Opn.vote: A Publicly Verifiable Blockchain-Based eVoting System for Democratic Participation

Abstract. We introduce Opn.vote, a publicly verifiable and decentralized voting application that offers both voter anonymity and end-to-end verifiability. Voter anonymity is formally defined using a game-based approach, requiring that even a coalition of all participating authorities should not be able to link a voter to their cast vote, even in scenarios where election keys are maliciously generated.
To enhance usability and improve the voting experience, Opn.vote leverages Ethereum, a public blockchain, with its EIP-4337 (Account Abstraction) to eliminate the need for voters to possess pre-existing digital wallets or cryptocurrency to participate. The design of Opn.vote requires no additional preparation beyond what is expected of a typical voter, significantly lowering barriers to entry and reducing transaction costs by $\approx 51\%$ compared to the leading voting system on Ethereum.
From a practical perspective, Opn.vote is set to be deployed during the ABSTIMMUNG21 referendum in Germany in September 2025. The system aims to transition 125,000 of the 250,000 participants to online voting, reducing transaction costs per vote from €2 to under €0.01. This deployment will serve as a large-scale demonstration of Opn.vote's potential to deliver secure, accessible, and cost-efficient online voting.

1 Introduction

Verifiable elections provide a mechanism through which internal participants and external observers can verify the validity of individual votes and the overall electoral outcome, even if voting devices and servers are faulty or are outright malicious [12]. *Verifiability*, alongside *transparency*, *usability*, and *security*, is key in building trust, influencing each and every case of success and failure in the implementation of e-voting [10]. Despite being central to the success of e-voting at scale, verifiability raises several critical issues.
Challenges in Voter Verification. Statistics from elections using e-voting systems show that a very small percentage of voters actually verify or succeed in performing this verification [5,20]. This low verification rate is largely attributed to the fact that the verification process is not usable enough, too complicated, and confusing for the voters [24]. This issue has received considerable attention in the context of national elections across various countries, including the Netherlands and Germany [16].
Centralized vs. Decentralized Approaches to Verifiability. Verifiability is commonly achieved through the implementation of a public bulletin board [1,14,30], which is typically managed by a singular entity, such as a voting server [13]. However, this centralized approach not only introduces vulnerabilities to single points of failure (e.g., website outages, denial-of-service attacks, and data breaches)

but could also undermine core security guarantees, such as privacy, correctness, and resistance to voter suppression. Public blockchains offer a promising solution by shifting critical processes, such as voting, data storage, and tallying, to a decentralized network. This ensures protocol integrity and correct execution without reliance on a single authority, mitigating risks like censorship, tampering, or system failure. However, as discussed in a recent review by Kharman et al. [23], many blockchain-based e-voting protocols proposed overlook practical constraints such as limited computational resources, high transaction costs, and throughput limitations. Moreover, when practically deployed, these protocols would often require voters to install wallet software and acquire cryptocurrency to pay transaction fees, significantly hindering usability and accessibility (e.g., [25,31]). To the best of our knowledge, there are only two privacy-preserving e-voting systems built on public blockchain technology, which are currently active and in use: MACI (Minimal Anti-Collusion Infrastructure) [25] and Vocdoni [31]. MACI uses zero-knowledge proofs to hide how each user voted, while still allowing verifiability. However, it relies on a single trusted coordinator to tally the final result, which is able to compromise voter privacy and requires a trusted setup. Vocdoni's architecture, on the other hand, relies on a permissioned blockchain (Vochain) for vote casting, controlled by pre-approved entities, and it uses Ethereum only for tasks like election setup. This design centralizes the critical voting process and, as a result, may reintroduce some risks of collusion, and single points of failure that public blockchains aim to eliminate.

Contributions. We present Opn.vote, a publicly verifiable and decentralized e-voting system, designed to address the verifiability challenges in existing voting solutions. With respect to voter verification, Opn.vote enhances the usability of the verification process by publishing the election decryption key to a public bulletin board post-tally. This approach facilitates straightforward end-to-end verifiability and system transparency, eliminating the need for additional proof verification mechanisms.

Moreover, Opn.vote delivers high usability and an improved voting experience by imposing minimal computational requirements on voters. More precisely, voters are only required to encrypt their votes and sign the ballots using their private keys, without the necessity of generating or verifying supplementary proofs. This protocol design leads to a reduction of $\approx 51\%$ of transaction costs compared to MACI [25]. Opn.vote also offers flexibility by allowing voters to recast their votes using the same credentials without requiring re-registration. In comparison to other on-chain voting solutions, Opn.vote demands no additional preparation beyond standard expectations for voter participation. Notably, Opn.vote leverages Ethereum's EIP-4337 (Ethereum Improvement Proposal 4337, also referred to as Account Abstraction) [6] to eliminate the need for transaction fees or wallet creation for voters, both of which are typically required in other on-chain voting systems [23]. Further technical details are provided in Section 4.

To formally model the security of Opn.vote, we generalize it into a generic voting scheme and introduce a notion of *voter anonymity* based on indistinguishability to capture its privacy. In particular, the generic scheme draws inspiration from FOO [18] and Cetinkaya et al.'s approaches [7], in which each voter is issued a public credential by an authorized entity that may potentially act maliciously. The voter derives their private credential from the public one and uses it to vote anonymously. The definition of voter anonymity captures the property that no adversary, including a coalition of all participating authorities, should be able to link a voter to their cast vote, even in scenarios where election keys are maliciously generated or revoting is involved. By proving that the generic voting scheme satisfies the definition of voter anonymity, we also demonstrate that Opn.vote adheres to this standard. This helps Opn.vote remove the single point of failure seen in MACI [25], where a trusted entity might collude, and it avoids the need for a trusted setup. Compared to Vocdoni [31], Opn.vote enhances transparency and resilience by shifting critical processes to a public blockchain. However, Opn.vote assumes the availability of an anonymous communication channel, which we revisit to explore implementation possibilities in light of recent advancements in blockchain-based technologies.

The last contribution consists of assessing Opn.vote's efficiency in large-scale elections; for this purpose, we present a comparative analysis with other voting systems. We note that Opn.vote remains an ongoing project. We discuss the security challenges present in its current implementation and outline future directions for mitigating these issues.

2 Voter Anonymity

We will now formally define the concept of voter anonymity by first establishing the syntax of a voting system, drawing upon the work of Chaidos et al. [8].

2.1 Syntax of a Voting System

Election systems generally consist of multiple entities, which are assumed to be single individuals for the sake of simplicity. The thresholdized version is discussed in the context of a specific application (see Section 4).

Definition 1 (Voting System). *A voting system $\mathcal{VS}$ is a tuple of* PPT *algorithms* (SetupElection, VerifyPK, Register, VerifyCred, Vote, Valid, Append, Publish, VerifyVote, Tally, VerifyResult) *associated to a result function $\rho : (\mathcal{V} \times \mathcal{C})^* \rightarrow \mathcal{R}$ where $\mathcal{V}$ is the set of voters, $\mathcal{C}$ is a set of voting options, and $\mathcal{R}$ is the result space, such that:*

- SetupElection(1^λ)*: On input security parameter 1^λ, generate the public and secret keys* $(\mathsf{PK}, \mathsf{SK})$ *of the election. Below,* PK *is an implicit input.*
- VerifyPK(PK)*: On input the public key* PK*, outputs* 1 *if and only if* PK *is valid;* 0 *otherwise.*
- Register(1^λ) *: Is an interactive protocol run by the registrar* R *and a voter* V *with an identifier* id *to output a public credential* d*. We simply write $\mathsf{d} \leftarrow \langle \mathsf{R}(1^\lambda, \mathsf{id}, \mathcal{V}), \mathsf{V}(1^\lambda) \rangle$ as* d *is the same output of both parties. The registrar then adds the pair* $(\mathsf{id}, \mathsf{d})$ *to the voter list $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\mathsf{id}, \mathsf{d})\}$.*

A voting system is termed registration-free if d *is set to* $\perp$*, and the registration process is considered non-interactive if it involves only a single communication from* R *to the voter.*

$\mathsf{VerifyCred}(\mathsf{d})$*: On input credential* d*, outputs* 0 *or* 1*. The algorithm outputs* 1 *if and only if the credential is valid.*

$\mathsf{Vote}(\mathsf{id}, \mathsf{d}, \mathsf{v})$*: When receiving an* id*, a credential* d*, and a vote* v*, output a ballot* b*, which is sent to the ballot box* BB*.*

$\mathsf{Valid}(\mathsf{BB}, \mathsf{b})$*: On input ballot box* BB *and a ballot* b*, outputs* 0 *or* 1*. The algorithm outputs* 1 *if and only if the ballot is valid.*

$\mathsf{Append}(\mathsf{BB}, \mathsf{b})$*: On input the ballot box* BB *and a ballot* b*, appends* b *to* BB *if* $\mathsf{Valid}(\mathsf{BB}, \mathsf{b}) = 1$*.*

$\mathsf{Publish}(\mathsf{BB})$*: On input ballot box* BB*, outputs the public view* PBB *of* BB*, which is the one that is used to verify the election.*

$\mathsf{VerifyVote}(\mathsf{PBB}, \mathsf{b})$*: On inputs the public board* PBB*, a ballot* b*, it returns* 0 *or* 1*. This algorithm is used by voters to verify whether their ballot has been properly processed and recorded.*

$\mathsf{Tally}(\mathsf{BB}, \mathsf{SK})$*: on input ballot box* BB *and private key of the election* SK*, outputs the tally* r *and a proof* Π *that the tally is correct w.r.t. the result function* ρ*.*

$\mathsf{VerifyResult}(\mathsf{PBB}, \mathsf{r}, \Pi)$*: On input public bulletin board* PBB*, result of the tally* r *and proof of the tally* Π*, outputs* 0 *or* 1*. The algorithm outputs* 1 *only if* Π *is a valid proof that* r *is the correct election result.*

The *election coordinator* CO is responsible for generating the election's keys by executing the $\mathsf{SetupElection}$ algorithm. Anyone can verify the public key PK using the $\mathsf{VerifyPK}$ function. Given the public parameters of the system, a voter V can interact with the *registrar* R to obtain a public credential via the $\mathsf{Register}$ protocol. Once the credential is validated using the $\mathsf{VerifyCred}$ function, the voter can prepare a ballot b using the Vote algorithm and send it to the ballot box BB. If the ballot is deemed valid according to the Valid function, it will be appended to the public board using the Append algorithm. Voters can confirm that their ballot has been correctly recorded on the public board by using the $\mathsf{VerifyVote}$ function. The *tallier* T runs Tally to compute the election result and produce a proof of correctness. Anyone can use these results and the contents of the PBB to verify the election outcome with $\mathsf{VerifyResult}$. The definition differs from that proposed in [8] by introducing the $\mathsf{VerifyPK}$ and $\mathsf{VerifyCred}$ algorithms to address our security model (see Section 2.2), where authorities are treated as adversaries capable of generating malicious keys.

2.2 Voter Anonymity

Existing anonymity definitions in the voting context are either too informal (e.g., [27]) or focus on different security goals (e.g., [4,21]). We present a formal game-based definition of voter anonymity, modeling the privacy of the voting system described earlier. The definition requires that even if an adversary generates a public credential for any chosen voter, it should still be unable to determine which voter cast a specific ballot by observing the public bulletin board, provided that these voters cast their ballots honestly. This condition must hold even if the adversary maliciously selects PK of the election or revoting scenarios.

Definition 2 (Voter Anonymity). *A voting system $\mathcal{VS}$ has voter anonymity if no* PPT *adversary $\mathcal{A}$ can distinguish between games $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{anon},0}(\lambda)$ and $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{anon},1}(\lambda)$ defined by the oracles in Figure 1, that is for any efficient algorithm $\mathcal{A}$:*

$$\left|\Pr\left[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{anon},0}(\lambda)=1\right]-\Pr\left[\mathsf{Exp}_{\mathcal{A}}^{\mathsf{anon},1}(\lambda)=1\right]\right|$$

is negligible in λ.

```
$\mathcal{O}\mathsf{init}(1^\lambda)$
$\mathsf{PK} \leftarrow \mathcal{A}(1^\lambda)$
if $\mathsf{VerifyPK}(\mathsf{PK}) = 0$ then return $\perp$
else return $\mathsf{PK}$
```

```
$\mathcal{O}\mathsf{register}(\mathsf{id})$
if $\mathsf{id}$ has not been queried yet:
 then $\mathsf{d} \leftarrow \langle \mathcal{A}, \mathsf{V}(\mathsf{id})\rangle$
if $\mathsf{VerifyCred}(\mathsf{d}) = 0$ then return $\perp$
else $\mathcal{CU} \leftarrow \mathcal{CU} \cup \{(\mathsf{id}, \mathsf{d})\}$
```

```
$\mathcal{O}\mathsf{voteLR}(\mathsf{id}_0, \mathsf{id}_1, \mathsf{v}_0, \mathsf{v}_1)$
if $\mathsf{v}_i \notin \mathcal{C}$ or $\nexists\ (\mathsf{id}_i, \star) \in \mathcal{CU}$ for $i = 0, 1$
 then return $\perp$
if either $\mathsf{id}_0$ or $\mathsf{id}_1$ was queried in any other pair:
 then return $\perp$
else
  $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{id}_0, \mathsf{id}_1)\}$
  $\mathsf{b}_0 = \mathsf{Vote}(\mathsf{id}_\beta, \mathsf{d}_\beta, \mathsf{v}_0)$
  $\mathsf{b}_1 = \mathsf{Vote}(\mathsf{id}_{1-\beta}, \mathsf{d}_{1-\beta}, \mathsf{v}_1)$
  $\mathsf{Append}(\mathsf{BB}_0, \mathsf{b}_0); \mathsf{Append}(\mathsf{BB}_0, \mathsf{b}_1)$
  $\mathsf{Append}(\mathsf{BB}_1, \mathsf{b}_0); \mathsf{Append}(\mathsf{BB}_1, \mathsf{b}_1)$
```

```
$\mathcal{O}\mathsf{cast}(\mathsf{b})$
if $\mathsf{Valid}(\mathsf{BB}_0, \mathsf{b}) = 0$ or $\mathsf{Valid}(\mathsf{BB}_1, \mathsf{b}) = 0$
 then return $\perp$
else $\mathsf{Append}(\mathsf{BB}_0, \mathsf{b}); \mathsf{Append}(\mathsf{BB}_1, \mathsf{b})$
```

```
$\mathcal{O}\mathsf{board}()$
return $\mathsf{Publish}(\mathsf{BB}_\beta)$
```

Fig. 1: Oracles used in the $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{anon},\beta}(\lambda)$ experiment for $\beta = 0, 1$. The adversary starts by invoking $\mathcal{O}\mathsf{init}$, followed by any sequence and number of calls to $\mathcal{O}\mathsf{register}$, $\mathcal{O}\mathsf{voteLR}$, $\mathcal{O}\mathsf{cast}$, and $\mathcal{O}\mathsf{board}$. It then outputs its guess for the bit β, winning if the guess is correct.

The game $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{anon},\beta}(\lambda)$ is parameterized by a bit β, and the adversary has access to the following oracles:

– $\mathcal{O}\mathsf{init}$: Allows the adversary $\mathcal{A}$ to initialize the voting system by generating secret and public keys for the election.
– $\mathcal{O}\mathsf{register}$: Lets $\mathcal{A}$ select a voter's id and generate a corresponding public and valid credential d for the voter. $\mathcal{A}$ obtains the view of the execution process. The voter id and its associated credential d are then added to the list of registered voters $\mathcal{CL}$.
– $\mathcal{O}\mathsf{cast}$: Allows $\mathcal{A}$ to cast a ballot b on behalf of any party. If the ballot is valid, it is placed in both ballot boxes.
– $\mathcal{O}\mathsf{voteLR}$: Takes two potential votes $(\mathsf{v}_0, \mathsf{v}_1)$ for any two voters' ids from $\mathcal{CL}$, produces ballots b_0 and b_1 for these votes and places them in both ballot boxes BB_0 and BB_1, provided that $\mathsf{v}_0, \mathsf{v}_1 \in \mathcal{C}$. In the case of revoting, the adversary may only invoke this oracle for the same pair of voter ids already present in $\mathcal{CL}$, irrespective of their order, and not for mismatched or crossed

pairs. This restriction prevents a trivial attack where $\mathcal{A}$ could exploit revoting to win the game by identifying repeated credentials on the bulletin board.
– $\mathcal{O}\mathsf{board}$: Returns BB_β, representing the view of the public bulletin board.

This game assumes that the adversary generates PK of the election, ensuring voter anonymity by preventing even colluding authorities from linking voters to their votes. This definition requires the ballot submission to remain anonymous. If this anonymity is compromised, the adversary could exploit metadata, such as network-level identifiers (e.g., IP addresses), or timing information to undermine voter anonymity. See Section 4 for feasibility.
Voter Anonymity and Ballot Privacy. Ballot privacy ensures the overall privacy of the voting system by relying on tally simulation, as demonstrated in Helios [1,3]. In contrast, voter anonymity excludes tallying and assumes two indistinguishable bulletin boards, where tally results provide no advantage to an adversary. Comparing the two security notions is challenging, as they are based on different structures. For instance, in the voter anonymity game, Helios (without including the voter's identity, as this would trivially allow the adversary to win) displays identical ballots for the same votes, providing no means for an adversary to distinguish between them. Additionally, unlike ballot privacy, which relies on talliers not prematurely decrypting ballots, voter anonymity guarantees privacy without requiring trust in talliers or other authorities.

3 The Construction based on Blind Signatures

We present a generic voting system construction, inspired by FOO [18] and Cetinkaya et al. [7], by first introducing the necessary cryptographic building blocks and then showing it satisfies voter anonymity. Detailed pseudocode is provided in Figure 3

3.1 Building Blocks

CPA **Encryption.** A CPA encryption scheme is a public-key encryption with three algorithms: $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. The key generation algorithm $\mathsf{Gen}(1^\lambda)$, given a security parameter 1^λ, outputs the decryption key dk and encryption key ek. The encryption algorithm $\mathsf{Enc}(\mathsf{ek}, m)$ takes the public key ek and a message m as input, producing a ciphertext c. The decryption algorithm $\mathsf{Dec}(\mathsf{dk}, c)$ outputs the original message m, given the secret key dk and ciphertext c.
Blind Signatures. Blind signature schemes, first introduced by Chaum [9], are interactive protocols between a signer and a user, enabling the user to obtain a signature that remains unlinkable to their identity.

Definition 3 (Blind Signature Scheme). *A blind signature scheme $\mathcal{BS}$ is a 4-tuple of polynomial-time algorithms* $(\mathsf{SGen}, \mathcal{S}, \mathcal{U}, \mathsf{SVerify})$*:*

– $\mathsf{SGen}(1^\lambda)$*: On input security parameter 1^λ, generate a pair of keys, the public (verification) key* vk *and the private (signing) key* sk*.*
– $\mathsf{Sign}\langle\mathcal{S}(\mathsf{sk}), \mathcal{U}(\mathsf{vk}, m)\rangle$*: Is an interactive protocol run by a (stateful) signer $\mathcal{S}$ with input the signing key* sk *and a (stateful) user $\mathcal{U}$ with input a message $m \in \mathcal{M}$ and the signer's public key. It outputa σ for the user and no output*

for the signer. We write this as $(\perp, \sigma) \leftarrow \mathsf{Sign}\langle \mathcal{S}(\mathsf{sk}), \mathcal{U}(\mathsf{vk}, m)\rangle$ *or simply* $(\perp, \sigma) \leftarrow \langle \mathcal{S}(\mathsf{sk}), \mathcal{U}(\mathsf{vk}, m)\rangle$.
- $\mathsf{SVerify}(\mathsf{vk}, m, \sigma)$*: Outputs* 1 *if the signature* σ *is valid w.r.t.* m *and* vk*,* 0 *otherwise.*

Correctness of a blind signature scheme requires that for any $m \in \mathcal{M}$, and for $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{SGen}(1^\lambda)$, and σ output by $\mathcal{U}$ in the Sign execution, it holds that $\mathsf{SVerify}(\mathsf{vk}, m, \sigma) = 1$ with overwhelming probability in λ.

The security of a blind signature scheme is defined by two notions: unforgeability and blindness [19, 22, 29] (see Figure 2). Unforgeability ensures that any probabilistic polynomial-time (PPT) adversary $\mathcal{U}^*$ engaging in at most k completed interactions with an honest signer $\mathcal{S}$ should not be able to output $k+1$ valid message/signatures pairs with different messages.

Definition 4 ($\mathcal{BS}$ Unforgeability). *A blind signature scheme $\mathcal{BS}$ is* unforgeable *if for any* PPT *adversary* $\mathcal{U}^*$*, the experiment* $\mathsf{Exp}^{\mathsf{uf}}_{\mathcal{BS},\mathcal{U}^*}(\lambda)$ *defined in Figure 2 (right) returns 1 with a probability negligible in* λ.

Blindness ensures that a malicious signer $\mathcal{S}^*$ cannot distinguish which of two messages m_0 and m_1, was signed first during two executions with an honest user $\mathcal{U}$. This condition must hold, even if $\mathcal{S}^*$ maliciously chooses the public key vk. Here, we consider sequential attacks, where only one signing session is active at a time. If $\mathcal{S}^*$ refuses to sign one input (i.e., $\sigma_i = \perp$), both resulting signatures are set to $\perp$, preventing any advantage from protocol disruption.

A blind signature scheme is secure if it is unforgeable and blind.

Definition 5 ($\mathcal{BS}$ Blindness). *A blind signature scheme $\mathcal{BS}$ satisfies* blindness *if for every* PPT *adversary* $\mathcal{S}^*$*, the experiment* $\mathsf{Exp}^{\mathsf{bl}}_{\mathcal{BS},\mathcal{S}^*}(\lambda)$ *defined in Figure 2 (left) returns 1 with a probability negligibly close in λ to* $\frac{1}{2}$.

```
$\mathsf{Exp}^{\mathsf{bl}}_{\mathcal{BS},\mathcal{S}^*}(\lambda)$
$(\mathsf{vk}, m_0, m_1, \mathsf{st}) \leftarrow \mathcal{S}^*(1^\lambda)$
$b \leftarrow \{0,1\}$
$(\mathsf{st}_1, \sigma_b) \leftarrow \langle \mathcal{S}^*(\mathsf{st}), \mathcal{U}(\mathsf{vk}, m_b)\rangle$ //1st session
$(\mathsf{st}_2, \sigma_{1-b}) \leftarrow \langle \mathcal{S}^*(\mathsf{st}), \mathcal{U}(\mathsf{vk}, m_{1-b})\rangle$ //2nd session
if $\sigma_0 = \perp$ or $\sigma_1 = \perp$ then $(\sigma_0, \sigma_1) \leftarrow (\perp, \perp)$
else $\sigma \leftarrow (\sigma_0, \sigma_1)$
$b' \leftarrow \mathcal{S}^*(\sigma, \mathsf{st}_1, \mathsf{st}_2)$
return $b' = b$
```

```
$\mathsf{Exp}^{\mathsf{uf}}_{\mathcal{BS},\mathcal{U}^*}(\lambda)$
$(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{SGen}(1^\lambda)$
$((m_1^*, \sigma_1^*), \ldots, (m_{k+1}^*, \sigma_{k+1}^*)) \leftarrow \mathcal{U}^{*\langle \mathcal{S}(\mathsf{sk}), \cdot \rangle^\infty}(\mathsf{vk})$
return 1 if
    $m_i^* \neq m_j^*\ \forall i, j$ with $i \neq j$, and
    $\mathsf{SVerify}(\mathsf{vk}, m_i, \sigma_i) = 1\ \forall i$, and
    at most $k$ interactions with $\mathcal{S}(\mathsf{sk})$ were completed.
```

Fig. 2: Security games of blind signatures [19, 22].

3.2 Voting Scheme

Election Setup. The protocol follows the generic voting system in Definition 1, with the election public key PK generated by both the election coordinator CO and the registrar R, as the latter acting as the signer in the $\mathcal{BS}$, responsible for issuing public credentials to voters. Both entities generate their key pairs using functions: $\mathsf{SetupElection}_{\mathsf{CO}}$ for the coordinator and $\mathsf{SetupElection}_{\mathsf{R}}$ for the registrar. The registrar generates its signing key pair using the SGen process from the $\mathcal{BS}$, while CO uses the Gen algorithm from the CPA encryption scheme. The pair $(\mathsf{ek}, \mathsf{vk})$ form the public key PK, which is published on the PBB.

```
$\mathsf{SetupElection}_{\mathsf{CO}}(1^\lambda)$
$(\mathsf{dk}, \mathsf{ek}) \leftarrow \mathsf{Gen}(1^\lambda)$
$\mathcal{L} \leftarrow \perp$
if $\mathsf{VerifyPK}(\mathsf{ek}) = 0$ then return $\perp$
else return $\mathsf{ek}$

$\mathsf{SetupElection}_{\mathsf{R}}(1^\lambda)$
$(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{SGen}(1^\lambda)$
if $\mathsf{VerifyPK}(\mathsf{vk}) = 0$ then return $\perp$
else return $\mathsf{vk}$

$\mathsf{Register}(1^\lambda)$
$(\perp, \mathsf{d_V}) \leftarrow \mathsf{Sign}\langle \mathsf{R}(\mathsf{sk}, \mathsf{id}), \mathsf{V}(\mathsf{vk}, \tau)\rangle$
$\mathcal{L} \leftarrow \mathcal{L} \cup \{\mathsf{id}\}$
return $\mathcal{L}$

$\mathsf{VerifyCred}(\tau, \mathsf{d_V})$
if $\mathsf{SVerify}(\mathsf{vk}, \tau, \mathsf{d_V}) = 0$ then return $\perp$
else return $1$

$\mathsf{Vote}(\mathsf{id}, \tau, \mathsf{d_V}, \mathsf{v})$
if $\mathsf{VerifyCred}(\tau, \mathsf{d_V}) = 0$ then return $\perp$
else $c \leftarrow \mathsf{Enc}(\mathsf{ek}, \mathsf{v})$
return $\mathsf{b} \leftarrow (\tau, \mathsf{d_V}, c)$

$\mathsf{Valid}(\mathsf{BB}, \mathsf{b})$
if $\exists \mathsf{b}' \in \mathsf{BB} : \mathsf{b}' = \mathsf{b}$ or $\mathsf{VerifyCred}(\tau, \mathsf{d_V}) = 0$
  then return $0$
else return $1$
```

Fig. 3: Instantiation of our voting scheme using a blind signature scheme.

Registration Phase. A voter V must obtain a private credential. This process begins with the generation of a token τ and interacting with the registrar R through the $\mathsf{Register}$ protocol to acquire a blind signature $\mathsf{d_V}$ on τ. During this interaction, R verifies the validity of the voter's identifier id. If the identifier is invalid, the process is terminated. Otherwise, R and V execute the Sign algorithm of the blind signature scheme. Upon completion of this interaction, V outputs a signature $\mathsf{d_V}$ on their token τ. The used id is then updated to the voter list $\mathcal{L}$ on the PBB to monitor the eligible voters who have been registered to vote. The registrar also stores the communication with each voter locally. If a voter later presents an identifier idthat has already been used to issue a credential, the stored communication will be replayed to ensure that only one credential is issued per voter.

Voting Phase. The voter then prepares a ballot b using the function Vote. Initially, the voter verifies the validity of their credential using the $\mathsf{VerifyCred}$ function, which invokes the $\mathsf{SVerify}$ algorithm to confirm that the signature $\mathsf{d_V}$ is valid with respect to the token τ and the registrar's public verification key vk. If the verification is successful, the pair $(\tau, \mathsf{d_V})$ is accepted as the voter's private credential. The voter then encrypts their vote v using the Enc algorithm of a CPA-secure encryption scheme, resulting in a ballot b. This ballot is subsequently submitted through an anonymous channel. The b is appended to the BB by the Append algorithm only if it is valid, i.e., $\mathsf{Valid}(\mathsf{BB}, \mathsf{b}) = 1$. A ballot is valid if it contains a valid credential and no identical ballot has been previously submitted. This means the system supports revoting; if the same credential reappears on the board (indicating the voter has previously cast a vote), the new ballot will still be appended. Note that after ballot submission, the voters' credentials are available on the PBB. The implementation must ensure that no one can steal a voter's credential and vote on their behalf (see Section 4).

Tallying phase. During the tallying process, only the most recent ballot associated with each credential is considered. The process begins with the verification of credential validity using $\mathsf{VerifyCred}$. Following the exclusion of invalid ballots, the tallier T runs the Tally function by decrypting each ballot and verifying that the decrypted vote lies within the designated vote space $\mathcal{C}$. Subsequently, the tallier announces the election results. To ensure universal verifiability, the tallier also publishes the decryption key dk on the public bulletin board. A voter can verify whether their ballot has been properly processed and recorded by using $\mathsf{VerifyVote}$, which allows them to decrypt the ballot corresponding to their credential. It is noteworthy that since anyone can verify the election results using dk, the role of the tallier is rendered superfluous since CO is capable of performing the same function. Thus, in the voting system the inclusion of an additional entity as the tallier is unnecessary.

Remark. Since the ballot is transmitted through an anonymous channel, the vote could technically be sent in plaintext. However, to mitigate the risk of influencing the final election outcome based on the current state of the PBB, the votes are encrypted using a CPA-secure encryption scheme.

3.3 Security of the Voting Scheme

Theorem 1. *If the blind signature scheme utilized in the proposed voting system satisfies the blindness property (Definition 5), then the voting system satisfies voter anonymity (Definition 2).*

Proof sketch. Due to space constraints, we provide only a proof sketch. The experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{anon},\beta}(\lambda)$ defines a game where an adversary $\mathcal{A}$ generates the election public key $\mathsf{PK} = (\mathsf{ek}, \mathsf{vk})$. By interacting with chosen voters via the $\mathcal{O}\mathsf{register}$ oracle, i.e., $(\mathsf{st}, \mathsf{d_V}) \leftarrow \mathsf{Sign}\langle \mathcal{A}, \mathsf{V}(\mathsf{vk}, \tau)\rangle$, $\mathcal{A}$ gains state st in the blind signing process and accesses the voter's identity id. To append ballots to BB_0 and BB_1, $\mathcal{A}$ may query the $\mathcal{O}\mathsf{cast}$ and $\mathcal{O}\mathsf{voteLR}$ oracles.

$\mathcal{O}\mathsf{cast}$ appends an identical valid ballot to both bulletin boards, regardless of whether the ballot is generated dishonestly. This does not provide the adversary with any advantage, nor does it introduce any distinguishable discrepancy between the two boards.

$\mathcal{O}\mathsf{voteLR}$ records two ballots, b_0 and b_1, which are generated honestly by two voters' ids chosen by the adversary, and adds them to both bulletin boards. As defined, if $\beta = 0$, the oracle appends $\mathsf{b}_0 = (\tau_0, \mathsf{d}_\mathsf{V}^0, c_0)$ and $\mathsf{b}_1 = (\tau_1, \mathsf{d}_\mathsf{V}^1, c_1)$ to both BB_0 and BB_1 in the exact order. Conversely, if $\beta = 1$, $\mathsf{b}_0 = (\tau_1, \mathsf{d}_\mathsf{V}^1, c_0)$ and $\mathsf{b}_1 = (\tau_0, \mathsf{d}_\mathsf{V}^0, c_1)$ are added instead, where τ_0, τ_1 are randomly chosen by the voters and blindly signed by $\mathcal{A}$, and $\mathsf{d}_\mathsf{V}^0, \mathsf{d}_\mathsf{V}^1$ are the unblinded signatures computed by the voters using their secret blinding factors.

Several observations can be made about this game. First, the encryption does not enable the adversary to distinguish between the two boards, even when the encryption keys are adversarially chosen, as the ciphertexts appear identical. Additionally, recasting provides no advantage, as it merely appends the ballots for same queried id pair to both boards. Second, if the adversary is able to distinguish between the two boards, this implies the ability to differentiate between the credential pairs $(\tau_0, \mathsf{d}_\mathsf{V}^0)$ and $(\tau_1, \mathsf{d}_\mathsf{V}^1)$. This directly reduces to the blindness property of the underlying blind signature scheme, where the adversary would be capable of identifying which message was signed first during two sequential interactions with honest voters (Figure 2).

End-to-end verifiability. Our voting scheme satisfies end-to-end verifiability, including cast-as-intended, recorded-as-cast, and tallied-as-recorded verifications [26]. Ballots can be decrypted using the election secret key dk, ensuring voters that their submission reflects their intent. Additionally, any observer can independently verify that all valid recorded votes are included in the final tally.

Remark. This protocol outlines a basic voting scheme that ensures voter anonymity through the use of a blind signature. Practical implementations, such as $\mathsf{Opn.vote}$, address further privacy considerations (e.g. eligible verifiability) as discussed in the following section.

4 Opn.vote

$\mathsf{Opn.vote}$ is an instance of the voting construction in Section 3.2 and thus inherits all security properties in Section 3.3. It employs CPA-secure ElGamal encryption [17] and the Schnorr blind signature scheme [28, 29]. To enhance security, $\mathsf{Opn.vote}$ incorporates several modifications to the generic scheme.

- *Public Bulletin Board:* Instantiated as an Ethereum blockchain.
- *Preventing Credential Theft:* To mitigate the risk of stolen voter credentials after ballot submission (e.g., frontrunning), each voter possesses a signing key pair $(\mathsf{sk}_\mathsf{V}, \mathsf{vk}_\mathsf{V})$ to sign their transactions; each transaction corresponds to a ballot submission on the blockchain. The verification key vk_V is unique to each voter and vk_V serves as the voter's Ethereum public smart wallet address. The key pair $(\mathsf{sk}_\mathsf{V}, \mathsf{vk}_\mathsf{V})$ form the voter's smart wallet.
- *Registration Phase:* The token τ is set as vk_V. After interacting with the registrar, a voter obtains a signature d_V, which is the registrar's signature on vk_V. Thus, $(\mathsf{vk}_\mathsf{V}, \mathsf{d}_\mathsf{V})$ forms the voter's private credential.
- *Voting Phase:* Each transaction must be signed by the voter using sk_V. A ballot is structured as $\mathsf{b} = (\mathsf{d}_\mathsf{V}, c)$, which is signed by sk_V. Since vk_V corresponds to the voter's Ethereum address, the Voting Smart Contract (VSC) extracts it directly from the transaction sender, eliminating the need to include it in the voting transaction.
- *Recasting Votes:* To recast a vote, a voter creates a new ballot $\mathsf{b}' = (\varnothing, c')$, where $\varnothing$ represents an empty element. After the initial vote, each voter can be authenticated through their public key pk_V, removing the need for a valid d_V in recasts.

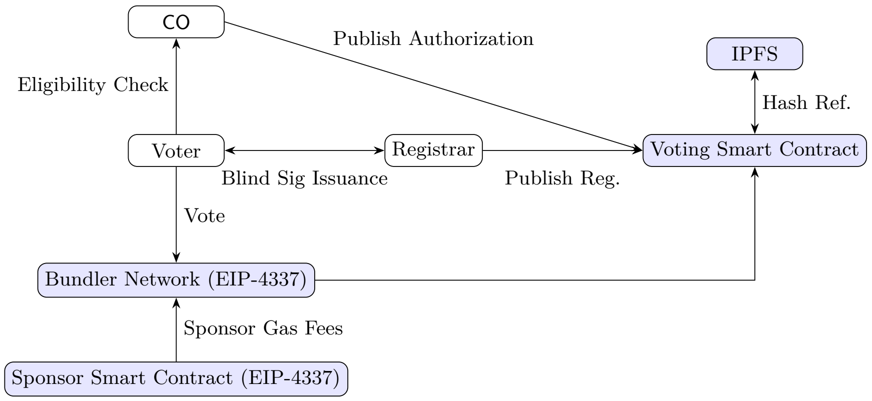

Fig. 4: High-level architecture of Opn.vote, highlighting decentralized networks and smart contracts.

4.1 System Architecture

Election Setup. The Election Coordinator CO and Registrar R configure keys within the VSC: The CO sets the encryption public key ek, and the R sets the verification key vk used to validate blind signatures.

All election data is stored decentralized. Essential short-format data are stored within the VSC. Long-format descriptions are stored on IPFS (InterPlanetary File System) [2] and referenced within the VSC, ensuring data integrity. The CO defines sponsorship rules (e.g., vote recast limits per voter) in the Sponsor Smart Contract (SSC) and funds it to cover transaction costs for voters. These rules and funding are publicly accessible within the SSC. Opn.vote introduces an eligibility check by the CO to validate voters before registration, reducing the risk of ballot stuffing through the registrar R. During the election setup, CO defines the eligibility criteria for the specific election (e.g., eID verification).

Registration Phase. The voter V completes an eligibility verification conducted by CO. After successful verification, CO confirms voters in the VSC. The voter V then interacts with R to obtain a private credential $(\mathsf{vk}_\mathsf{V}, \mathsf{d}_\mathsf{V})$ where d_V is the signature of the registrar on vk_V.

Voting Phase. After encrypting the vote choice, V creates the ballot b. Voters in Opn.vote do not need to pay for transaction fees, because it is sponsored by CO. To make this works, EIP-4331 and its decentralized Bundler Network were integrated. To submit the ballot to the VSC with sponsored transaction fees, a User Operation object is created, which includes the ballot to submit, the election ID and a reference to SSC. This operation is signed by sk_V and sent to the EIP-4331 Bundler Network.

Nodes in the Bundler Network validate the submitted operations against the SSC and its sponsorship rules, and upon successful validation, forward them to the VSC via the main blockchain network. Nodes are reimbursed by the SSC for gas fees incurred during processing. More details on the Bundler Network in 4.3. Gas sponsorship is an important feature of Opn.vote. Voters could also send voting and recast transactions directly to the main blockchain network at their own expense. VSC does not impose any limit on the number of votes or recasts.

Upon receiving a transaction, the VSC verifies whether the voter's address $\mathsf{pk_V}$ has previously submitted a valid ballot for the specific election. If a prior vote exists, the new encrypted vote replaces the previous one. If the voter has not yet voted in the specified election, the VSC checks the validity of the credential $\mathsf{d_V}$ against the registrar verification key vk and verifies that it is valid for the voter's address $\mathsf{pk_V}$. If both checks pass, the ballot is cast and stored in the VSC.
Tallying Phase. After the election deadline, CO publishes the decryption key dk and the election results. Voters can verify their ballots and the results by running the tally process locally.

4.2 Efficiency

To assess the feasibility of Opn.vote, we benchmark its gas consumption and local computational load against other established voting applications in the field.

Gas-Cost Comparison. Gas costs are transaction fees to execute operations on the Ethereum network. We evaluate two cryptographic combinations for on-chain implementation in Opn.vote. The first combination, RSA, consists of Blind RSA signatures and RSA encryption [29], while the second, ECC, uses Schnorr blind signatures and elliptic curve ElGamal encryption. We aim to compare Opn.vote with two active on-chain voting systems, MACI [25] and Vocdoni [31]. However, since Vocdoni does not use a public blockchain for voting, we restrict our comparison to MACI.

Table 1: Average gas consumption (in units) for one-time registration, voting, and recasting on Ethereum Sepolia testnet.

System	Register	Vote	Recast
MACI	248,870	436,864	436,852
Opn.vote (RSA)	394,525	396,388	92,836
Opn.vote (ECC)	161,541	176,264	47,701

Table 1 presents the average gas consumption from the registration phase (which includes storing a voter's public credential on-chain), a single voting event and recasting. The combination using ECC signatures results in lower gas consumption, primarily due to reduced storage used. This outcome is expected, as ECC signatures are significantly smaller than RSA signatures. Furthermore, Opn.vote using ECC reduces the total gas costs for one registration and one vote by $\approx 51\%$ compared to MACI. The $\approx 90\%$ reduction in gas costs for recasting results from the separation of credentials and ballots in Opn.vote, ensuring that only the ballot is updated during a recast. In contrast, MACI employs zero-knowledge proofs, which increase on-chain costs for each recast due to the computational complexity of the proofs.
Our experiments were conducted on the *Sepolia testnet.* While our real-world deployment will occur on the *Gnosis Chain Mainnet*, both networks share an Ethereum Virtual Machine (EVM) foundation, ensuring comparable gas usage. Sepolia is used for testing, whereas the Ethereum Mainnet provides

higher security against threats like double-spending and Sybil attacks [15]. The Gnosis Chain offers a balance, with lower and more predictable fees due to its use of a stable asset (xDai).

Computation Performance. We compared Opn.vote's computation time with two well-known e-voting systems, Helios [1] and Civitas [11]. Helios and Civitas rely on re-encryption mixes and shuffle proofs (measured on older hardware), while Opn.vote utilizes lightweight blind signatures and modern cryptographic libraries. As a result, Opn.vote demonstrates significant efficiency improvements, completing the preparation of 1000 ballots in 6.15 ms on an M1 MacBook Air (16 GB RAM), compared to 300 ms for Helios and 20 ms for one ballot in Civitas, as reported in their original studies. Additionally, Opn.vote tallies 500 ballots in just 635 ms, eliminating the computational overhead from shuffle proofs seen in the other systems.

4.3 Implementation of Anonymous Channel

To preserve voter privacy in Opn.vote, an anonymous ballot submission channel is required. While challenging, the decentralized architecture of public blockchains offers advantages over traditional systems

After successful registration, voters are able to export a QR code containing their private credential d_V and private signing key sk_V. The vote submission process is fully decentralized: voters only need a direct connection to the decentralized network and no interaction with Opn.vote is required. Voters can generate and submit ballots through the Opn.vote interface or use any third-party tool.

Sponsored ballots are submitted through the EIP-4337 Bundler network, a decentralized network of nodes that aggregates and processes user operations, similar to traditional blockchain node networks. The primary privacy threats are caused by nodes potentially storing metadata, such as IP addresses, device information, and timing attacks.

Timing attacks, in which an attacker links a voter's identity by monitoring votes cast immediately after registration, can be mitigated by ending the registration phase before the voting phase begins or by allowing voters to submit votes with randomized delays. The threat of breaking anonymity through metadata is minimized by the decentralized nature of the network. With multiple nodes and no default sharing of sender metadata (e.g., IP addresses), the critical factor for maintaining anonymity is the connection between the voter and the first node they connect to. While Opn.vote suggests a default trusted node for submitting ballots, voters can use the Opn.vote interface to select any node they trust. While most public blockchains currently lack privacy at the network layer, adding privacy at that layer could significantly enhance voting systems like Opn.vote. For example, a mixnet implementation would prevent the entry node from storing metadata, as an attacker would need to control multiple nodes.

5 Conclusion & Future Work

We present a cryptographic game-based definition of voter anonymity that resists collusion and malicious key generation, and we introduce Opn.vote, a publicly verifiable and decentralized voting system. It prioritizes voter anonymity, as it is critical for large-scale public elections, especially in politically sensitive environments. Opn.vote ensures voter anonymity even in scenarios where all entities collude and eliminates reliance on trusted entities or pre-existing cryptocurrency, significantly enhancing usability and accessibility, while reducing transaction costs by 51% compared to existing blockchain-based systems. Its design ensures that voter privacy is preserved without requiring a trusted setup. The system is set to be deployed in an upcoming large-scale elections, demonstrating its potential for secure, accessible, and cost-efficient online voting.

To enhance the security of Opn.vote, we plan to implement a native app with randomized submission delays, which would improve privacy against timing attacks and allow a direct connection between voters and the first node of the decentralized network. A second area of improvement is addressing the risk of collusion between the registrar R and election coordinator CO, which could lead to ballot stuffing. While the registrar can only stuff ballots up to the total number of eligible voters, and CO publishes all successful eligibility checks, collusion between these entities remains a concern. To mitigate this risk, decentralizing these roles through Blind Multi-Signatures could reduce reliance and enhance the robustness of the system.

Another concern is that a malicious CO could decrypt votes too early. A potential solution is to use ZK-proofs and multiple CO. Finally, to counter the risk of voters copying encrypted votes, including voter addresses in the encryption scheme could mitigate this issue.

References

1. Adida, B.: Helios: Web-based open-audit voting. In: Proceedings of the 17th USENIX Security Symposium. pp. 335–348. USENIX Association (2008)
2. Benet, J.: Ipfs-content addressed, versioned, p2p file system. arXiv preprint arXiv:1407.3561 (2014)
3. Bernhard, D., Pereira, O., Warinschi, B.: How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios. In: ASIACRYPT 2012. pp. 626–643. Springer (2012)
4. Bhargava, M., Palamidessi, C.: Probabilistic anonymity. In: CONCUR 2005–Concurrency Theory. pp. 171–185. Springer (2005)
5. Brightwell, I., Cucurull, J., Galindo, D., Guasch, S.: An overview of the ivote 2015 voting system. New South Wales Electoral Commission, Australia, Scytl Secure Electronic Voting, Spain pp. 1–25 (2015)
6. Buterin, V., Weiss, Y., Tirosh, D., Nacson, S., Forshtat, A., Gazso, K., Hess, T.: Erc-4337: Account abstraction using alt mempool. Tech. rep., Ethereum Improvement Proposals (2021)
7. Cetinkaya, O., Doganaksoy, A.: Pseudo-voter identity (pvid) scheme for e-voting protocols. In: The Second International Conference on Availability, Reliability and Security (ARES'07). pp. 1190–1196. IEEE (2007)
8. Chaidos, P., Cortier, V., Fuchsbauer, G., Galindo, D.: Beleniosrf: A non-interactive receipt-free electronic voting scheme. In: ACM CCS 2016. pp. 1614–1625 (2016)
9. Chaum, D.: Blind signatures for untraceable payments. In: Advances in Cryptology: Proceedings of Crypto 82. pp. 199–203. Springer (1983)
10. Cid, D.D.: A theoretical framework for understanding trust and distrust in internet voting (2022)

11. Clarkson, M.R., Chong, S., Myers, A.C.: Civitas: Toward a secure voting system. In: IEEE S&P 2008. pp. 354–368 (2008)
12. Cortier, V., Galindo, D., Küsters, R., Müller, J., Truderung, T.: Sok: Verifiability notions for e-voting protocols. In: 2016 IEEE Symposium on Security and Privacy (SP). pp. 779–798. IEEE (2016)
13. Cortier, V., Lallemand, J., Warinschi, B.: Fifty shades of ballot privacy: Privacy against a malicious board. In: 2020 IEEE 33rd Computer Security Foundations Symposium (CSF). pp. 17–32. IEEE (2020)
14. Culnane, C., Ryan, P.Y.A., Schneider, S.A., Teague, V.: vvote: A verifiable voting system. ACM Trans. Inf. Syst. Secur. **18**(1), 3:1–3:30 (2015)
15. Douceur, J.R.: The sybil attack. In: International workshop on peer-to-peer systems. pp. 251–260. Springer (2002)
16. Duenas-Cid, D.: Trust and distrust in electoral technologies: Lessons from the failure of electronic voting in the netherlands. In: Proc. 25th Int. Conf. on Digital Government Research. pp. 669–677 (2024)
17. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory **31**(4), 469–472 (1985)
18. Fujioka, A., Okamoto, T., Ohta, K.: A practical secret voting scheme for large scale elections. In: Advances in Cryptology – AUSCRYPT'92. pp. 244–251. Springer (1993)
19. Garg, S., Rao, V., Sahai, A., Schröder, D., Unruh, D.: Round optimal blind signatures. In: CRYPTO 2011. pp. 630–648. Springer (2011)
20. Heiberg, S., Parsovs, A., Willemson, J.: Log analysis of estonian internet voting 2013–2014. In: International Conference on E-Voting and Identity. pp. 19–34. Springer (2015)
21. Jonker, H., Pieters, W.: Anonymity in voting revisited. In: Towards Trustworthy Elections, New Directions in Electronic Voting. Lecture Notes in Computer Science, vol. 6000, pp. 216–230. Springer (2010)
22. Juels, A., Luby, M., Ostrovsky, R.: Security of blind digital signatures. In: Annual International Cryptology Conference. pp. 150–164. Springer (1997)
23. Kharman, A.M., Smyth, B.: Perils of current dao governance. arXiv preprint arXiv:2406.08605 (2024)
24. Kulyk, O., Volkamer, M.: Usability is not enough: Lessons learned from'human factors in security'research for verifiability. Cryptology ePrint Archive (2018)
25. MACI: Maci github page (2022), https://github.com/privacy-scaling-explorations/maci, last accessed on 2023-11-17
26. Pereira, O.: Internet voting with helios. In: Real-World Electronic Voting, pp. 293–324. Auerbach Publications (2016)
27. Pfitzmann, A., Köhntopp, M.: Anonymity, unobservability, and pseudonymity—a proposal for terminology. In: Workshop on Design Issues in Anonymity and Unobservability. pp. 1–9 (2000)
28. Pointcheval, D., Stern, J.: Provably secure blind signature schemes. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 252–265. Springer (1996)
29. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of cryptology **13**, 361–396 (2000)
30. Ryan, P.Y., Rønne, P.B., Iovino, V.: Selene: Voting with transparent verifiability and coercion-mitigation. In: Financial Cryptography and Data Security: FC 2016 Workshops, pp. 176–192. Springer (2016)
31. Vocdoni: Vocdoni documentation (2021), https://docs.vocdoni.io/architecture/general.html