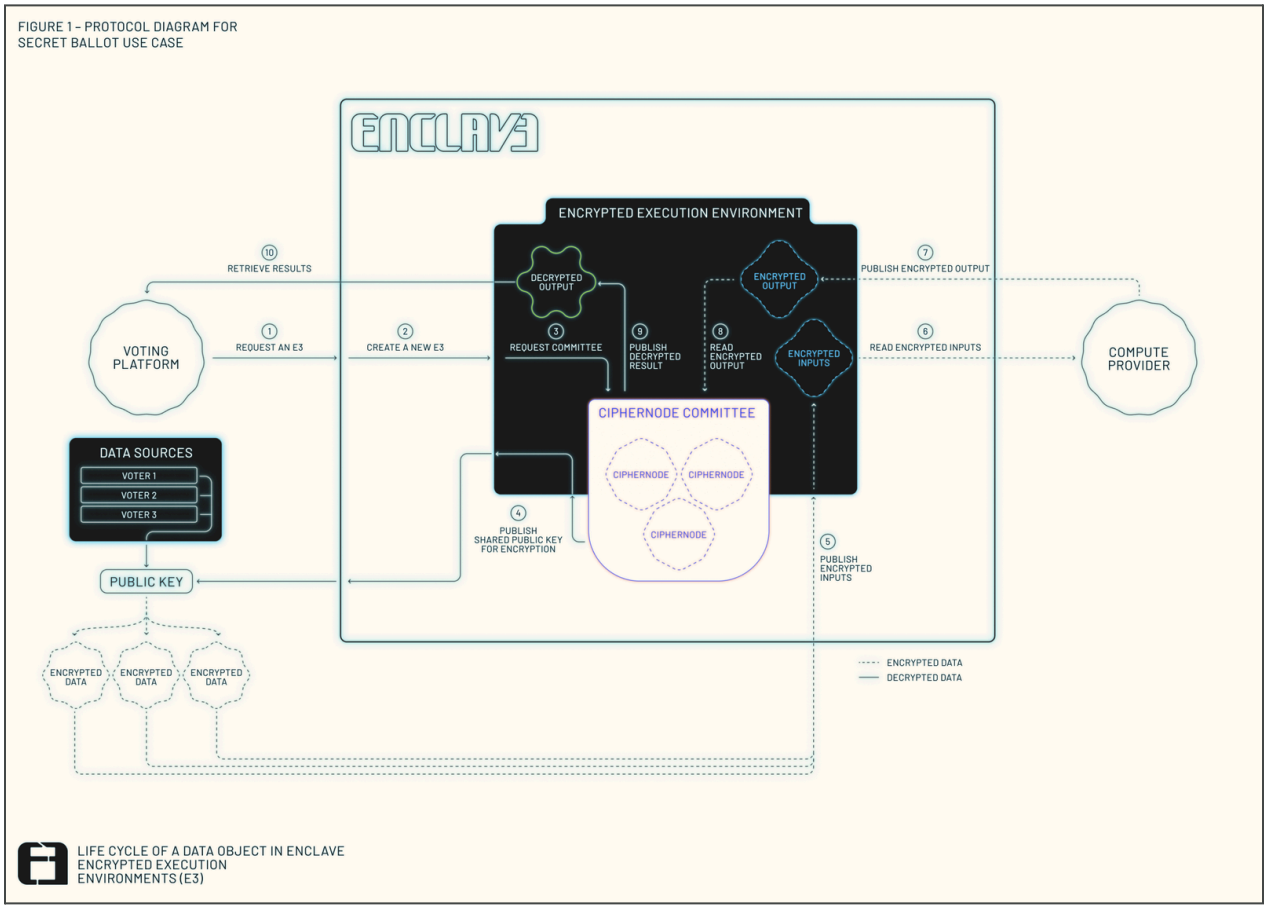
Enclave is an open-source protocol designed for Encrypted Execution Environments (E3), allowing secure and confidential computations to take place without ever exposing sensitive data. One of its flagship applications is CRISP—the Coercion-Resistant Impartial Selection Protocol—a private voting system that combines fully homomorphic encryption (FHE), Distributed Threshold Cryptography (DTC), and zero-knowledge proofs to achieve end-to-end privacy and verifiable results.

A voting round begins when an operator requests an E3 instance from the Enclave network. During initialization, the operator defines parameters such as the voting duration, eligibility requirements, and the set of Ciphernodes that will later participate in decryption. To prepare for the round, these Ciphernodes generate a threshold keypair through Distributed Key Generation (DKG), ensuring that no single node ever holds the complete private key. Once this process is complete, they publish the public key for voters to use when encrypting their ballots.

In the registration phase, voters join anonymously using unique, round-specific cryptographic keys, and their eligibility is verified by the Voter Registry. When submitting votes, participants encrypt their choices and send them to the Enclave system. The protocol allows voters to change

their keys or resubmit ballots at any time, ensuring receipt-freeness and protecting against coercion.

Vote tallying is performed by independent Compute Providers, who process the encrypted data without decrypting it. Zero-knowledge proofs are used to confirm the correctness of computation throughout. Once tallying is complete, the Ciphernodes collectively decrypt the aggregate result and publish the final outcome. Individual votes remain encrypted at all times, guaranteeing both voter privacy and public verifiability of the final tally.



Analysis

1. **`Correct Execution`** - Yes. The combination of (fully homomorphic encryption) FHE, distributed threshold cryptography, and zero-knowledge proofs mean that no single part can falsify the tally. In Enclave's CRISP reference program, voters also submit a zero-knowledge proof along with their vote to prove it was formed correctly.
2. **`Robustness`** - High. Enclave relies on an honest majority assumption where a proportion of the network's nodes need to collude in order to threaten the assumptions of the protocol. Ciphernodes are economically staked, and they are slashed if they misbehave.

3. **Censorship Resistance & Liveness** - Yes. It is possible to submit messages directly to the smart contracts or via an alternative relaying service to avoid vote censorship. The same principle applies for sign up. Because there exists a set of Ciphernodes, if the set is sufficiently decentralized, no single entity can stop the tally.
4. **Coercion Resistance** - Yes. Voters can override their vote at any time by submitting an updated ciphertext alongside a zero-knowledge proof of validity. The vote is encrypted to the public key provided by Ciphernodes, and only the last valid ciphertext for each voter is counted. To prevent users from proving how they voted, anyone is able to perform a masking operation on any user's storage slot holding their vote ciphertext. The masking operation is simply an addition of a zero ciphertext to the original vote, packaged with a zero-knowledge proof that does not disclose the action performed—vote casting, overriding or masking. Thus, it will be impossible for a voter to prove how they voted provided that enough masking operations are performed by a healthy network of nodes, voters and/or entities that want to ensure fairer governance. Therefore, a briber has no way of knowing if their bribe was effective.
5. **Private Vote** - Yes. Votes are encrypted to a shared key provided by the Ciphernodes. They are never decrypted but they are summed with other votes to reach the final tally, which is then decrypted.
6. **Running Tally Privacy** - Yes. Once the vote has finished, the Compute Provider computes the encrypted tally and produces a proof of correct execution. A threshold of Ciphernodes then decrypts the final ciphertext.
7. **Quorum Status** - No. Votes and tallies are private, and the final tally is only revealed at the end of the vote by the Compute Provider, which is then decrypted by Ciphernodes. The running tally cannot be inferred to determine quorum status. The output is the raw tally for each option. However a future version will make this a configurable option. Whereby the quorum is defined and the output is simply the outcome, rather than the tallies for each option.
8. **Individual Verifiability** - Yes. Votes are recorded onchain when submitted, each vote is accompanied with a zero-knowledge proof that proves the ciphertext is a correctly formed vote signed by a registered voter. A voter can verify their message is included in the smart contract.
9. **Universal Verifiability** - Yes. After the voting period, the Compute Provider will compute the tally ciphertext, and Ciphernodes will decrypt the output and post the hash of plaintext results onchain. The Compute Provider runs a fully homomorphic encryption (FHE) computation and provides a zero-knowledge proof of correct computation that can be verified
10. **Eligibility** - Yes. Each vote is accompanied with a zero-knowledge proof that the vote is correctly formed and comes from a registered user
11. **Uniqueness** - Yes. Each registered voter has a unique, single-use key pair for a poll, as implemented in CRISP. Each eligible voter can only submit a vote to their assigned storage slot by proving their eligibility through a zero-knowledge proof and their ECDSA key. The

slot is only overridable, thus the value held in the slot at the end of the voting period will be counted in. The ciphertext is guaranteed to be a valid vote from the eligible voter, or the result of a masking operation. This is guaranteed by the zero-knowledge proof submitted with each ciphertext. Only the latest message for a particular key is counted.

12. **Multiple Choice, Fractional Voting** - Yes. Votes are an encryption of polynomials with a large number of coefficients. This means that there is lots of space to implement many types of votes including multiple choice, fractional voting. In order to implement these voting methods, the zero-knowledge proof that is used to prove that the vote was formed correctly needs to be adjusted. For example, in a quadratic voting scenario, the zero-knowledge proof should verify that a voter can square their assigned votes, and that value should still be within their total voting power assigned.
13. **Weighted Voting** - Yes. Each registered user has a vote weight that is defined by CRISP's Voter Registry smart contract. As mentioned in the above property description, a wide variety of vote types can be supported, including QV.
14. **Vote Updatability** - Yes. Voters can change their vote at any time prior to the deadline by submitting another encrypted vote. Only the latest message is counted.
15. **Vote Delegation** - Possible. CRISP protocol doesn't hard-code delegation, but it's built so a vote delegation can be inherited from the voting application with which it is integrated.
16. **Delegation Privacy** - NA. Depends on vote delegation logic.
17. **Engagement Disclosure** - NA. Depends on vote delegation logic.
18. **Vote Disclosure** - NA. Depends on vote delegation logic.
19. **Delegation Updatability** - NA. Depends on vote delegation logic.
20. **Implementation Simplicity** - Low. Enclave's architecture involves several components and uses a mix of fully homomorphic encryption (FHE), distributed threshold cryptography, and zero-knowledge proofs. A full-stack implementation using CRISP needs a frontend client, a backend coordination server, computation program, smart contracts, circuits, and deployment scripts. This complexity can be abstracted away by Enclave's CLI and SDK, which handle most orchestration steps for developers.
21. **Ease of Use** - Medium. Enclave's Compute Providers and Ciphernodes can handle the majority of tasks required for CRISP to function. Vote organisers will need to request an E3 from the network, and then define parameters such as voting duration, eligibility criteria, and the Ciphernodes required for decryption. The team is also working on integrations into tooling providers to make CRISP private voting more accessible to DAOs
22. **Gas Efficiency** - Medium-High. Good benchmarks for Enclave do not yet exist. Chains will interact with the voting protocol by calling a contract implementing the E3 program interface, which is connected to the Enclave network. The two main function calls here are `verify()` and `validate()`. Because of the heavy cryptography involved, it will be potentially quite gas heavy.

23. **Ease of Integration/Deployment** - Medium. Enclave provides documentation and a full-stack example for using CRISP. CRISP is a complete example of an E3 Program, so there is no need to write your own E3 program.
24. **Implementation Maturity** - Low. The Enclave E3 infrastructure that CRISP will use has not gone to production yet so has not reached maturity. Enclave is currently in devnet and testnet phases. The team is targeting Q1 2026 for a mainnet release.

Kite

Overview

Kite is proposed in a recent paper and introduces a protocol for private delegation. Voters can delegate, revoke and re-delegate their weight without revealing who they delegated to (even the delegate does not learn who delegated). Only the fact that a delegation occurred is public. The protocol supports both public and private voting by delegates and is implemented on Ethereum contracts. It is focussed on delegation privacy, but it also addresses private voting—the system is customisable and you can choose any of the following four configurations: public/private voting + public/private delegation.

- Paper: https://arxiv.org/pdf/2501.05626
- Testnet: No
- Mainnet: No

Explanation

Kite is a protocol designed to enable private delegation of voting power in DAOs. A voter begins by holding voting power (e.g., via tokens) and, when they wish, they can delegate that power to a delegate. Importantly, the protocol keeps the identity of who delegated to whom hidden.

When a voter chooses to delegate, they submit a delegation operation onchain that publicly records that a delegation occurred, but not the identity of the delegate or the voter. The protocol employs zero-knowledge proofs so that the act of delegation is verifiably correct without revealing the linkage. The delegate themselves does not learn who delegated to them.

If the voter later wishes, they can revoke or re-delegate their voting power to someone else. This is done via the same private delegation mechanism: the onchain record notes a revocation or re-delegation event, again without disclosing the parties involved. Throughout, the voter retains the freedom to change their delegate.

When a delegate votes on behalf of their aggregated voting power, Kite supports either public voting (where the delegate's vote is visible) or private voting (where it is hidden). The system is implemented as an extension of the widely-used Governor Bravo smart contract on Ethereum, so it integrates with standard DAO governance workflows.

By combining private delegation with verifiable voting, Kite enables voters to delegate without compromising privacy or introducing coercion risk, while still ensuring the governance process remains auditable and secure.

Analysis

1. **Correct Execution** - Yes. Kite specifically ensures correct execution of vote tallies. This integrity guarantee rests on a few cryptographic mechanisms, inc: homomorphic encryption of vote weights, and zero-knowledge proofs of delegation and vote submissions. The only residual trust is in the tally authority's honesty during decryption—it cannot forge results (because all ciphertexts are public and verifiable), but it could refuse to publish the final decrypted total. That risk can be mitigated by replacing the Trusted Authority with a threshold or MPC decryption scheme, which the authors note as a future improvement.
2. **Robustness** - Medium. Kite is robust against most participant or network failures—invalid votes, dropped users, or reordered transactions can't corrupt results thanks to onchain verification and zero-knowledge proofs—but the existing implementation relies on a single Trusted Authority to decrypt the final tally. If that authority fails or acts maliciously, the vote can't be completed (though tampering would be detectable). However, it must be noted that the trusted authority is not strictly enforced in the design, it can be implemented as a committee with honest majority assumptions. This issue is addressed further in an upcoming update to the paper, and is explicitly acknowledged in the current edition that this entity can be decentralized. The role is relatively simple, the only thing it does is it decrypts the tally and posts the corresponding correctness proof. Threshold decryption can be used, and each party can generate a proof for their share. In short, Kite guarantees correctness under normal and partial failures but a decentralized tallying authority is needed to increase robustness, so under this assumption, is not fault-tolerant to authority compromise, making it cryptographically strong yet operationally fragile.
3. **Censorship Resistance & Liveness** - No. As noted above, the Trusted Authority could be offline or refuse to publish the final decrypted total. That risk can be mitigated by replacing the Trusted Authority with a threshold or MPC decryption scheme, which the authors note as a future improvement.
4. **Coercion Resistance** - No. Kite does not provide coercion resistance or receipt-freeness. Kite improves over typical DAO systems by making vote delegation private. But a coercer can demand the voter prove that they delegated to a specific address (by revealing encryption randomness or pre-image). Kite also lacks fake-receipt mechanisms that would let voters fool a coercer.
5. **Private Vote** - Yes. Kite focuses on private delegation, not private voting. The protocol's primary goal is to make delegation relationships private, while still enabling delegates to cast votes (either publicly or privately) on behalf of others. That said, this feature is enabled in private voting mode.
6. **Running Tally Privacy** - Yes. Kite focuses on private delegation, not private voting. The protocol's primary goal is to make delegation relationships private, while still enabling

delegates to cast votes (either publicly or privately) on behalf of others. Kite has a private voting mode, where the running tally can be kept private.

7. **Quorum Status** - No. Not by default, but this can be enabled if the tally committee periodically decrypts the tally.
8. **Individual Verifiability** - Yes. If the delegate votes are public or private, a voter can confirm their own delegation was accepted since delegations are binding and verifiable onchain, this gives the voter individual verifiability. If delegate votes are private: the voter can still confirm that their delegation was included, but cannot learn the delegate's actual choice. In any case, both delegate voting modes provide inclusion assurance.
9. **Universal Verifiability** - Yes. Kite ensures strong universal verifiability: all encrypted delegations, delegate votes, and zero-knowledge proofs are published onchain, allowing anyone to verify that each submission is valid, counted once, and correctly aggregated. The final tally can be recomputed from these public ciphertexts and checked against the Trusted Authority's decrypted result, making any manipulation or false tally immediately detectable.
10. **Eligibility** - Yes. Kite ensures voter eligibility by linking voting power to a snapshot of token holdings stored in a Merkle tree. When delegating, a voter provides a zero-knowledge proof that they appear in this tree with a valid token balance and haven't already delegated. The smart contract verifies this proof onchain, allowing only legitimate token holders from the snapshot to delegate—without revealing their identity or balance.
11. **Uniqueness** - Yes. The onchain contract then performs several checks: it verifies that the delegate is active, confirms that they haven't voted previously, and validates the Merkle proof. Only after successfully passing these checks does the contract homomorphically add the delegate's encrypted voting power to the tally for the selected option.
12. **Multiple Choice, Fractional Voting** - NA. Kite focuses on private delegation, not private voting. The protocol's primary goal is to make delegation relationships private, while still enabling delegates to cast votes (either publicly or privately) on behalf of others. The delegate voting module may or may not support delegate multiple choice or fractional voting.
13. **Weighted Voting** - NA. Kite focuses on private delegation, not private voting. The protocol's primary goal is to make delegation relationships private, while still enabling delegates to cast votes (either publicly or privately) on behalf of others. The delegate voting module may or may not support delegate weighted voting.
14. **Vote Updatability** - NA. Kite focuses on private delegation, not private voting. The protocol's primary goal is to make delegation relationships private, while still enabling delegates to cast votes (either publicly or privately) on behalf of others. The delegate voting module may or may not support delegate vote updatability. It would be straightforward to implement vote updateability. The voter could simply post a new ballot, which could trigger subtracting the old ballot from the tally and adding the new one

15. **Delegation Functionality** - Yes. Voters can freely delegate, revoke, and re-delegate their voting power without publicly revealing any information about who they delegated to. Even the delegate does not learn who delegated to them. The only information that is recorded publicly is that the voter delegated or re-delegated their vote to someone.
16. **Delegation Privacy** - Yes. Voters can freely delegate, revoke, and re-delegate their voting power without publicly revealing any information about who they delegated to. Even the delegate does not learn who delegated to them. The only information that is recorded publicly is that the voter delegated or re-delegated their vote to someone. However, a coercer can demand the voter prove that they delegated to a specific address (by revealing encryption randomness or pre-image). Kite also lacks fake-receipt mechanisms that would let voters fool a coercer.
17. **Engagement Disclosure** - Yes. Delegate engagement disclosure is enabled by default as the fact that someone voted is always public, regardless of the voting mode.
18. **Vote Disclosure** - No. Kite does not include any mechanism that forces or even enables delegates to selectively disclose how they voted only to their delegators. The design goal of Kite is to keep delegation relationships completely private, so that delegates don't know who delegated to them.
19. **Delegation Updatability** - No. Voters cannot re-delegate after the voting phase has begun and have it take effect. Kite follows a two-phase model: Delegation phase—voters privately delegate, revoke, or re-delegate their voting power. Voting phase—once voting starts, the system takes a snapshot of all valid delegations to "freeze" the distribution of voting power. The nuance is that voters can re-delegate mid-vote, but it will not apply for an ongoing election.
20. **Implementation Simplicity** - Low. Kite's implementation is highly complex because it combines homomorphic encryption, Merkle-tree commitments, and zero-knowledge proofs within a modified Governor Bravo smart contract. It demands careful cryptographic engineering, secure key management for the Trusted Authority, and gas-efficient onchain proof verification. Generating and validating proofs is computationally heavy, and building a user-friendly interface for voters adds further difficulty, making Kite feasible as a prototype but challenging for large-scale DAO deployment.
21. **Ease of Use** - Low. Kite's ease of use for end users is low in its current form. Voters must locally generate zero-knowledge proofs, manage cryptographic keys, and interact with smart contracts—tasks that are far beyond the comfort zone of most DAO participants. There's no native wallet or interface integration yet, so users rely on command-line tools or custom scripts to delegate or revoke votes. Furthermore, proof generation can take minutes on a standard laptop.
22. **Gas Efficiency** - Low. Casting a vote in Kite requires submitting a zero-knowledge proof that the vote was properly formed. The authors' tests on using Anvil reported the following averages for verifying proofs: 406,646 gas with a median of 396,323 gas. Gas optimisation was not a major focus of the research so there would be scope to bring this down in future implementations. The benchmarks for delegation were done with a relatively small delegation set of between 5 and 20 (delegate sizes in real-world

democracies range in the hundreds). This smaller anonymity set also means higher gas prices must be absorbed for more privacy.

23. **Ease of Integration/Deployment** - Medium. Kite is built to extend Ethereum's Governor Bravo, so technically it can plug into existing DAO frameworks, but doing so requires modifying core governance contracts, adding cryptographic verification logic, and deploying new infrastructure for encrypted ballots and zero-knowledge proof validation.
24. **Implementation Maturity** - Low. Kite's implementation maturity is prototype-level, not production-ready. The authors built a functional proof of concept integrated with Ethereum's Governor Bravo using the Noir zero-knowledge framework to demonstrate feasibility and measure performance. While the cryptographic design is sound and formally analyzed, the implementation lacks large-scale testing, formal audits, and usability optimization.

Shutter - Permanent Shielded Voting

Overview

This version extends Shutter's threshold encryption to enable permanent privacy via threshold ElGamal encryption. Votes remain encrypted even after the voting period, only aggregated results are revealed using homomorphic tallying. This PoC aims to bring fully private, verifiable, and auditable governance to DAOs. Currently, it's in the proof-of-concept stage.

Stage: PoC + Snapshot partnership (research and testing)
Integrations: Snapshot PoC and planned for onchain use and future governance platforms

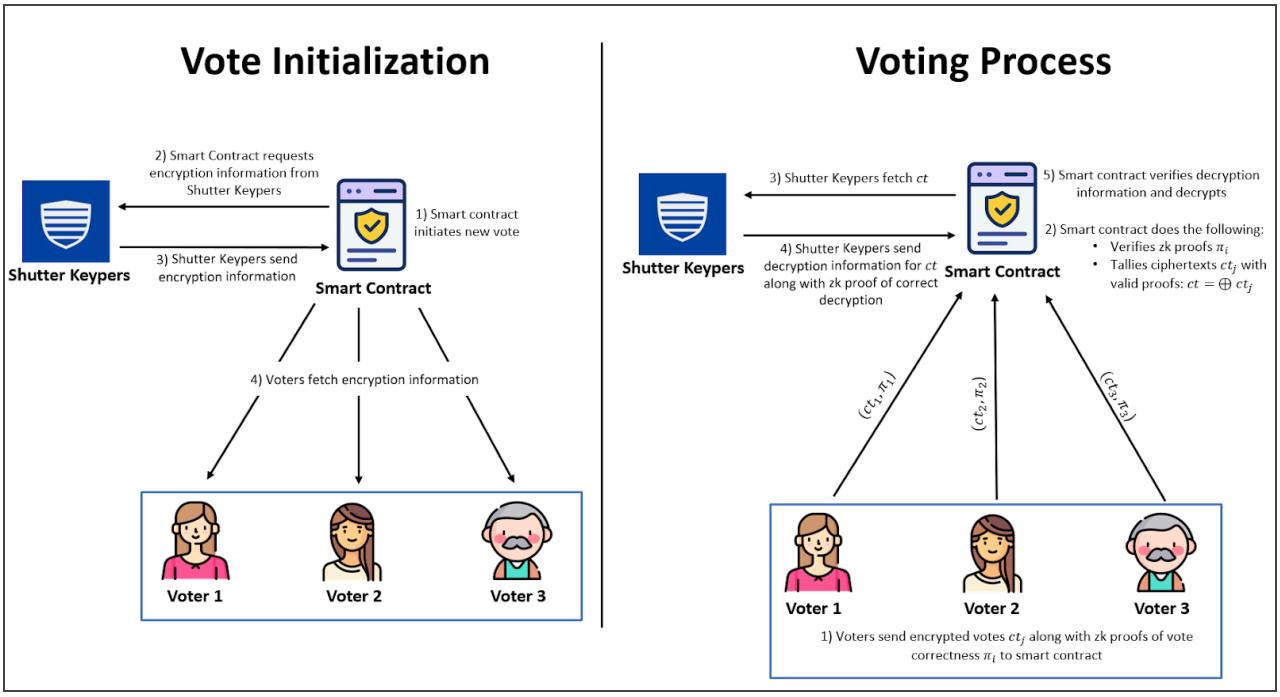
- Website: https://www.shutter.network/shielded-voting
- PoC and Explainer: https://blog.shutter.network/coming-soon-to-daos-permanent-shielded-voting-via-homomorphic-encryption/
- Snapshot PoC: https://blog.shutter.network/permanent-shielded-voting-is-coming-to-snapshot/

Explanation

Shutter Permanent Shielded Voting extends Shutter's threshold encryption to provide permanent ballot privacy. Instead of decrypting individual votes at the end of a proposal, it uses threshold ElGamal encryption and homomorphic tallying so that only the final aggregated result is ever revealed.

When a proposal is created, a decentralized committee of Keypers runs a Distributed Key Generation (DKG) to produce a shared encryption key. Voters encrypt their ballots with this public key and submit them to the governance platform (for example, Snapshot or an onchain Governor). Once the voting period ends, the encrypted votes are aggregated homomorphically, and the Keypers jointly decrypt only the final tally. Individual ballots remain permanently confidential.

All ciphertexts, aggregation steps, and decryption shares are public, enabling anyone to verify that the published tally matches all valid encrypted votes. A proof of concept demonstrating this system on Snapshot is live, and further integrations with on- and offchain governance frameworks are under active development.



Analysis

1. **`Correct Execution`** - Yes. Each vote is encrypted under threshold ElGamal, and tallies are computed homomorphically. Verifiable decryption proofs allow anyone to confirm the final tally matches all valid encrypted votes without revealing them.
2. **`Robustness`** - High. Liveness depends only on a threshold of Keypers being active. Since correctness is cryptographically guaranteed, malicious Keypers cannot manipulate outcomes. Recovery procedures can rotate keys if a subset fails.
3. **`Censorship Resistance & Liveness`** - Yes. Threshold operation ensures no single Keyper or entity can censor, decrypt, or stall voting unilaterally.
4. **`Coercion Resistance`** - No. Votes remain permanently secret, but there are still no mechanisms preventing voters from producing receipts or proving their choices. Future improvements could add cryptographic receipt-freeness.
5. **`Private Vote`** - Yes. Votes remain encrypted indefinitely; only aggregated results are revealed.
6. **`Running Tally Privacy`** - Yes. The tally is computed from encrypted votes without decrypting individual ballots.
7. **`Quorum Status`** - Yes. Although the contents of the votes are encrypted during the voting period, the identities of the voters and the voting power deployed are both visible throughout the voting period. As a result, the quorum status is visible throughout the voting period.

8. **Individual Verifiability** - Yes. Each voter can verify that their encrypted ballot was included in the final computation without revealing its content.
9. **Universal Verifiability** - Yes. Anyone can verify, using public proofs, that the published tally correctly corresponds to all valid encrypted votes.
10. **Eligibility** - Yes. Shutter relies on the host governance system (e.g. Snapshot) to enforce which addresses are allowed to vote.
11. **Uniqueness** - Yes. Each eligible voter can submit only one encrypted vote per proposal, ensuring fair counting.
12. **Multi Choice, Fractional Voting** - Yes. The protocol supports multiple choice voting, fractional voting and other weighted voting schemes used in DAOs.
13. **Weighted Voting** - Yes. The protocol supports quadratic voting and other weighted voting schemes used in DAOs.
14. **Vote Updatability** - Yes. Voters can submit updated encrypted ballots within the voting window.
15. **Vote Delegation** - Yes. Shutter relies on the host governance system (e.g. Snapshot) for delegation functionality.
16. **Delegation Privacy** - Possible. Delegation relationships (who delegates to whom) are visible and can be proven via the state onchain (ERC-20 delegate). However, delegation privacy could be enabled with additional encryption layers.
17. **Engagement Disclosure** - Yes. Although the contents of the votes are encrypted during the voting period, the identities of the voters and the voting power deployed are both visible throughout the voting period. As a result, delegate engagement (or lack thereof) is publicly visible.
18. **Vote Disclosure** - Possible. Delegate votes could be revealed to or hidden from select voters, depending on the host governance system design.
19. **Delegate Updatability** - Possible. Works with any delegation mechanism, including those that support re-delegation mid-vote. Shutter Permanent Shielded Voting will be integrated into Snapshot, which supports re-delegation mid-vote.
20. **Implementation Simplicity** - Medium-High. Shutter Permanent Shielded Voting introduces additional cryptographic components for homomorphic tallying and threshold decryption, which increases complexity compared to temporary Shielded Voting. This makes the protocol moderately complex from a technical standpoint, but still simpler than full (fully homomorphic encryption) FHE or more complex MPC-based voting systems.
21. **Ease of Use** - High. User experience expected to remain similar, with encryption handled automatically.
22. **Gas Efficiency** - Medium. The purpose built, linearly homomorphic encryption is orders of magnitude more scalable than fully homomorphic encryption (FHE), but there's still some compute needed. No issue for offchain applications and should also work comfortably onchain with some optimizations.
23. **Ease of Integration/Deployment** - Medium. Requires custom cryptographic handling and deeper integration until SDKs are available.

24. **Implementation Maturity** - Low. Research-stage PoC only; not ready for production

Table of Project Evaluations

	Freedom Tool	MACI V3	Semaphore V4	Shutter - Shielded Voting	SIV	Incendia	DAVINCI	Aragon/Aztec	Cicada	Enclave	Kite	Shutter - Permanent Shielded Voting
Correct Execution	Yes	Yes	Partial	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Robustness	High	Medium	High	Medium-High	Medium	High	High	Medium	Medium	High	Medium	High
Censorship Resistance & Liveness	Yes	Partial	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No	Yes
Coercion Resistance	No	Yes	No	No	Yes	Partial	Yes	No	No	Yes	No	No
Private Vote	Yes	Yes	Partial	Partial	Yes	Partial	Yes	Yes	Partial	Yes	Yes	Yes
Running Tally Privacy	No	Yes	No	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
Quorum Status	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes
Individual Verifiability	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Universal Verifiability	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Eligibility	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Uniqueness	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Multiple Choice, Fractional Voting	No	Yes	Possible	Yes	Yes	Yes	Yes	No	No	Yes	NA	Yes
Weighted Voting	No	Yes	Possible	Yes	Yes	Yes	Yes	No	No	Yes	NA	Yes
Vote Updatability	No	Yes	Possible	Yes	Yes	Yes	Yes	No	No	Yes	NA	Yes
Vote Delegation	No	No	Possible	Yes	Possible	Yes	Possible	Yes	Possible	Possible	Yes	Yes
Delegation Privacy	No	No	NA	No	Yes	Yes	NA	No	NA	NA	Yes	Possible
Engagement Disclosure	No	No	NA	Yes	Yes	Yes	NA	No	NA	NA	Yes	Yes
Vote Disclosure	No	No	NA	No	No	No	NA	No	NA	NA	No	Possible
Delegation Updatability	No	No	NA	No	Yes	No	NA	No	NA	NA	No	Possible
Implementation Simplicity	Medium-High	Low	High	Medium	Medium	Medium	Low	Low	Low	Low	Low	Medium-High
Ease of Use	High	Medium-High	Medium-High	High	High	Medium	Medium	Medium	Medium	Medium	Low	High
Gas Efficiency	Medium-High	Medium	Medium	High	N/A	High	High	Low	Medium-High	Medium-High	Low	Medium
Ease of Integration/Deployment	High	Low	High	High	Low	Medium	Medium-High	Low	Medium	Medium	Medium	Medium
Implementation Maturity	High	High	High	High	High	Medium	Medium-Low	Low	Low	Low	Low	Low

Recommendations

Each of the private voting protocols we analyzed has its own benefits and trade-offs. In this section, we describe the protocols we think are the best to address specific use-cases. Each private voting protocol has unique benefits and trade-offs that should be considered for specific use-cases.

Best Protocols for Implementation Maturity

The following list is filtered by high evaluations of the “Implementation Maturity” property, ordered alphabetically.

- **Freedom Tool** - Freedom Tool has already been battle tested in multiple elections linked to country-specific topics (e.g. Russia 2024 parallel election to promote democracy, freedom and peace).

- **MACI V3** - MACI is currently on the 3rd iteration of the protocol. MACI has been used in several funding rounds (e.g. Gitcoin rounds, RetroPGF (Retroactive public goods funding).

- **Semaphore V4** - Semaphore can be considered a battle-tested tool because it is widely used in the Ethereum ecosystem and the new zkApps landscape. It is one of the most used ZK protocols on Ethereum and is on its 4th iteration.

- **Shutter - Shielded Voting** - Snapshot (a DAO tooling provider) has allowed any of its DAOs to use Shutter shielded voting since 2022. It has been used by over 850 DAOs thus far, with over 5500 proposals shielded, and over 350,000 votes encrypted.

- **SIV** - is a private voting application used for real life elections (polls). SIV has been used in multiple real life local elections in the US. SIV is being used for the largest digital vote for a government official in US history—to replace a State Senator in Utah, with an expected 70k eligible voters.

Best Protocols for Real-World Elections

- **Freedom Tool** - Freedom Tool has been used multiple times for real-world elections, and has an app which provides users with an option to vote on their phones

- **DAVINCI** - The Vocdoni app and also the DAVINCI protocol have been designed to be used for real-world elections. The team has run several successful real-world elections and votes in the past.

- **SIV** - Another project specifically designed around real-world elections. SIV works completely offchain so has no associated gas costs, and has been used multiple times in US-based elections.

Best Protocols for Censorship Resistance & Liveness

The following list is filtered by high evaluations of the “Censorship Resistance & Liveness” property ordered alphabetically.

- **Freedom Tool** - The application consists of smart contracts with zero-knowledge registries using the Rarimo protocol and a user-facing application. Freedom Tool does not encrypt the votes so anyone could tally them without the need of a trusted party. If the user-facing application went down, voters will be able to submit their votes directly onchain.

- **Semaphore V4** - A simple private voting protocol will have a set of smart contracts (with zero-knowledge proof verifiers) and a user-facing application. In case the application went down or was censoring a specific user, the user could go and submit a vote directly onchain.

- **Shutter - Shielded Voting** - The protocol uses smart contracts, a user-facing application and the MPC Keypers network. If the application went down or was censoring users, they could go and submit their votes directly to the smart contracts. The Keypers network ran a distributed key generation process to publish the public encryption key that voters will use to encrypt their choice. In the votes decryption step, a set of Keypers can go offline (as long as the online nodes are above the threshold level) and the network would still be able to successfully decrypt the votes and finish the tallying process. In practice, it is improbable that the online honest Keypers go below the threshold.

- **Incendia** - The protocol uses smart contracts (with zero-knowledge proof verifiers) and a user-facing application. If the application went down or was censoring anyone, voters will be able to generate their private proof-of-burn locally and submit their vote directly onchain.

- **DAVINCI** - The protocol uses smart contracts (with zero-knowledge proof verifiers), a sequencer network (that acts similar to an L2) and a user-facing application. If the

application went down or was censoring anyone, voters will be able to interact directly with the smart contracts in order to submit a vote. The decentralized sequencers are in charge of the distributed key generation process for the public encryption key and of proof verification. All data is stored in Ethereum blobs under EIP-4844. In the vote decryption step, a threshold majority of sequencers is needed in order to process the votes and publish the final tally. If the number of honest nodes is less than the threshold, then the tally process could be halted. If sequencers go offline, users would be able to reconstruct past election states by reprocessing the blob-stored data.

- **Aragon / Aztec** - The protocol uses smart contracts (with zero-knowledge proof verifiers), a timelock randomness provider and a user-facing CLI. The CLI generates and submits votes locally so there are no risks of censorship or network down times. The timelock randomness provider could go offline or rogue and halt the randomness required to build the private key that decrypts the votes.

- **Cicada** - The protocol consists of smart contracts (with zero-knowledge proof verifiers) and a homomorphic encryption time-lock puzzle solver. Voters can submit their votes directly onchain. After the poll has been created and the public key has been posted onchain, a solver should start computing to find the private key that derives the public key. This process is computationally intensive and needs to be performed in time so that the final tally is revealed timely after the poll ends.

- **Enclave (CRISP)** - The protocol consists of an Encrypted Execution Environment (E3) instance and a user-facing application. If the application goes down or is censoring anyone, voters can submit their votes directly to the E3 instance at the smart contract level. The E3 encompasses a computation network that performs operations over homomorphic encrypted data and verifies zero-knowledge proofs. The liveness of the protocol depends on an honest majority of nodes larger than the threshold.

- **Shutter - Permanent Shielded Voting** - Censorship resistance is achieved by allowing users to submit votes directly onchain and the liveness is achieved in practice by having a strong set of honest Keypers above the threshold level.

Best Protocols for Coercion Resistance

The following list is filtered by high evaluations of the “Coercion Resistance & Receipt Freeness” property ordered alphabetically.

- **MACI V3** - The key change feature allows for coercion resistance because there is no way to verify a specific vote will be the final tallied vote submitted by any one voter. It is worth pointing out that a malicious coordinator can see this information and can disclose it to a briber.

- **SIV** - The anti-bribery/coercion feature (needs to be activated for specific polls) guarantees coercion resistance. A user can work together with the poll authority to secretly cancel their vote and submit a new one. The briber and anyone outside of these two parties would not know the vote was updated. The poll authority can see this information and can disclose it to a briber.

- **DAVINCI** - The re-encryption, stealth vote overwriting and sequencers' re-randomization features prevent voters from producing a verifiable record of how they voted.

- **Enclave (CRISP)** - The nodes' masking feature adds an encrypted zero to users' votes and totally changes the original vote ciphertext without modifying the underlying value. This prevents users from proving how they voted and a healthy node network will perform enough masking operations on a particular poll. Enclave also supports key changes.

Best Protocols for Gas Efficiency

The following list is filtered by high evaluations of the "Gas cost" property ordered alphabetically.

- **Shutter - Shielded Voting** - Votes are encrypted in the user's local memory and then posted onchain, so transactions are gas-efficient. The vote decryption and tally process happen in the governance host so there is no onchain encrypted data operation.

- **SIV** - SIV votes happen completely offchain, so there is no associated gas cost.

- **Incendia** - Private Proof-of-Burn is generated in the user's local memory after the user has successfully transferred to an unspendable address. The vote submission function execution onchain is gas efficient (consumes approximately 300k gas) and relayers could be implemented in order to submit gasless transactions.

- **DAVINCI** - The cost of executing a state transition containing batches of 60 votes costs approximately 350k gas. This is a more tentative assessment since the production L2 is not live yet.

Best Protocols for Delegation Privacy

The following list is filtered by high evaluations of the "Delegation Privacy" property ordered alphabetically.

- **SIV** - Voters can easily delegate their vote ability to third-parties that would vote on their behalf. IP address leakage could reveal who is the delegate.

- **Incendia** - Voters can transfer their private proof-of-burn to a delegate and the delegate will not learn anything about the voter. The voting process of a delegate will be the same as a normal voter.

- **Kite** - Kite was specially designed with private delegation in mind. It is possible to run the protocol in several modes: private delegation for voters, and private delegation for voters plus private voting for delegates.

Future Work

As exemplified by this report, there are a number of exciting (existing and upcoming) private voting protocols on Ethereum . However, there are a number of opportunities we encourage these projects and others to explore.

1. **Protocol Trust Assumptions** - Most of the private voting protocols analyzed introduce trust assumptions that could compromise parts of the voting process. The trade-offs of such setups should be analyzed according to the needs of a particular DAO. For example, a single trusted tallying authority and even a decentralized network could go offline or influence the voting process negatively (maliciously or not maliciously). An ideal private voting protocol would have the same trust assumptions as a non-private fully onchain voting protocol (trust is placed in smart contracts and cryptography).

2. **Secret Sharing with Snitching** - Adding a snitching mechanism to threshold cryptography schemes reduces the risk of key shareholder collusion by making it provable and subsequently slashable. This ensures that any collusion or invalid behavior during vote aggregation or decryption can be attributed to the specific entities who have misbehaved

3. **Privacy & Delegation** - Although most protocols and projects examined in this report support vote delegation (in practice or in theory), much work remains to be done at the intersection of privacy and delegation. Few protocols support delegate vote privacy (which is key to preventing vote marketplaces) and no protocol supports delegate vote disclosure (which is key to addressing the delegator-delegate principal-agent problem). Furthermore, no onchain protocol supports mid-vote delegate updatability.

4. **Improve Developer Experience (DevEx)** - Private voting protocols come with significant DevEx hurdles. Work can be done to improve the DevEx of integrating and using these protocols. Examples include building plugins/modules to popular DAO tooling providers that act as a plug and play option for private voting.

5. **Gas Efficient Voting and L1 Voting** - We need voting protocols that achieve the best of both worlds: asset security on Ethereum L1 and cheap gas on L2 transactions. The most popular DAOs in the Ethereum ecosystem are deployed on Ethereum mainnet. Cross-chain L1-L2 solutions can tackle this problem by allowing vote submission in L2s but publishing the results in L1. Protocols that leverage cryptography come with significant cost compared to simply voting transparently. Novel solutions could improve the affordability of private voting and decrease the gas costs of doing so on L1

6. **Privacy Awareness and Demand** - Despite calls to introduce private voting, we have not yet seen a strong demand from DAOs. A clear marketing strategy could promote the benefits of privacy in DAOs. The aim here is to promote the narrative that privacy for DAOs is good.

7. **Quantum Resistant Private Voting** - Quantum computers are a medium to long-term threat to the Ethereum ecosystem. When they arrive, they could break encryption and interfere with the election process in different ways (vote submission, votes tallying, results publishing, etc.). The current state-of-the-art cryptographic primitives used in voting protocols take advantage of homomorphic encryption and threshold encryption properties. As an ecosystem we need to research and find new quantum-resistant primitives that have the same properties but are safe from quantum attacks.

Conclusion

Private voting on Ethereum is now a rapidly maturing reality. As outlined in this report, the space now includes a diverse range of protocols and projects pushing the limits of private voting using zero-knowledge proofs, threshold encryption, ElGamal encryption, and anonymous identity frameworks. Many of these efforts are no longer solely confined to prototypes—they are production-ready, audited, and actively empowering decentralized onchain communities as well as real-world political institutions. A number of exciting projects are soon going to production which will push the space forward even more.

Acknowledgments

This report was prepared in collaboration with the Shutter Network team and the Privacy Stewards of Ethereum (PSE) team. We thank all the private voting teams in this report for their help and guidance in reviewing our analysis.

This work was inspired by the State of Private Voting 2024 report written by Odyslam.