BLOCKSTAND

Standardizing On-chain Confidential Governance Operations

Abstract

The work provides a comprehensive technical framework for implementing standardized, confidential on-chain governance infrastructure, with a focus on supporting real-world institutional adoption across cooperative, associative, and federated structures in Europe. It addresses persistent challenges in blockchain-based governance, namely low participation, limited enforceability, and regulatory friction by designing a privacy-preserving voting architecture that maintains auditability and compliance. The study integrates Ethereum's emerging modular standards, such as ERC-4337 (Account Abstraction), EIP 7702 and ERC-7579 (Modular Smart Accounts), with cryptographic primitives including zero-knowledge proofs, zk-nullifiers, and fully homomorphic encryption (FHE), to ensure verifiable, pseudonymous participation and end-to-end confidentiality. Supporting standards such as ERC-4824 (for off-chain metadata) and ERC-7683 (for cross-chain messaging) extend the system's applicability across multichain environments and legal contexts. Comparative analysis of leading privacy-preserving governance protocols demonstrates the feasibility and trade-offs of different approaches.

The result is a blueprint for trust-minimized, privacy-preserving digital governance systems that align with EU regulatory priorities and promote scalable, interoperable institutional use.

Confidential On-Chain Governance

Introduction

Bridging on-chain votes with real-world legal effect remains difficult. Organizations like cooperatives or associations require that digital voting methods produce outcomes recognized under law. Without legal recognition and enforceability, on-chain votes risk being seen as informal. Additionally, ensuring that only authorized members vote (and only once) is crucial for real-world governance (e.g. one-person-one-vote in cooperatives). These challenges highlight the need for more secure, private, and standardized governance mechanisms. Confidential voting has emerged as a key requirement to improve participation and compliance in digital governance, as discussed next.

On-chain governance using blockchain smart contracts for community decision-making holds promise for transparency and automation. However, practical adoption faces major challenges in engagement, legal compliance, and real-world applicability:

- Low Engagement & Participation: many DAOs suffer from voter apathy and low turnout, concentrating power in the hands of a few active or wealthy participants. This undermines the decentralized ethos, as low participation can centralize decision-making and erode the legitimacy of outcomes.
- Legal & Regulatory Compliance: aligning blockchain governance with external laws is complex. On-chain processes must meet regulatory standards (e.g. KYC/AML, privacy laws) without sacrificing decentralization

There is often a tension between a blockchain's openness and the centralized requirements of regulation (identity verification, audit trails, etc.). Many jurisdictions lack clear rules for DAO decisions, making compliance a moving target.

The Need for Confidential Voting in Digital Governance

In traditional democratic processes and corporate governance, the secret ballot is a cornerstone to protect voter freedom and prevent undue influence. Digital governance systems similarly benefit from confidential voting: it encourages honest participation by reducing fear of retaliation or social pressure, and it aligns with legal norms (many jurisdictions mandate secret ballots for certain elections). However, blockchain governance to date has favoured transparency – every vote is typically public on-chain – which can conflict with privacy needs.

Confidential voting aims to balance transparency with privacy and compliance. With the help of cryptography, It is possible to have open verification of results and integrity without revealing individual votes. This is crucial to satisfy both blockchain's demand for transparency and regulators' demand for privacy and fairness. In other words, modern techniques allow us to avoid a false choice between privacy and compliance[1].

By using zero-knowledge proofs, encryption, and pseudonymous identities, a voting system can keep individual choices secret while still providing an auditable trail that the tally is correct and only eligible participants voted.

Why confidentiality matters?

- Prevents Coercion and Bribery: If votes are secret, it's impossible to prove how one voted, which deters vote-buying and coercion attempts. Voters can act according to their true preferences (a property known as receipt-freeness, meaning voters cannot obtain a "receipt" to show how they voted).
- Ensures Regulatory Compliance: In many contexts (e.g. shareholder votes, union votes, cooperative assemblies), secret ballots are legally required to protect voter rights. A confidential on-chain voting system can meet these requirements, enabling blockchain-based votes to be compliant with privacy laws and governance rules, thereby building regulatory trust in the system.
- Balances Transparency: While individual votes are hidden, the overall process remains transparent and verifiable. Every eligible vote is counted and the tally is correct, but observers (and even vote organizers) cannot link votes to voters. This preserves trust in the outcome without compromising privacy.

[1] https://stanford-jblp.pubpub.org/pub/onchain-privacy-compliance/release/1#:~:text=We%20argue%20that%20advances%20in,choice%20between%20privacy%20and%20compliance

Ethereum's Modular Standards for Privacy-Preserving Governance

To achieve this balance, It was decided to use Ethereum's modular standards and advanced cryptography. Those standards could be referenced into the OASIS Project, supported by Ethereum foundation and referenced into the Rolling Plan for ICT Standardisation 2023[2].

The following sections explain how a combination of Ethereum improvements (ERC-4337, ERC-7579, etc.) and cryptographic tools (zero-knowledge proofs, nullifiers, fully homomorphic encryption) can underpin a privacy-preserving, auditable, and pseudonymous voting system suitable for institutional use.

Building a confidential governance system on Ethereum requires several building blocks working together. Recent standards and technologies provide modules for identity, privacy, and execution that can be composed into a unified solution:

1) Account Abstraction (ERC-4337) and Modular Smart Accounts (ERC-7579)

ERC-4337[3] transforms Ethereum's account model by allowing users to employ smart contract wallets as first-class accounts, without changing the consensus layer. Instead of externally-owned accounts (EOAs) directly sending transactions, users submit a UserOperation to an alternative mempool, and specialized nodes called Bundlers package these operations into transactions[4]. ERC-4337 thus enables custom wallet logic (e.g. multi-sig, social recovery, or custom verification) by decoupling account execution from Ethereum's base protocol.

EIP 7702[5] also achieve similar functionalities; it enables Ethereum Externally Owned Account to obtain contract account capabilities. Since this standard proposal is not part of Ethereum Mainnet now (it will be part of the next upgrade called "Pectra and announced on May 7th, 2025), ERC 4337 will be taken as main reference forward.

ERC-7579[6] builds on this by standardizing the interfaces for smart contract wallets to support pluggable modules. In essence, ERC-7579 treats a smart wallet like a LEGO baseplate to which various modules can be attached to extend functionality.. A user's account can load modules for specialized purposes (e.g. session keys, rate-limited spending, or in our case, governance voting logic). Modules are interoperable across different wallet implementations, avoiding vendor lock-in. The standard defines how an account executes transactions and delegates calls to modules, ensuring that any ERC-7579-compliant wallet can work with modules from any provider[7].

What are the implications for governance ?

Using ERC-4337 + ERC-7579, we can create a smart account module for voting that any user's wallet can adopt. This module can house custom verification logic (for example, requiring a valid zero-knowledge proof of identity before allowing a vote transaction). The account abstraction flow (via

---

[2] https://interoperable-europe.ec.europa.eu/collection/rolling-plan-ict-standardisation/blockchain-and-distributed-digital-ledger-technologies-rp2023

[3] https://eips.ethereum.org/EIPS/eip-4337

[4] https://www.alchemy.com/overviews/what-is-a-bundler

[5] https://eips.ethereum.org/EIPS/eip-7702

[6] https://eips.ethereum.org/EIPS/eip-7579

[7] https://medium.com/@vgabrielmarian21/erc-7579-modular-smart-accounts-for-everyone-80a32d1b0f54#:~:text=The%20LEGO%20Analogy%3A%20Understanding%20ERC

UserOperations) means even if the user's action involves complex proofs or credential checks, the infrastructure (bundlers, paymasters) can handle it seamlessly off-chain and then finalize on-chain. Account abstraction thus improves UX and compliance – e.g., gas fees could be sponsored or abstracted, and multi-step vote logic is encapsulated in one user operation. The modular account ensures this vote verification logic is reusable across many wallets and use cases, promoting standardization.

Concretely, one can implement a Validation Module (in ERC-7579 terms) that plugs into the wallet. This module intercepts a UserOperation for casting a vote, verifies required conditions (eligibility proofs, signatures), and only then authorizes the operation. Recent research indeed proposes using an ERC-7579 Validation Module to check zero-knowledge credentials before allowing transactions. By separating user authentication (with zk-proofs) from transaction execution, the account can ensure privacy and compliance throughout the process[8].

2) Self-Sovereign Identity and Merklized Credentials combined with ZK nullifiers

A cornerstone of confidential voting is proving voter eligibility without revealing members identity. This is achieved through self-sovereign identity (SSI) systems that combine the use Verifiable Credentials (VCs) with Merkle trees structure on chain. With this approach, an issuer (e.g. an organization or registrar) provides each participant with a credential attesting their right to vote (for example, membership in a cooperative, or a token holding above some threshold). These credentials can be structured as claims in a merklized identity: each user maintains a Merkle tree of claims, and the root of this tree is anchored as their identity commitment[9].

The process might work as follows:

- A trusted issuer contract on Ethereum issues a claim (credential) to the user's identity. For instance, a DAO might issue a "Member" credential to address 0xABC, represented as a piece of data (with schema, expiration, etc.) that gets included in that user's Merkle tree of identity claims. The issuer contract updates the user's identity state (recording the new Merkle root).
- The user stores their identity (Merkle tree and secret key) in their wallet or identity manager app. They now possess an on-chain verifiable credential that they are an authorized voter. Importantly, this credential does not publicly link their real-world identity, it's just a cryptographic attestation.
- When it's time to vote, the user generates a zero-knowledge proof from their identity credentials. This proof will demonstrate two things: (1) Eligibility: "I possess a valid credential from issuer X (e.g. I am a member of the organization)" – proven via a Merkle inclusion proof and signature verification inside the ZK circuit; and (2) Uniqueness: "I have not voted yet in this election" – enforced via a nullifier. The proof can be generated in the user's device.
- The user submits this proof as part of their vote transaction, likely via the account abstraction UserOperation. The smart account's validation module will verify the proof on-chain against the

[8] https://ethresear.ch/t/enabling-standardized-on-chain-executions-through-modular-accounts/20127
[9] https://docs.iden3.io/protocol/spec/#properties

issuer's public parameters (e.g. the issuer's public key and the latest credential Merkle root). If the proof is valid, the account knows this operation is authorized[10].

This approach means the user never exposes their identity or credential data on-chain – only a ZK proof is shared. The credential itself can encode attributes (like membership status or even voting weight) in a privacy-preserving way. For example, a user could prove "I am in the list of eligible voters" without revealing which entry they are.

The use of merklized credentials is key for scalability and revocation. It allows batching many identities into a single Merkle root (which can be updated as members join/leave) and using efficient Merkle proofs instead of storing individual credentials on-chain. It also aligns with emerging digital identity standards in the EU, where decentralized identity (DID) and verifiable credentials are being explored under frameworks like EBSI (European Blockchain Services Infrastructure) for regulatory compliance. By tying on-chain voting eligibility to verifiable credentials, it is ensured that only legitimate, known participants (e.g. verified by an off-chain KYC or membership process) can vote – an important compliance factor for institutional governance.

Zero-knowledge proofs (ZKPs) enable the system described above: the user proves eligibility without revealing who they are. However, a naive use of ZKPs could allow the same user to vote multiple times unless a way to enforce uniqueness is provided. This is where zk-nullifiers come in. A nullifier is a cryptographic tag derived from a secret (such as the user's private key or identity secret) that identifies a one-time action without revealing the actor. In voting, each voter should have one vote, so we use a nullifier to mark that "some authorized user has used their vote" exactly once.

A common design is to compute nullifier = Hash(identity_secret, election_id). The ZK proof will output this nullifier, and the smart contract will record it to prevent reuse. For example, a recent Nouns DAO private voting prototype had users prove knowledge of an Ethereum private key in a Merkle set and calculate a Poseidon hash of that key with the poll ID as a nullifier. The nullifier is published on-chain, but it's just a random number with no link to the user's actual identity. If the same person tries to vote again, the proof would yield the same nullifier, and the contract can reject it as a duplicate[11]. Nullifiers thus achieve one-person-one-vote without breaking anonymity. Even across multiple votes, if designed properly, nullifiers can be scoped per election so that a user's participation in different votes cannot be correlated. (In the Nouns example, including the poll_id in the hash ensured nullifiers differ each poll

From an implementation standpoint, a Validation Module in the smart account would verify the zk-proof and extract the nullifier. It would check a storage mapping to ensure this nullifier hasn't appeared before for that election, then mark it used. This could be done in the Ethereum EntryPoint's validation phase (pre-transaction) so that an invalid (duplicate) vote is rejected early.

By using ZKPs and nullifiers, pseudonymous voting is accomplished: everyone proves they are allowed to vote and only vote once, but on-chain they appear only as an anonymous proof. No on-chain link exists between the voter's regular address and their voting activity. This greatly enhances privacy and mitigates risks like voter profiling or retaliation.

---

[10] https://ethresear.ch/t/enabling-standardized-on-chain-executions-through-modular-accounts/20127

[11] https://hackmd.io/@eW_xY9qxShKPKKubzERJpw/zk-poll-nouns#:~:text=circuit%20proves%20that%20the%20user,While%20burner%20wallets

3) Fully Homomorphic Encryption for Confidential Tallying

While ZKPs and nullifiers hide who voted and who they voted for, there is still the question of how to hide what the vote was until the final tally. In some governance votes, especially sensitive ones, even interim results or individual votes should remain confidential. Zero-knowledge proofs can prove a vote was valid without revealing its value (for instance, a ZKP could prove "I voted Yes or No" without saying which). However, tallying typically requires summing up the votes. This is where Fully Homomorphic Encryption (FHE) becomes a powerful tool.

Fully Homomorphic Encryption allows computation on encrypted data. Voters can encrypt their vote choices with a public key, and these encrypted ballots can be added together (homomorphically) to compute an encrypted tally. Using FHE, the smart contracts or off-chain agents can perform the tally without ever decrypting individual votes. Only the final result is decrypted (using the corresponding private key) at the end, revealing the aggregate outcome but not the individual contributions.

For example, consider a yes/no vote. Each voter encrypts a 1 for "yes" or 0 for "no" under an FHE public key (which could be published by the organization at the start). The contract receives this encrypted vote and uses an FHE library (like Zama's FHE EVM (fhEVM) toolkit) to add it to a running total. The sum is kept as an encrypted state on-chain. At the end of the voting period, the final encrypted total is decrypted by the key holder, yielding (say) 37 yes votes out of 50. Throughout the process, no one learned any individual's vote – only the final counts are revealed.

Projects like Zama's Suffragium demonstrate this approach, combining ZKPs for identity with FHE for vote secrecy. Suffragium maintains privacy, integrity, transparency, and verifiability by leveraging these techniques. In Suffragium's design, voters undergo an identity verification via ZK proof (e.g., based on an email or other credential) and then cast an encrypted vote on-chain. The smart contract (running on a modified EVM that supports FHE operations) updates the encrypted tally state with each voteEven the authority managing the vote cannot see individual votes; they only decrypt the final tally at conclusion. This ensures that individual votes remain confidential throughout the process, aligning with the principle of secret ballots while still using a public blockchain[12].

One practical consideration is key management for FHE. Unlike ZK, which needs no secret key at verification time, FHE does require a decryption key for the results. A robust implementation for institutional use would use threshold encryption where the private key is split among multiple trustees, or even executed via Multi-Party Computation, so that no single party can decrypt alone.

Vocdoni's DAVINCI protocol, for instance, uses threshold ElGamal encryption with a distributed key generation among voting sequencers[13]. This kind of approach can be adopted in an Ethereum context as well: e.g., a committee of oracles or contract guardians jointly hold the FHE decryption key, or a secure enclave arrangement is used to reveal results.

In summary, FHE gives us confidential computation in governance. Combined with ZK identity proofs, we get a voting system where: eligibility is proven by credentials, anonymity is preserved (via nullifiers), votes are end-to-end encrypted, and the tally is verifiable and correct (via on-chain computations and

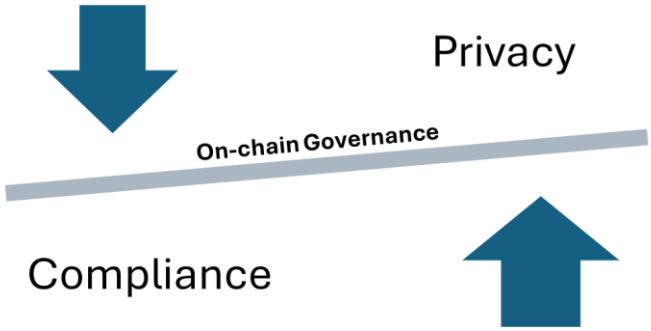
---

[12] https://www.zama.ai/post/encrypted-onchain-voting-using-zk-and-fhe-with-zama-fhevm#:~:text=Once%20voters%20successfully%20generate%20their,trust%20within%20the%20voting%20process

[13] https://hackmd.io/@vocdoni/BJY8EXQy1x#:~:text=Mainnet.%20The%20system%20,voting%20and%20simplifying%20civil

final decryption proof). The outcome is auditable, is possible to publish the proof of correct decryption and even allow independent re-count by re-encrypting the result to compare with on-chain encrypted sum, satisfying transparency. This blend of privacy and auditability directly addresses the privacy/transparency balance needed for compliance. As the Stanford Blockchain Law journal noted, such cryptography lets us overcome the "binary" of choosing either privacy or compliance, in fact we can have both[14].

4) Interoperability via Off-Chain Metadata and Cross-Chain Standards



Real-world governance often involves rich context (charters, proposal documents, etc.) and sometimes spans multiple blockchain networks or communities. Two emerging standards help integrate our confidential voting system into a broader, interoperable governance ecosystem:

Off-Chain Metadata

While the vote execution and tally occur on-chain, much of the governance context lives off-chain – proposal descriptions, supporting materials, voter discussions, etc.

ERC-4824[15] represents DAO metadata standard, defines a structured way to link on-chain governance contracts to off-chain information via a daoURI. Essentially, a daoURI is a pointer (for example, an IPFS or HTTPS link) to a JSON document that describes a DAO's key properties: its name, purpose, membership criteria, and crucially, proposals (with human-readable text, attachments, etc.).

In a confidential voting setting, ERC-4824 can be used to ensure transparency of governance inputs even as individual votes are hidden. For instance, when a new proposal is created on-chain, the proposal contract can include a daoURI reference that contains the proposal title, description, and any off-chain references. Stakeholders (and auditors) can retrieve this to understand what is being voted on. This way, even if the votes themselves are encrypted, the community can openly verify the proposal details and the fact that a vote is underway. ERC-4824 standardizes this process, making it easier for

[14] https://stanford-jblp.pubpub.org/pub/onchain-privacy-compliance/release/1#:~:text=We%20argue%20that%20advances%20in,choice%20between%20privacy%20and%20compliance

[15] https://eips.ethereum.org/EIPS/eip-4824#:~:text=An%20API%20standard%20for%20decentralized,representations%20of%20membership%20and%20proposals

user interfaces (like DAO dashboards) to automatically fetch and display governance info across different organizations[16].

Additionally, ERC-4824 covers membership representation. For example, it can provide a list or criteria of who is considered a member of the DAO (which could, in our system, simply point to the credential registry or Merkle root being used for eligibility). By having a common JSON schema for these, institutional users and regulators can parse the governance structure of any standardized DAO. This contributes to operational transparency (everyone knows the rules of the game and the current proposals) even though the voting process under the hood is privacy-enhanced.

In short, ERC-4824 ensures that adopting privacy does not mean operating in obscurity, governance may be documented off-chain in a standardized, accessible format. A policy officer or compliance auditor could retrieve the daoURI for a vote, see the proposal content and metadata (e.g., voting window, options, required quorum), and later see the final outcome posted on-chain, all without needing to see individual votes.

Cross-Chain Messaging Execution On-chain

As blockchain adoption diversifies, governance might not be confined to a single chain. For example, an industry consortium might have a presence on multiple networks (Ethereum mainnet for record-keeping, plus a private chain or an L2 for operations). ERC-7683[17], the Cross-Chain Intent standard, provides a unified interface for expressing and executing user intent based actions across chains. Initially proposed for things like cross-chain token swaps, the concept of intents can equally apply to governance decisions.

In a federated governance scenario, one member could use ERC-7683-like mechanisms to propagate voting actions or results across chains. For instance:

- Votes might be cast on an L2 (for cheaper transactions and use of custom zk circuits) but the final intent is to execute an action on mainnet (like updating a registry or transferring funds). A cross-chain message can carry the verified result to the mainnet contract for execution.
- Alternatively, separate sub-DAOs on different chains might each run a vote on a common question, and then aggregate those results into a global outcome. Cross-chain messages could be used to send each sub-DAO's encrypted tally to a master chain or to exchange nullifier information to ensure no double voting across networks.

ERC-7683 aims to standardize how such messages are formatted and handled, making it easier to build multi-chain governance workflows that remain consistent. By using a standard, it would be possible to rely on shared permissionless infrastructure (eg. relays, bridges) with a common function logic rather than custom integrations for each project[18]. In practical terms, this means our confidential voting module could emit a cryptographic proof or intent that "Proposal X passed with Y votes" which could be consumed in trust-less way on another chain to trigger an outcome, using the common format.

For EU institutions or consortia, this is important: it enables federated governance. Imagine a federation of cooperative banks each on their own private Ethereum network, coordinating a joint decision. Each

[16] https://dhive.io/proposal/606
[17] https://eips.ethereum.org/EIPS/eip-7683
[18] erc7683.org

bank's members vote privately on their chain; then a cross-chain intent is sent to a central public chain contract that tallies or acknowledges the votes from each branch.

The end result is an interoperable, multi-chain governance process – all still using encrypted, privacy-preserving votes locally, but sharing results globally. ERC-4824 and ERC-7683 complement the core privacy tech by ensuring the inputs and outputs of governance are standardized and shareable. Off-chain metadata provides human-readable transparency and context; cross-chain intents provide technical interoperability across systems. Both are key to institutional adoption: they allow our confidential voting system to plug into larger governance workflows and multi-chain environments without losing clarity or connectivity.

Architecture & Design of a Confidential On-Chain Voting System

Bringing together the above components, we can outline a technical architecture and process flow for a standardized confidential voting system. This design will use some references from existing protocols as empirical validation of the process (Iden3 identity proofs, ERC-4337 UserOperations, a zk-proof validation module with nullifiers, and FHE-based tallying). The process is anchored in to a four step process the description.

Process	On-chain Identity Issuance
	User-Ops Vote Casting
	Account Validation & Execution
	Tallying & Verification

1. System Setup and on-chain Identity Issuance:

- **Issuer & Identity Setup:** The appointed authority operates an Identity Issuer Contract (following the Iden3 identity standard). All eligible members are onboarded by having a decentralized identity (DID) created for them (if they don't have one already). Each member's DID is essentially a public key pair and a Merkle-tree state of claims. The issuer contract mints a verifiable credential to each member's DID indicating voting eligibility (e.g., "Member of X where "X" is the authority, valid for XXX number of votes"). This could be done by writing a hash of the credential into the on-chain Merkle tree or by signing it off-chain and letting the user include it in their identity tree.
- **Publishing Election Details:** A Vote Contract is deployed on-chain governance framework for the specific election or proposal. This contract might be a minimal contract that stores the vote parameters (proposal ID, voting period, encryption public key, etc.). The contract also includes the Merkle root (or roots) of eligible voters or a reference to where that root can be obtained (perhaps via an ERC-4824 metadata field). This serves as the source of truth for who can vote. The vote contract also generates or is associated with an FHE public key for encrypting votes (and the corresponding private key is split among trustees or kept secure off-chain for result decryption).
- **User Wallet Preparation:** Each user (member) uses a wallet that supports account abstraction (ERC-4337 contract accounts) and implements a governance voting module (a ERC-7579 module). The module is configured with the necessary verifier contracts (e.g., the ZK proof verification key or contract address, the issuer's public key for credential signatures, and the nullifier tracking logic). Users might also install a client application, potentially integrated with their wallet app, that can generate the ZK proofs (for example, an app or a browser extension that handles the proof generation when prompted).

2. Casting a Vote (User Operation Flow):

When voting opens, a member can cast their vote with the following steps:

1. **Proposal Review:** The user fetches the proposal information via the daoURI provided by the Vote Contract (off-chain metadata with details of the vote). They decide their vote (e.g., YES/NO or a choice among candidates).
2. **Proof Generation:** The wallet's identity app creates a zero-knowledge proof of the user's eligibility. The proof will typically be generated by selecting the appropriate credential (membership) and the target Vote Contract (which provides the election ID or Merkle root and nullifier salt). The result is a proof, π, which encodes: "I am a member in the Merkle root R" (with issuer's signature) and "I have not voted in election E yet", along with a calculated nullifier value N for this voter in this election. Simultaneously, the user encrypts their vote using the FHE public key provided by the Vote Contract. For example, if the vote is YES (1), they use the FHE scheme (say, TFHE or similar) to encrypt the bit 1 into an encryptedVote ciphertext.
3. **UserOperation Creation:** The wallet assembles a special transaction, an ERC-4337 UserOperation targeting the Vote Contract's castVote function (or a generic relay contract). Instead of a direct transaction, the UserOperation includes:

- The target contract and function call (e.g., castVote(electionId, encryptedVote, proofData)).
- The call data carries the encrypted vote and the ZK proof outputs (which might include the nullifier and some public inputs like a membership commitment).
- The signature field of the UserOperation is not a traditional EOA signature but will be used by the wallet's Governance Module as a container for the zero-knowledge proof verification. (In ERC-4337, the EntryPoint will call the wallet's validateUserOp method to check if this operation is authorized.

4. **Submission to Mempool:** The UserOperation is sent to the ERC-4337 mempool (via a Bundler RPC endpoint). Notably, the user could choose to submit it through a trusted relayer if they want extra privacy (so their IP is not linked, etc.), or even via a specialized network (some implementations allow bundlers to accept encrypted userOps that only they can decrypt, but that's beyond our current scope). The key point: the user does not need to pay gas directly; they might use a Paymaster mechanism, or the bundler could accept to sponsor certain governance ops. This relieves users from holding ETH, which is practical for participants who only have an identity credential.

3. Validation and Execution (On-Chain):

- **EntryPoint and Wallet Validation:** A bundler picks up the UserOperation and sends it to the Ethereum EntryPoint contract (the core of ERC-4337). The EntryPoint calls the user's smart account (wallet) validateUserOp method. Inside the wallet, this call is forwarded to our Governance Voting Module (which acts as the validation logic for this type of operation).

The module then performs the following checks:

- It invokes the embedded ZK Verifier contract to verify the proof π against the expected inputs (the election's Merkle root, etc.). If the proof is invalid, validation fails and the UserOperation is rejected.
- If the proof is valid, the module retrieves the nullifier N from the proof. It checks a storage (either in the module or a global nullifier registry contract) whether N has been seen for this election. If N is already present, it means this identity voted before – the module then fails validation (causing the operation to be rejected). If N is not present, the module records N (e.g., in its storage or by calling the Vote Contract to register the nullifier). This ensures the identity cannot vote again.
- The module may also enforce any other policy (for instance, checking the election is currently open, etc., though typically the Vote Contract will handle timing).
- After these checks, the module returns success to EntryPoint, possibly also a "signature valid" indicator (in ERC-4337, the module might internally use isValidSignature via ERC-1271[19] to signal approval). Essentially, the module says "this UserOp is authorized – the user proved themselves".

[19] https://eips.ethereum.org/EIPS/eip-1271

- **Executing the Vote Cast:** Once validated, the EntryPoint proceeds to call the Vote Contract's executeUserOp (or directly the function, depending on implementation). This triggers the actual castVote function on the Vote Contract, now in the context of the user's account. The Vote Contract, upon castVote, will:
    - Verify again that this nullifier N hasn't been used (or trust the wallet module's mark – a design choice; double-checking is a safety net in case of replay from different accounts).
    - Mark the voter as having voted (could simply be storing N in a mapping voted[N] = true). This is the on-chain permanent record preventing double voting.
    - Accept the encrypted vote and add it to the tally. If using FHE with an on-chain library (like Zama's FHE toolkit), the contract will have a state variable encryptedResult (initialized to encryption of 0). It will call an operation like encryptedResult = TFHE.add(encryptedResult, encryptedVote)[20]. This homomorphic addition updates the encrypted tally. Optionally, the contract could emit an event with the new encryptedResult or with the vote (still encrypted) for transparency that a vote was counted.

The contract does not need to know the plaintext vote; it only stores the encrypted sum. If the vote is more complex than yes/no (say a choice among candidates or a ranking), the encryption and tallying scheme would be set accordingly (e.g., use a vector of counts, or use multiple ciphertexts). Also, the contract locks out further votes from the same nullifier (any further UserOp with same proof nullifier will fail as ensured).

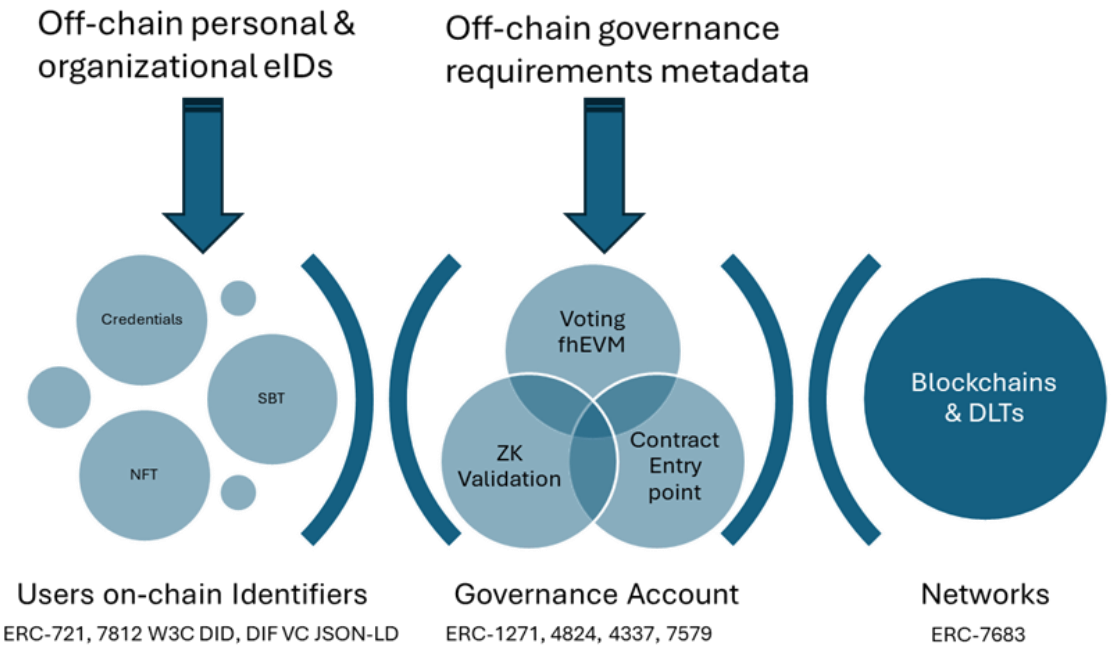
Because the actual execution is done through the account abstraction framework, the user's own account was the origin of the transaction, and it only executed after passing the zk proof check. This separation of concerns makes the system modular and secure: even if someone tried to call the Vote Contract directly without the proper proof, the contract's logic plus the account module's logic ensure it won't succeed.

4. Tallying and Verification:

- **During Voting:** Throughout the voting period, votes are being added homomorphically. No one knows the interim results. This prevents momentum swings or undue influence that might happen if live results were public.
- **Closing the Vote:** After the voting deadline, the Vote Contract can be instructed to finalize. At this point, the final encryptedResult contains the sum of all votes. The designated decryptors will perform a decryption of the homomorphic tally. For example, if using threshold FHE, a set of trustees each provide partial decryption shares which are combined to reveal the plaintext result. The outcome – say "Yes: 30, No: 20" – is then either published on-chain by calling a publishResult function or emitted as an event. The system can also produce a Zero-Knowledge proof of correct decryption to post on-chain, to assure observers that the decrypted result indeed corresponds to the on-chain encrypted tally.

[20] https://www.zama.ai/post/encrypted-onchain-voting-using-zk-and-fhe-with-zama-fhevm#:~:text=ebool%20support%20%3D%20TFHE

- **Auditability:** Now the result is out, but individual votes remain secret. Anyone can verify that the number of votes counted equals the number of unique nullifiers (to ensure no missing or extra votes). The cryptographic audit trail includes: the list of nullifiers (proving number of voters), the final encrypted tally (on-chain data), and the proof of decryption to the result. Additionally, each voter can be given a way to verify their vote was included. This can done by providing the voter with a unique receipt in zero-knowledge. In our design, the voter's nullifier could serve as a private receipt (the voter knows their secret, hence knows their nullifier, and can check that it appears in the election's record, thereby confirming their vote was counted, without revealing which vote was theirs). This achieves end-to-end verifiability, in other words, the voters trust the process because they can verify inclusion, and observers trust it because they see proofs of correctness[21].
- **Execution of Outcome:** If the governance process is binding, the result can trigger on-chain actions. For instance, if the vote was to allocate funds or change a parameter, the final step might be an on-chain transaction executed by the Vote Contract or a connected governance executor. Thanks to this design, it can be done automatically and trust-less way once the outcome is known. Cross-chain intent standards (ERC-7683) could be used here to propagate the decision to another chain if needed (e.g., an L2 where a DAO's treasury resides could accept a cross-chain message that "Proposal X passed with Y votes" and then execute the corresponding action).



Throughout this flow, we used standard components: the ERC-4337 account abstraction for flexible execution, ERC-7579 modules for plugging in custom logic, on-chain identifiers, ZK-SNARKs for privacy and nullifiers, and FHE for encrypting votes. Each component has a clear responsibility and is replaceable or upgradable as tech evolves (for instance, one could switch out the FHE scheme or use a different ZK proving system without changing the high-level architecture).

[21] https://maci.pse.dev/#:~:text=enables%20on,help%20ensure%20fair%20and%20transparent

This architecture is privacy-preserving, auditable, pseudonymous, and standardized. It enables an organization to run a vote where members are known and verified off-chain (satisfying legal KYC requirements), but on-chain they participate under cryptographic pseudonyms. The outcome is reliable and can be enforced by smart contracts, bridging the gap between a traditional secret ballot and decentralized governance execution.

Market solutions: a comparison of Privacy-Preserving Voting Protocols: Da Vinci & MACI Protocols

To put our approach in context, I compare two prominent real-world solutions for decentralized, private voting: **Vocdoni's Da Vinci Protocol**[22] and Ethereum's **MACI**[23] **(Minimal Anti-Collusion Infrastructure)**. Both aim to solve similar problems (secret ballot, anti-bribery), but with different designs. Below is a feature-by-feature comparison:

Feature	**Vocdoni – Da Vinci (DAVINCI)**	**MACI (Minimal Anti-Collusion Infrastructure)**
Architecture	Layer-2 zkRollup specialized for voting. Uses its own network (Vocdoni was originally on Vochain BFT and now evolving to an Ethereum-secured rollup). Integrated smart contracts on Ethereum coordinate the rollup and data availability.	Smart contract system on Ethereum (L1 or L2) with off-chain coordinator. Primarily an on-chain contract + off-chain server model: votes are posted to Ethereum (as encrypted data), tallying happens off-chain by a coordinator, and results are proven on-chain
Privacy Mechanism	zkSNARKs + Threshold Homomorphic Encryption: Uses zkSNARK proofs to ensure each vote is valid (one per voter, voter is registered) and ElGamal homomorphic encryption for vote contents. Multiple sequencers jointly generate keys and perform threshold decryption, so no single point can see votes. Ensures strong privacy and decentralization of trust.	Encryption + zk-SNARKs: Each voter's message (vote) is encrypted with the coordinator's public key. All votes are posted encrypted. Coordinator (or a set of managers) decrypts in a special way only after voting ends. Uses zk-SNARK proofs on-chain to prove the tally is correct without revealing individual votes. Privacy relies on the encryption; individual votes are never plaintext on-chain.
Anti-Collusion & Bribery	Receipt-freeness is explicitly addressed. DAVINCI allows vote overwriting – voters can submit multiple times, only the last counts. And by re-encrypting votes randomly, even a coercer cannot be sure a voter didn't change their vote. This makes it extremely hard to prove how one voted, deterring bribery and coercion.	Minimal Anti-Collusion is the namesake: MACI ensures a user cannot prove how they voted. It achieves this by having a coordinator shuffle and process votes in a way that outsiders can't link inputs to outputs, and by allowing "message override" (if someone bribed a voter to vote a certain way, the voter can later override that vote with another, invalidating the bribe). MACI's encryption and

[22] https://hackmd.io/@vocdoni/BJY8EXQy1x#:~:text=match%20at%20L514%20Handling%20Collusion,new%20vote%20from%20a%20voter

[23] https://maci.pse.dev/docs/introduction

Feature	Vocdoni – Da Vinci (DAVINCI)	MACI (Minimal Anti-Collusion Infrastructure)
		nullifier scheme inherently prevent a voter from obtaining a receipt of their vote.
Verifiability	End-to-end verifiability: Voters can verify inclusion of their vote in the rollup (each vote has an inclusion proof in the batch). zkSNARKs ensure the rollup state transition (tally update) is correct each batch. The final tally is publicly verifiable: anyone can audit the zero-knowledge proofs and the threshold decryption proof to confirm the result	Public verifiability via on-chain proofs: After the coordinator computes results, it posts a zk-SNARK proof to the Ethereum contract that the tally was computed correctly from the encrypted votes. The contract verifies this proof before accepting the outcome. The scheme is receipt-free, but each voter can be given a way to confirm their vote was included (for instance, MACI can publish a Merkle tree of votes). The design is trust-minimized: one must trust the coordinator not to censor, but even if they try, it can be detected.
Scalability & Cost	High scalability via dedicated rollup: thousands of TPS for voting have been demonstrated (Vocdoni achieved ~700 TPS on their earlier chain). Using data availability on Ethereum (EIP-4844 blobs) to keep costs low. Suitable for large-scale voting (millions of voters) by batching operations and using L2 efficiency. The trade-off is more complex infrastructure (sequencers, rollup).	Scalability is moderate: since all votes (encrypted) and proofs end up on Ethereum L1, cost grows with number of voters. MACI is efficient for tens of thousands of voters in practice when used on an L2 or sidechain. It relies on heavy off-chain computation (coordinator performs zk proof generation which can be intensive). There is ongoing work to optimize MACI for larger quadratic voting use cases. For now, it's suitable for small to mid-sized votes or quadratic funding rounds, but might become expensive at nation-state election scale without L2 help.
Standardization & Extensibility	Vocdoni's DAVINCI is moving toward a universal voting protocol vision. It is not yet an Ethereum standard, but it heavily uses standard components (zkSNARKs, Ethereum contracts for coordination). It provides APIs for organizations to run votes. Being a specialized network, integration means interfacing via bridges or SDKs. Its design principles (cryptography as source of truth, trustlessness, etc.) align with the goals of standardized confidential voting.	MACI is more a building block than a plug-and-play product. It's open-source and developers can integrate MACI contracts and circuits into their dApps (for example, clr.fund integrated MACI for quadratic funding votes). There is no ERC standard number for MACI, but it's become a de facto reference for anti-collusion voting. It's extensible (one can modify the circuits for different vote tally logic, e.g., quadratic voting, rank-choice). Tools like MACI wrappers and libraries are emerging to simplify deployment.

Both protocols share the goal of privacy and integrity in voting, but take slightly different approaches in trust models.

DAVINCI leans on decentralization (multiple sequencers, threshold keys) and high performance via a rollup – making it appealing for institutional adoption where scalability and removal of single points of trust are key.

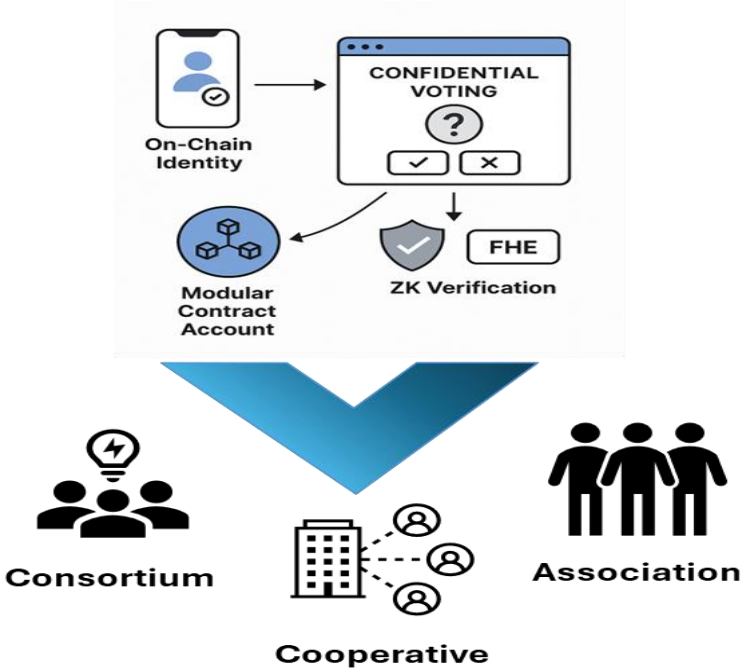
MACI, on the other hand, offers a very elegant Ethereum-native solution with minimal moving parts (one main server role) and heavy use of smart contracts to enforce correctness. MACI might still require some trust in the coordinator for availability (they could censor votes, though not alter them without detection), whereas DAVINCI's decentralized sequencers can mitigate censorship by design.

The proposed architecture in this document is closer to MACI in that it can be implemented directly on Ethereum with a combination of contracts and off-chain proof generation. However, it also aspires to DAVINCI's decentralization by suggesting threshold key management for FHE and potentially multiple validators (the ERC-4337 bundler network could even play a role in decentralizing the transaction processing).

In fact, the envisioned solution deployed functionality as an ERC-7579 module as the "voting module" described echoes protocols logic (nullifiers, proofs, etc.) but within each user's wallet, rather than a single global contract. This modular approach, aligned with standards, is what would make a solution truly interoperable and future-proof for institutional use.

Potential Business Scenarios

To conclude, let's ground these concepts in real-world scenarios, particularly relevant in the EU context, to illustrate the impact of confidential on-chain governance.



1) Cooperative Governance with Privacy and Compliance

Scenario: A large European cooperative (e.g., a farming coop or a financial mutual) with 10,000 members spread across EU countries wants to adopt on-chain voting for its General Assembly decisions. By law, each member gets one vote (regardless of capital contributed) and many votes must be by secret ballot. Regulators require that voter identities are verified (to ensure only legitimate members vote) and that results can be audited if challenged.

Application: Using the system the cooperative issues **digital membership credentials** to all members (perhaps reusing a national e-ID or a KYC process, but turning the result into a verifiable credential). When a vote is called, members vote through the confidential voting smart contract.

- The pseudonymity means members can vote without fear of retribution from management or peers, which is crucial in, say, votes about leadership or sensitive policy changes. This could increase participation because members trust that their privacy is protected (addressing the engagement issue of on-chain votes being too public).
- Regulatory trust: An auditor or regulator could be given access to certain parts of the system for oversight. For instance, the cooperative's notary (common in EU coops) might be one of the threshold key holders for decryption. They can co-sign the tally decryption and certify that the vote was conducted fairly. The verifiable proofs (membership proofs, tally proofs) are all on-chain, which could be presented in a compliance report. The system inherently enforces one-member-one-vote, a key requirement, and this can be demonstrated via the nullifier logs and credential audits.
- Operational enforceability: Suppose the cooperative's bylaws say a successful vote automatically triggers a policy change. The on-chain execution can directly implement the decision (like updating a smart contract that represents the cooperative's bylaws or triggering a payment distribution if it was a dividend vote). This removes ambiguity between the vote and action – unlike paper votes which then require manual implementation, here it's automatic yet still legally sound due to the above compliance features.

Importantly, the cooperative doesn't need to trust a tech vendor's black-box system; it's using open standards and open-source contracts. This aligns with EU procurement preferences for transparency and interoperability. If the cooperative later joins a federation or needs to migrate to a different blockchain, the use of standards (ERC-4824 metadata, ERC-7579 modules, etc.) ensures portability of the governance process.

2) Professional Associations and Unions with Pseudonymous Participation

Scenario: A European professional association, say a medical association, or a labour union wants to hold elections and referendums among its members. Turnout in these votes is often low due to inconvenience, and members sometimes fear their vote could be known to leadership or employers (for unions).

Application: A confidential on-chain voting system can be provided as a service to all members via a simple web interface or wallet. Members verify their identity once to get a digital membership ID, then vote in various ballots pseudonymously.

- For a union, the fact that votes (e.g., on whether to strike) are secret even on a transparent ledger means members can vote their conscience without fear. The anonymity encourages higher participation, addressing the engagement challenge. At the same time, the union leadership can reassure everyone, outside observers, that the vote was fair and free of manipulation, because the results are independently verifiable and each member's participation (though anonymous) is confirmed by the credential check.
- From a legal perspective, member personal data (names, etc.) is never put on-chain, only hashes and related proofs that mixed homomorphic encryption reduce the sound of personal data. If a member exercises their right to leave the association, their credential can be revoked, by updating the Merkle root, and they can no longer vote. The system inherently practices data minimization (a GDPR principle) by only revealing what's necessary (which is essentially nothing about identity, only eligibility).
- If required, the association could even allow an anonymous but authenticated discussion period for proposals using the same identity scheme – e.g., members post comments or questions in a forum by proving membership via zk-proof, without login. This would create a richer digital democratic process protected by cryptography.
- Enforceability: If the association's decisions need to be carried out in the real world (like adopting a new code of conduct), on-chain execution might be less directly applicable. However, the audit trail and cryptographic proof of the decision provides a tamper-proof mandate. The association could integrate this with their internal records or even notarial record by URI. As EU organizations consider e-governance, having a mathematically certain record of a vote's outcome can reduce disputes and streamline governance.

3) Federated Industry Consortia and Multi-Chain Governance

Scenario: Suppose multiple companies in the EU form a consortium (e.g., a joint venture) with each company as a member. They need to vote on consortium decisions, but each company might want to maintain control of its own identity system or blockchain. Additionally, the consortium might operate across multiple networks (one for internal stuff, one public).

Application: Using cross-chain messaging (ERC-7683 intents), each entity can vote in its local environment and results can be aggregated:

- Each member organization in the consortium has its own governance system (could even be a private instance of the voting contract on a permissioned chain). When a consortium-wide vote is called, an aggregator contract on a main chain (or common L2) is set up with the FHE key and collects encrypted tallies from each sub-vote.
- For example, Company A, B, C each run a vote among their internal stakeholders (using the confidential voting system) to decide how to cast their single vote in the consortium's decision. They then send an intent message to the consortium contract with either their encrypted vote or their portion of votes. Alternatively, if each company has one vote by structure, they could simply send a signed message approving or rejecting a proposal. But if those need to be private until all in, they could encrypt and then collectively decrypt. The consortium contract then tallies those or waits for all and publishes outcome.

- In a multi-chain DAO case, one could have a hierarchical vote: each sub-DAO votes privately, sends the result to the main DAO contract which then computes the combined outcome (perhaps weighted by sub-DAO sizes). Because of standardization, each sub-DAO's voting contract speaks the same "language" (maybe all follow ERC-4824 for proposal metadata and a standard interface for result reporting). Cross-chain communication carries over the needed proof that "SubDAO X's result was Y with proper verification".

In both cases, pseudonymity and confidentiality might be needed not just at the individual level but at the subgroup level. In a consortium, perhaps each company's stance is sensitive information. Using threshold encryption, the consortium could even decide to reveal only the final outcome (e.g., "Approved 2-1") without naming which company voted which way, if that was desired to maintain harmony. Alternatively, if each organization's vote is public by agreement, they could choose not to encrypt that phase. The point is, the tools are flexible to permit different levels of disclosure.

By employing standards-based modules, such consortia can implement this without building bespoke systems from scratch. This fosters interoperable governance. Interoperable governance should be key interest for cyber resilient institutions that want to avoid being locked into a single vendor or chain. For example, the European Union itself could someday run a multi-country vote where each country's votes are cast on a national blockchain and aggregated at an EU-level smart contract.

Conclusion: Emphasizing Trust, Pseudonymity, and Enforceability

Across these use cases, some common themes emerge as paradigm for standardized on-chain governance operations:

- **Regulatory Trust:** By design, the system should provide an open audit trail that regulators or authorized third parties can inspect. Instead of trusting a closed electronic voting system, regulators get mathematical guarantees. The use of open standards (Ethereum EIPs) and well-vetted cryptography (zkSNARKs, FHE) means the system's integrity doesn't depend on hidden algorithms. Moreover, the system can be tailored to meet legal requirements (like keeping data in certain jurisdictions or enabling an authorized data escrow to deanonymize in case of court order – although that last resort undermines privacy, it could be built if absolutely required under specific laws).

- **Pseudonymous Participation:** Participants should be identifiable to the system but not to each other. This is an advantage in diverse settings: it removes biases (votes are judged only by content, not by who submitted them), and it protects personal data. It allows entities to engage in governance using digital credentials that carry only the qualifications needed (e.g., "is a member in good standing") and nothing more. This could encourage broader participation, as more users feel safe using the platform.

- **Operational Enforceability:** When a decision is made, smart contracts can directly enforce it (transfer funds, change parameters, record outcomes immutably). This should reduces the gap between decision and action. It also means that smart contract should be able to enforce operations that may be also represented off-chain as agreements encoded in consortium charters or bylaws can be automatically upheld. For instance, if a supermajority vote is required to approve something, the contract will only execute when that condition (supermajority in tally proof) is met, otherwise not – eliminating any possibility of "cheating" the rules. From an

operational standpoint, this significantly increases confidence that the governance process is binding and effective.

Finally, because this document is aimed at supporting institutional standardization in the EU, it is worth noting that these approaches dovetail with EU digital strategy for cyber resilience trends, and data governance regulations that favour privacy. By adopting a standardized confidential voting system, institutions, whether public bodies, private consortia, or cooperatives, they can lead in modernizing governance while staying squarely within legal bounds. This can increase trust from all stakeholders (members, leaders, and regulators) that on-chain governance can be safe, fair, and effective for real-world use.

Confidential on-chain governance is not only technically achievable, but also rapidly becoming necessary infrastructure for bridging blockchain innovation with traditional institutional requirements. By addressing engagement through privacy, ensuring compliance through verifiable credentials and proofs, and enabling real-world enforceability, such systems can transform how organizations make decisions.

In this document, it was analyzed how to build these systems using the modular components Ethereum now offers: account abstraction (ERC-4337) for flexible execution, modular smart accounts (ERC-7579) for extensibility, self-sovereign identity credentials, zero-knowledge proofs (for anonymity and authenticity), and homomorphic encryption (for computing results invisibly). We also integrated auxiliary standards like ERC-4824 for rich metadata and ERC-7683 for cross-chain execution, to ensure that a confidential voting process does not exist in isolation but connects seamlessly with multi-chain, off-chain, and legacy contexts.

The analysis of Vocdoni's DaVinci and MACI protocols highlighted that the ecosystem already has workable solutions on governance. The aim of the work in standardizing should be to be intended to provide an overview of existing solutions to combine their strengths in terms of scalability, trustlessness, anti-collusion to support innovation harmonization into a general framework that any protocol can adopt.

Finally, we analysed some business scenarios into which governance operations are generally key. Use cases in cooperatives, associations, and consortia illustrated that the impact goes beyond technical elegance: it combines inclusive participation, legally robust decisions, and automation of governance in areas that may still leverage opaque frictions for dystopian purposes. Key future EU policies such as the SME Passport may be key enabler environment for these industrial uses to become reality.

Moving forward, institutions and policy bodies in the EU should consider pilot programs using these standardized approaches. Over time, this could evolve into an EU governance standard, potentially informing future EIPs or even legislation for recognized digital voting processes. By embracing confidential on-chain governance, we preserve the core values of blockchain (transparency, security, immutability) while mitigating its risks with respect to privacy and compliance. The result is a pseudonymous yet accountable democracy for the digital age, one that can empower members of any organization to voice their choices freely and confidently.

Eugenio Reggianini

DISCLAIMER