Freedom Tool: Passport-based Anonymous Voting Solution

v2

December 27, 2024

Abstract

Initially, the blockchain concept introduced the reliable data and history provenance layer. Then, it scaled with the ability to compute something via a decentralized way, receiving the shared result of computation

the At the same time, the blockchain itself (any of them) knows nothing about the external world

This paper thoroughly explores advanced methods and technological infrastructures crucial for developing a reliable digital voting system. Our extensive analysis covers various key phases: creating voter profiles using biometric passport information, producing and validating eligibility credentials, and employing Verifiable Credentials for authentication. We also examine the details of setting up polls, compiling voter registries, the intricacies involved in the voting process, and the techniques used for calculating results. Moreover, we present an in-depth discussion on security claims and presumptions, aiming to guarantee the highest level of integrity and security throughout the voting procedure.

1 Introduction

In the current digital era, technological advancements have significantly enhanced the ability to develop systems that facilitate people's expression of collective opinions. This paper delves into developing a robust digital voting system, leveraging cutting-edge methodologies and technological frameworks.

We focus on building solutions that empower people to voice their opinions, enabling seamless and secure participation in the democratic process.

2 Preliminaries

Let $\mathbb{B} = \{0, 1\}$ be a binary element and $z \in \mathbb{Z}_p$ be a field element.

Let $zkHash$ be the zk-friendly hash function that $zkHash(\mathbb{B}^*) \rightarrow \mathbb{Z}_p$, $hash_n$ be a regular cryptographic one-way function that $hash_n(\mathbb{B}^*) \rightarrow \mathbb{B}^n$.

Let $\mathscr{T}_h$ be a binary Tree instance with the height h and elements $T_{i,j}$, where $i \in [0,h]$ and $j \in [0, 2^{h-i-1}-1]$. $path(T_{0,j}) \rightarrow T_{m,k}^{h-1}$ is a Merkle path for the leaf $T_{0,j}$.

Let $S(priv^*, pub^*, rel^*)$ be a statement that sets the list of mathematical relations rel^* between private $priv^*$ and public pub^* signals. Let $\mathbb{D} =< d_p, d_v >$ be a tuple of keys needed for generating and proving zk-SNARK for the statement S using the trusted procedure.

$prove(d_p, priv^*, pub^*) \rightarrow \mathbb{P}$ is a proving construction verified by the $verify(d_v, pub^*, \mathbb{P}) \rightarrow \mathbb{B}$, where 0 means the proof correctness, overwise 1.

Let $Cert_{PK} \in \mathbb{C}$ be a valid certificate that belongs to the passport issuer with the public key PK.

Let $sig(m, sk)$ be a signature function with message m and with secret key sk. The result of the function is sig value that can be verified by $sigverify(sig, PK, m) \rightarrow \mathbb{B}$, where 0 means the signature correctness, overwise 1.

3 Profile creation

Voting systems must depend on certain sources that provide information about voters' authenticity, uniqueness, and eligibility. Finding a source that verifies all these three aspects can be challenging, especially in elections in authoritarian countries. Consider state elections as an example:

- Uniqueness and realness can rely on biometric data.
- Eligibility (citizenship and age) is more difficult since the state is the only source confirming such parameters.

3.1 Passport as user's identity setup

A biometric passport serves as a tool to provide the three essential properties of voting systems. We will address concerns about passport falsification and state-issued passports later (see sections 5, 9, and 11). Here's how a biometric passport fulfills these requirements:

- It confirms citizenship, which is a key aspect of voter eligibility.
- It contains the individual's age, another factor in determining eligibility.
- The passport number acts as a unique identifier, ensuring the uniqueness of each voter.
- Information about the issuing authority is included, which helps establish the passport's authenticity.

Furthermore, the advanced features of biometric passports, such as incorporating cryptographic methods, make them a highly reliable resource for organizing elections.

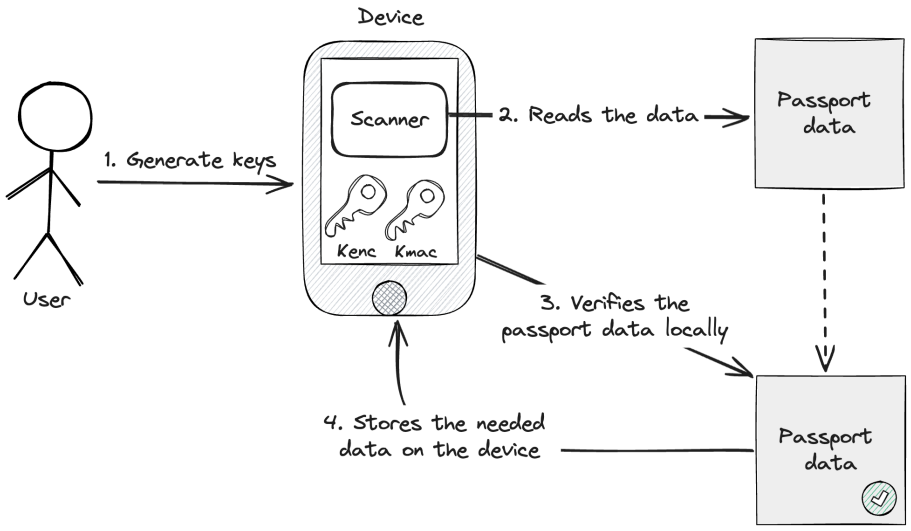

Figure 1: The process of reading a passport data.

Hereinafter, when we use the word passport, we refer to biometric passports implementing the Doc 9303 Machine Readable Travel Documents standard[INT21].

3.2 Reading and decrypting the data from the passport

To create a profile, the user must submit or 'read' the passport information. To do this, the user can use their mobile phone with an RFID (radio-frequency identification) reader (hereinafter, we will use "device"). The reading process consists of the following steps:

1. Receive the basic passport data from MRZ (Machine Readable Zone) and generate authentication keys.
2. Scan the data from the passport in encrypted form.
3. Decrypt and verify it locally.
4. Store the relevant passport data locally on the device.

For generating authentication keys and receiving data from the passport chip:

1. The device scans the passport's MRZ using the camera.
2. The device derives Document Basic Access Keys (K_{ENC} and K_{MAC}).
3. The device and the passport's chip authenticate keys to each other and derive KS_{ENC} and KS_{MAC} session keys[INT21].

These keys authenticate and encrypt messages between the device and the passport. The device reads the data from the chip and verifies it locally:

1. The device reads the Document Security Object (SOD) S_{OD} and receives the Document Signer Certificate (CDS) $C_{DS} \in \mathbb{C}$ - public key certificate of the party that issued and validated the passport.

2. The device builds and validates the certification path $path_{cert}(C_{DS})$ from the Trust Anchor (Certificate Authority) to C_{DS} by verifying the signature $sig_{C_{DS}}$ that covers S_{OD}. Very often the certificate path length is equal to 1, but could be different.
3. The device reads Data Groups DG_i, $i \in [1,16]$, from the passport.
4. The device verifies that $\forall\ DG_i$ are authentic and not modified by $\forall\ hash(DG_i) = S_{OD}.DGhash_i \wedge sigverify(sig_{C_{DS}}, C_{DS}, S_{OD}) \rightarrow 0$ with corresponding hash values in the S_{OD}.

If all verifications are performed correctly - the passport is valid, so the device stores the following list of data:

1. DG_1: Personal Details. This includes the primary identity information of the passport holder, such as name, date of birth, nationality, and passport number. It reflects much of the information printed on the passport's data page.
2. DG_2: Facial photograph. This contains the portrait of the passport holder. In the future, it will be possible to extend the protection method based on it with face recognition and ZKML proofs.
3. DG_7: Signature/Image of Holder. This data group stores the scanned image of the passport holder's handmade signature.
4. DG_{15}: Active Authentication Public Key. This includes the public key used for active authentication, a security feature to prevent unauthorized copying of passport data.
5. Hash values of other DG_i.
6. Signature of the Document that covers *SOD* (hash values $\forall\ DG_i$).
7. Document Signer Certificate (*CDS*)

Let's note that in the future (or for some countries right now), the list of biometric data groups can be extended; DG_2 and DG_7 (the portrait and the photo of a physical signature) are currently the most supportable biometrics in passports.

All actions within this process are performed locally, without access to the Internet (the certificate path can also be verified on the device if the application stores the set of valid certificates of Trust Anchors[INT23, INT24]). No personal data is shared anywhere or accessible to outside parties.

Note. We have named this process "profile entry" because registration, login, and recovery processes are absent. Users can enter their unique profile using the biometric passport and only by reading its data.

3.3 Creating the keys for digital identity

Now, the user should generate the keypair (sk_{ID}, PK_{ID}) for identity management. The proof of the user's eligibility will be connected to the created identity, and keys will be used to confirm the user's actions on smart contracts. You can imagine it as a grant issued by the user and delegating their representation rights on the blockchain layer (because the blockchain realizes the different cryptography).

For the mentioned purposes (and man-in-the-middle protection), the user should confirm

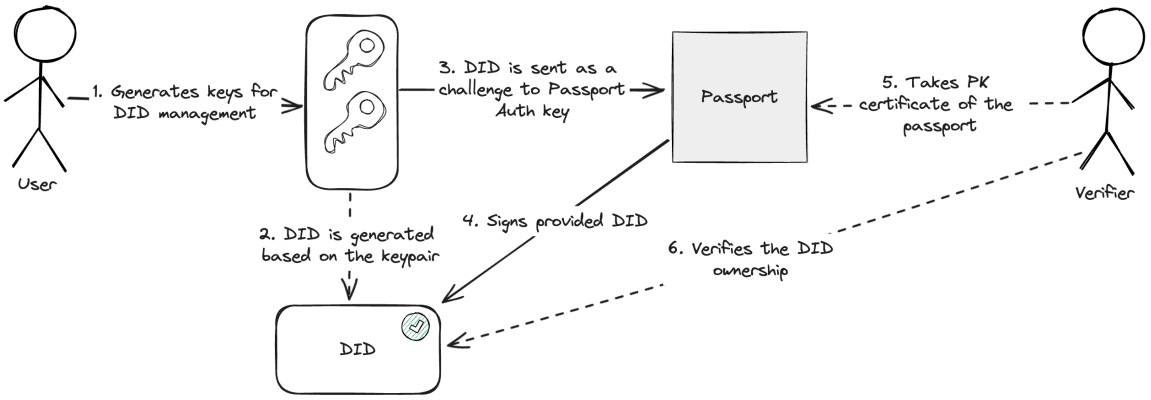

Figure 2: The signing of DID keys with the passport.

that they generate and control the defined key. Again, they use a passport or a rather active authentication flow to make that possible.

This flow consists of generating a challenge (a fixed-length string) and signing it with a private key stored in a secure segment of the passport. The corresponding public key is located in DG_{15}. Instead of a challenge, the user provides the generated DID and asks to sign it. You can think of this as a self-signed PK certificate signed using the passport's secret.

Thus, when transferring DID to an external consumer (identity provider or smart contract), the verifier will be sure that the user controls the keys to identity by $sigverify(sig, DG_{15}, DID)$. DID is generated based on the Iden3 standard[Ide24].

4 Generating the proof of eligibility

Once a local voting profile has been created, it must be tied to publicly verifiable credentials, which on-chain contracts can verify without disclosing personal data. We define eligibility verification as follows:

1. Verify that the passport was issued by one of the authorized authorities by $sigverify(sig_{C_{DS}}, C_{DS}, S_{OD})$.
2. Check that the DG_1 group includes a field with a specific citizenship $DG_1.Cz \in \mathbb{CZ}$.
3. Check that the DG_1 group includes a field with a date of birth that meets the necessary criteria $DG_1.DoB \geq t_1$.
4. Check that the passport is unexpired $DG_1.Exp < t_2$.
5. Verify that DG_1, which holds personal data, and the DG_i, $i \in [2,16]$ are included in the SOD.
6. Verify that DG_{15} is included in the S_{OD} and that the user has used the correct private key to sign the DID (passport control) by $sigverify(sig_{user}, PK_{pass}, DID)$.

Potentially, these verification checks could be executed using on-chain contracts. However, this approach would mean that some of the data would need to be made public, and generating the necessary proofs for this process can be quite challenging for user devices. Alternatively,

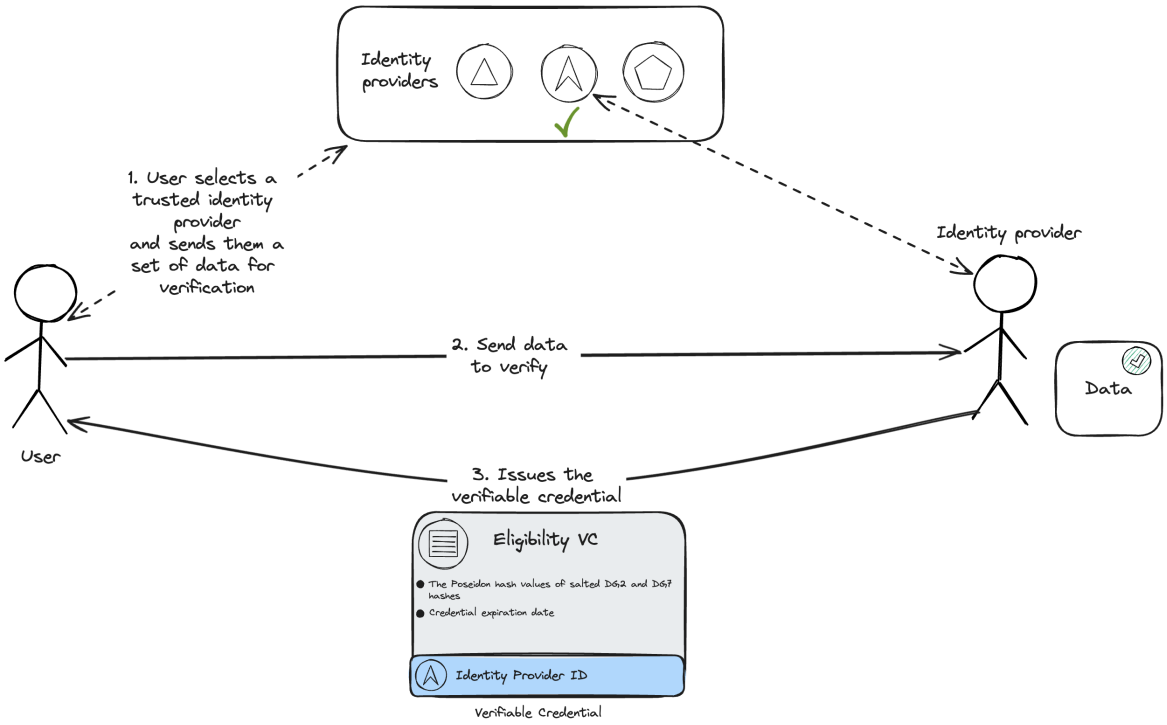

Figure 3: The process of proving the eligibility.

users should engage with the system's identity provider(s) to complete the required verification process. Importantly, this should be done in a way that doesn't involve disclosing any sensitive, personal, or traceable information.

To do this, the user selects one of the providers and sends them the following set of data:

- *DID* + signature sig_{pass} generated by the passport secret key.
- Authentication public key (DG_{15}).
- S_{OD}
- Proof of eligibility:
  - The zero-knowledge proof of citizenship is included in DG_1.
  - The zero-knowledge proof that the user is older than 18 (or will be at some point in time, depending on the voting parameters).
  - The zero-knowledge proof that the expiration date in DG_1 isn't met:
    * User can reveal the passport's expiration date to the identity provider to receive the verifiable credential with the same expiration date.
    * If the user doesn't want to reveal the passport expiration date - they can prove that the expiration time is lower than a particular value set by the identity provider. As a result - the user will receive the verifiable credential with the equivalent expiration date.
- The signature of the *SOD* and Document Signer Certificate (*CDS*).

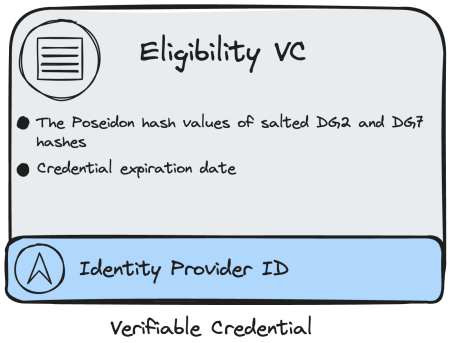

Figure 4: Verifiable credential.

5 Receiving the Verifiable Credential

Identity Provider starts the verification procedure after receiving the aforementioned dataset. If all checks (discussed above) were performed correctly - the Identity Provider issues the Verifiable Credential (VC) based on the W3C standards to the user[W3C24]. VC consists of:

- The blinded Poseidon hash values of DG_2 and DG_7 hashes (with the salt generated by the identity provider).
  - The identity provider stores the salt value, but the committed value (salt hash value) must be public and published before $\mathbb{VC}$ are issued.
  - If the salt value is revealed, the external auditor will be able to recognize which users are registered in the pool.
  - If the identity provider loses the salt value, it cannot prove the eligibility of all issued credentials.
- Credential expiration date.

This data is stored on the user's device. At the same time, the identity provider stores:

- SOD data.
- Poseidon hash values of DG_2 and DG_7 hashes.
- Salt value.
- The signature of SOD is generated by the certificate authority.
- Document Signer Certificate (CDS).
- Proof of eligibility.

If state manipulation occurs, this data can be used to prove it. The attack vectors and protection methods are discussed below (sections 7 and 8).

As a basic platform for Verifiable Credential issuance, we propose using the Rarimo protocol[Rar24]. This protocol allows VCs to be issued with the ability to transfer them to the needed blockchain later. It's a fundamental property because the voting organizer cannot know where the final voting process will be performed but wants to allow users to be registered in advance. Rarimo allows the transfer of the global identity state by one cross-chain message but not transferring of each VC separately. At the same time, the process is completely decentralized - only the needed quorum of Rarimo validators (threshold) must sign the message to confirm it on the

destination network.

The mentioned approach allows for flexibly building voting pools by separating different processes and receiving the maximum value from combining technologies and networks (some can provide unique opportunities for organizing the voting procedure exactly on their technical stack - like grants, etc.).

6 Pool creation

In this section, we delve into creating a pool instance, a series of contracts enabling the entire voting process. This includes:

- Setting up the relayer reward pools.
- Registration in the voting registry.
- Facilitating anonymous voting.
- Result calculation.

6.1 Components of the voting contracts infrastructure

Using the default infrastructure approach (described in the paper), anyone can create a pool (the particular realization can differ). The creator must call the pool factory method and pass the necessary parameters. Calling the method initiates the creation of 3 contract systems:

- Contract for investment in voting ($\mathbb{INV}$). It is needed to collect funds to compensate for the expenses of relayer nodes (see section 6).
- Contract for registering voters in the pool ($\mathbb{REG}$). It is needed to verify user ownership of the necessary $\mathbb{VC}$ and add an anonymous entity to the register of voters.
- Voting Contract ($\mathbb{VOT}$). It is needed to send anonymous votes and count voting results.

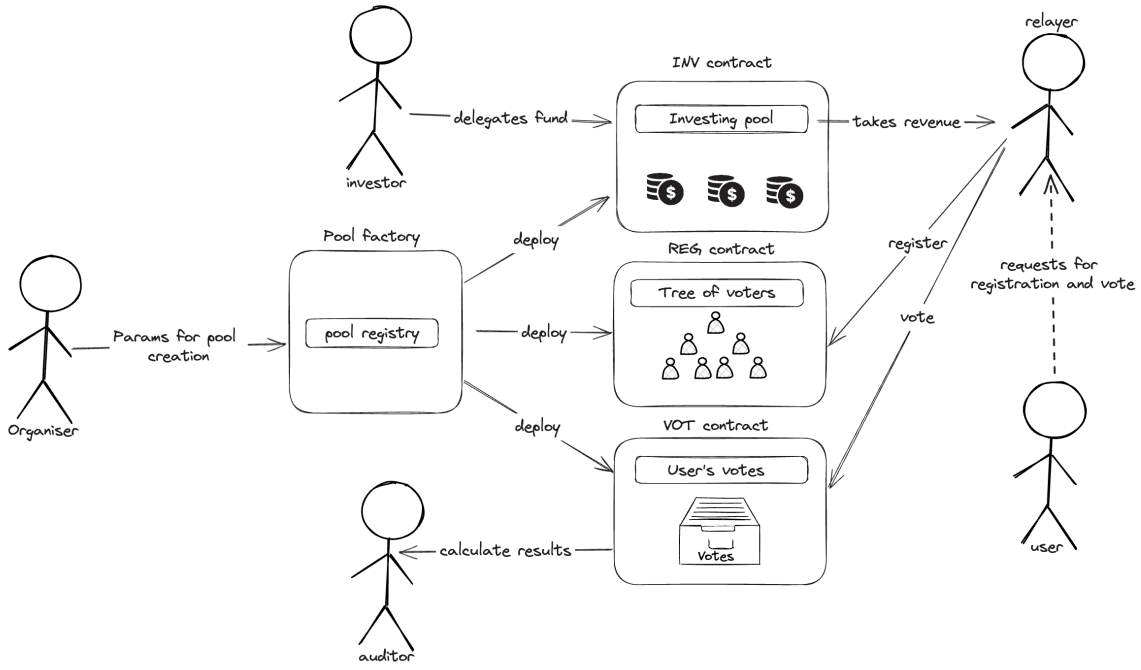

Figure 5: The infrastructure of voting contracts.

Note. The general approach to contract infrastructure is presented above. Some realizations can be different (for example, it can be one contract with different methods for described functionality).

6.2 Voting parameters

To initiate the creation of contracts, you need to define the following parameters:

1. The time frame for the functioning of each of the contracts. It is essential to understand that contracts are involved in various voting stages and executed in the sequence INV-¿REG-¿VOT-¿INV. At the end of each stage, a condition may block the contract from participating in the next stage.
2. The total number of voters (maximum) and minimal acceptable threshold - the minimum threshold of the total number of voters; upon reaching the number of registrations, a decision will be made (automatically) to vote. The minimum number of voters required to decide is also separately determined (the number of registered parties may exceed the number of actual voters).
3. The estimated average cost of transactions interacting with contracts: registration in the pool and voting.
4. Relayers reward rate. This parameter can also be adjusted depending on the amount of investment in the INV and the number of voters.
5. Voting and other options (including multiple choice options).
6. The list of trusted Identity Providers who can issue VCs to voters and confirm their eligibility.

7. The needed VC types (schemas).

6.3 Contracts initiating flow

The crowdfunding campaign's soft cap and hard cap values in the INV (Investment) contract are determined based on predefined parameters. When the campaign launch date arrives, investors can deposit their funds into the INV contract using a specific method. These funds are then automatically allocated among various pools according to the ratio of transaction costs.

If the soft cap is not met by the campaign's end, the REG (Registration) contract is not deployed, and investors (campaign funder) receive their funds back. However, if the softcap is reached (or the hardcap is met earlier), anyone can trigger the creation of the REG contract.

From the registration start date, users (directly or via relayers) can use a method to add themselves to the voting list anonymously. During this process, they set secret parameters that later verify their inclusion on the list. The outcome of this stage is an anonymous list of registered voters. If the number of registered users falls below a minimum threshold, the actual voting contract (VOT) deployment is halted to prevent potential de-anonymization risks. However, based on the number of confirmed transactions, relayers may still claim refunds from the INV contract.

The VOT contract is deployed if the minimum threshold of registered users is met (as we mentioned before, any user can initiate this process only if requirements are met). Through this contract, users can vote with proof of registration. The VOT contract also tallies the final vote count. There's an option to keep the results undisclosed until the voting process, which requires a trusted setup for encrypting votes with a threshold key, but this is optional.

6.4 How does the user find the required pool?

To provide the user with information about the voting pool, it is enough to provide the address of the master contract and the pool identifier (for example, in the form of a QR code). The voting application must support the correct interface for communication with appropriate contract methods.

7 Relayers

In conditions where we want to combine the usage of decentralized systems and provide voting rights to every eligible person, there is a list of challenges we need to solve:

1. Users must perform auditable and protected actions that are timestamped and can not be deleted or modified. It's possible with public blockchains (using transactions with appropriate smart contract calls).

2. Users don't want (or can't) to pay a transaction fee. Moreover, some users (the majority) don't know about blockchain - installing crypto wallets and buying the native currency is an additional barrier for them.
3. Some services could send transactions instead of users (they can't modify signatures or proof). Setting up the single relayer service is a possible but unreliable solution (a high probability of DoS).
4. Allowing any interested party to relay users' transactions is possible. But obviously, these relayers don't want to pay for someone else if it's not profitable.

While decentralized relayers seem promising, a critical question arises: can we build a scheme where proxy services receive some rewards or benefits when they confirm users' transactions?

At the same time, some organizations (we will call them investors below) could provide grants under the following conditions:

- Investors should be keenly interested in being engaged in the activities by end users, especially in processes that attract a large audience via voting.
- They need to ensure that the funds they provide are allocated fairly and in proportion to the involvement of relayers.
- Additionally, investors should be able to verify that relayers are rewarded only for performing appropriate actions, such as confirming valid transactions that activate specific methods of a smart contract, and this verification should occur only after these actions are confirmed.

Investors, functioning as crowdfunders in this scenario, are prepared to offer financial backing with the anticipation of facilitating a fair voting event. To meet this challenge, it's crucial to develop a solution that benefits all parties involved - users, proxy services, and investors - ensuring that each participant derives value from their involvement in this interconnected cycle.

Let's see how it works as a component of our voting tool.

7.1 Relayers' incentivizing

Pre-requirements:

- There is a voting event where we want a lot of voters (not only web3).
- The voting contract exists on the public blockchain, and each transaction with the vote requires a fee payment.
- We know the approximate number of users - "EXPECTED NUM VOTERS".
- We know the average vote price - "AVG VOTE PRICE".
- In the case of big elections, it can affect the total performance of the particular network and increase the average vote price (by competing for a place in the block).
- Additionally, some multiplier could be set - it displays the profitability of confirming the particular vote by the relayer "INCENTIVE COEFF".

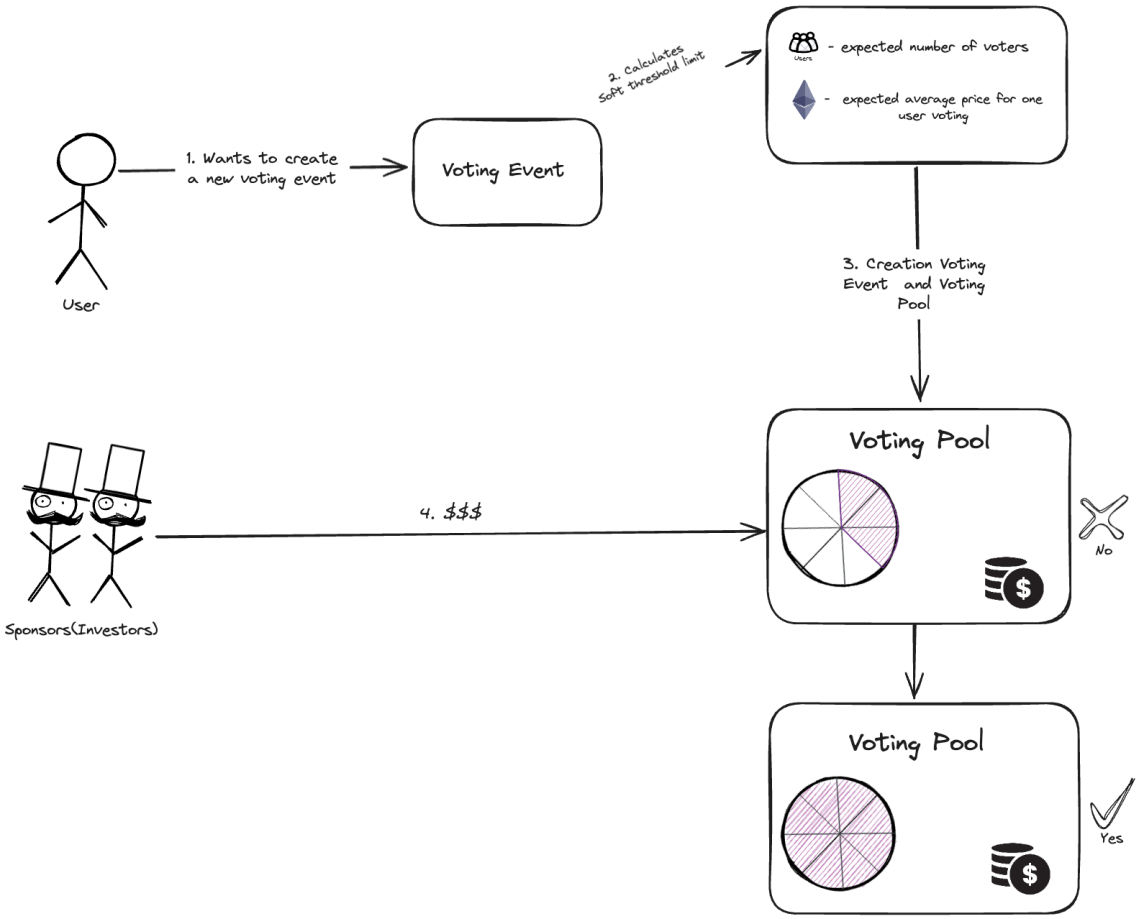

Figure 6: Investing funds to the contract infrastructure.

The solution for relayers works the following way:

1. The process starts with creating the rewards pool, which will be distributed among relayers. The following parameters are used for the pool creation:
   (a) Date and time where voting starts (or the date and time of registration procedure if it exists)
   (b) Softcap = EXPECTED NUM VOTERS * AVG VOTE PRICE * INCENTIVE COEFF
      i. The minimum amount of funds needed to start the voting process.
   (c) Hardcap (if needed)
      i. The maximum amount of funds needed to start the voting process.
   (d) Voting contract parameters (contract address, methods' names, etc).
2. Then, investors can donate funds into the pool if they agree to all defined data. The donation process looks like a crowd-funding campaign: voting starts only if the soft cap is reached. Funds are returned to investors' accounts if the soft cap isn't reached during the voting procedure.
3. When the voting process starts, users generate valid votes (they can be anonymous) with signatures/proofs and send them to relayers. They can send votes to predefined relayers they trust or send the proof to the decentralized storage network, where any

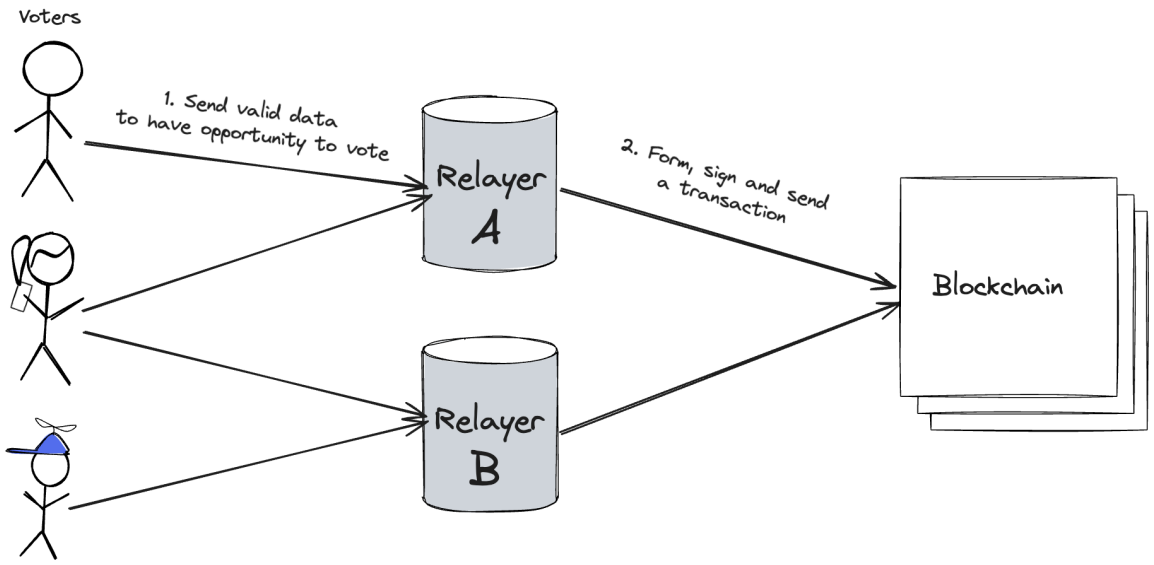

Figure 7: Relayers send transactions to the network.

relayer can take it and finally submit it to the blockchain.

4. In these conditions, it's not profitable for relayers to wait until they can receive MEV (maximal extractable value) based on the cost of transaction confirmation if the current fee is lower than the reward they receive from the investing pool (this amount is clear and visible to everyone). Any node that sends the transaction - will receive the profit.
   (a) Relayers can compete with each other - it's OK. The most important thing is the existence of at least one node that can extract the value from the confirmed transaction.
   (b) Theoretically, relayers can agree, confirm only one transaction, and divide the revenue between all relayers (without paying fees). Still, they can't trust each other that someone will not confirm other transactions and not increase their share in the pool.
   (c) During the voting, submitting transactions for relayers becomes unprofitable as the total pool rewards are lower than the total gas cost of transactions. Proxies might continue submitting transactions to get bigger shares and decrease losses. But it's better to have a basic INCENTIVE COEFF that is reasonable and calculated the proper way.
   (d) Each proxy creates a strategy for how and when to send transactions to extract maximum value. But it's better not to wait - any confirmed transaction increases the share in the pool and potentially leads to higher rewards.
5. After the voting process has ended, the reward pool is shared among all accounts that have submitted transactions during the voting procedure. The share size depends on the amount of submitted transactions.
   (a) Each relayer must initiate the rewards claim, but the contract can automatically calculate which EOAs submitted needed transactions.

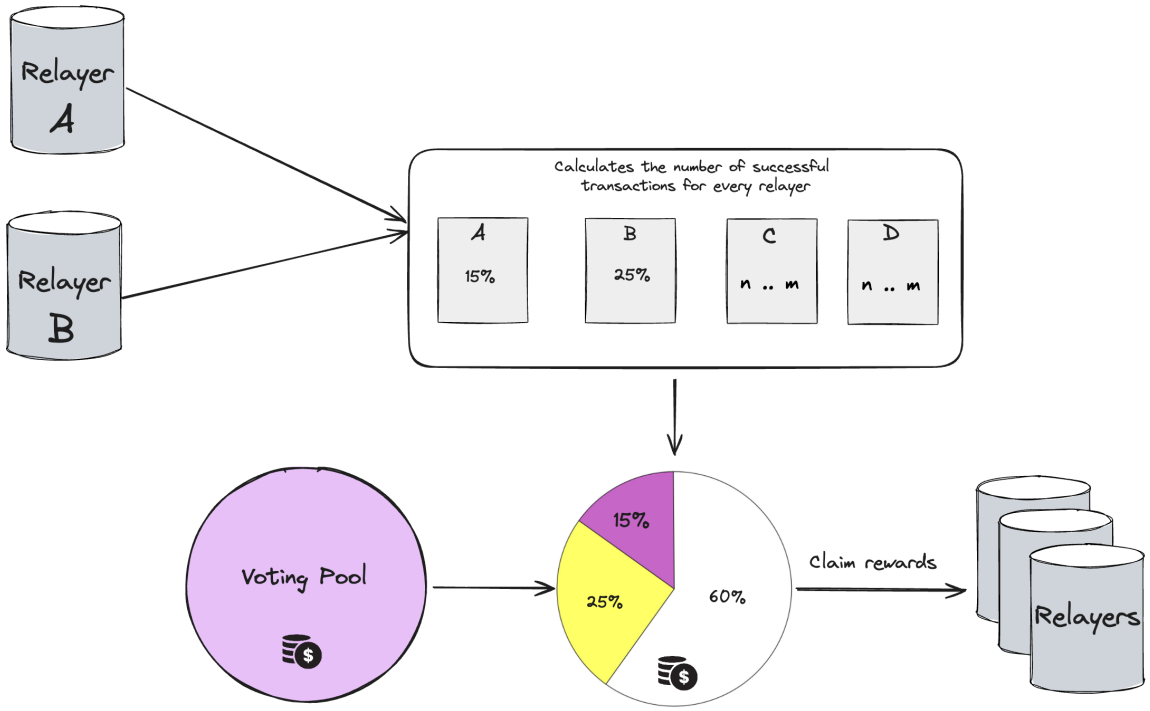

Figure 8: Relayers receive rewards based on sent transactions.

Also, there is a general description of how the web3 operations can be refunded. But sometimes, the amount of transactions within some process is more significant than one. For example, the described solution requires the following:

- Transaction for creation VC (potentially could be paid by identity providers).
- Transaction for registration into the voting pool (should be sent by relayers).
- Transaction with a vote (also should be sent by relayers).

The cost of transactions on steps is different, so dividing the pool into two sub-pools with the relation depending on the transaction types sent by relayers makes sense.

In any way, the described approach isn't a pill that replaces interaction between users and blockchains. It only opens doors for using blockchain capabilities by users who don't use wallets and aren't too much in the web3 area. Interestingly, each user can act as a relayer if they wish - in this case, they will also be refunded for their transactions.

The relayers method strongly answers the difficulties faced in on-chain decentralized voting. It incorporates crowdfunding from investors to overcome user hesitation in covering transaction expenses. This approach establishes a minimum limit for voters, sets a flexible target for crowdfunding, and guarantees that the voting only starts when there is enough public backing. During the vote, participants send their transactions through intermediaries. The reward pool formed as a result is then shared according to the volume of transactions. This strategy effectively addresses transaction cost concerns while promoting greater involvement and innovation. It offers an attractive option for managing on-chain decentralized voting within the blockchain sphere.

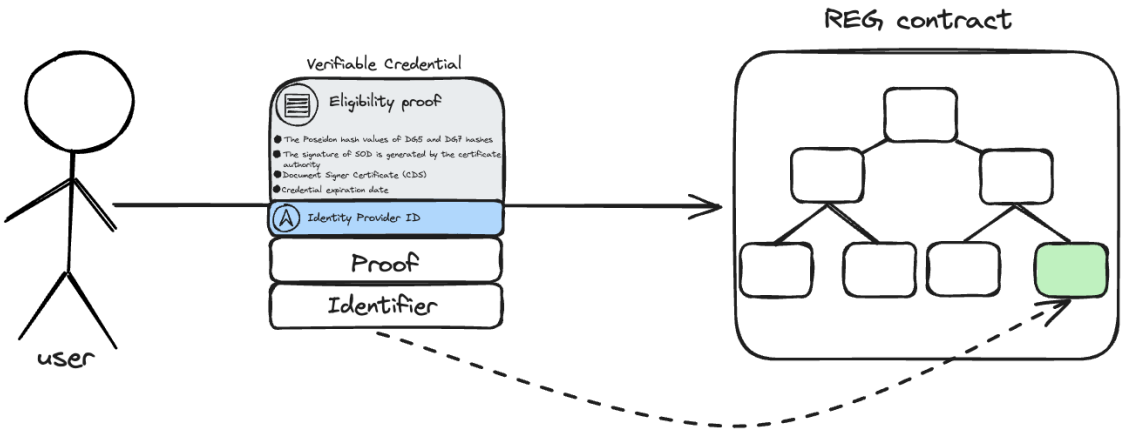

Figure 9: The process of registering the user before the voting.

8 Registration as a voter in the pool

This procedure is intended to anonymize user votes subsequently - ultimately, no one can link adding transactions to the voting list with transactions with votes[Vit23]. To add a user to the registry, follow these steps:

1. The user generates two private values: $Nullifier$ and $Secret$.
2. The user calculates$Identifier = Poseidon(Nullifier, Secret)$.
3. The user sends proof they are eligible to vote (valid and not expired Verifiable Credential) and the list of the following data:
   (a) The salted Poseidon hash values of $DG2$ and $DG7$ hashes. This data is required to check the user's uniqueness and can be used as fraud-proof if somebody wants to corrupt the user's vote.
   (b) The proof that mentioned hash values are included in VC.
4. At the same time, user provides an Identifier, which is added to the Merkle Tree list (if all verifications are passed).
   (a) The salted Poseidon hash values of $DG2$ and $DG7$ hashes are stored in the contract and don't allow the user to be registered the second time.
5. The Merkle Tree is being updated.

At this point, it's important to highlight that the user must securely store the Nullifier and Secret values on their device. These values enable the user to vote in the final stage. It's also crucial to note that once a leaf has been added to the tree, the user can erase all data related to their passport. Furthermore, for the final stage of voting, the user can even use a different device that never had access to the passport data. The only requirement for this final step is the knowledge of the Nullifier and Secret values.

9 Validity proofs of passport correctness

Some auditors may attempt to discredit elections by making claims regarding the issuance of invalid VCs by the identity provider (for example, the government claims that the identity provider created new souls or issued credentials to users who do not have the right to vote). In this case, there is a procedure for proving by the identity provider that a passport verification procedure was carried out for a certain list of unique identifiers in VCs.

The proof process is as follows:

1. The auditor makes a statement that the identity provider has violated protocol rules.
2. The auditor randomly selects a set of unique identifiers stored in the voting register. Identifier positions may be selected in any manner at the auditor's discretion.
   (a) The auditor can request evidence for all identifiers, thereby providing a 100% guarantee of the validity of all voters. However, the procedure for generating evidence in this case will be very labor-intensive, and the identity provider can determine the "probability threshold" for the test.
3. The auditor passes an array of identifiers to the identity provider.
4. For each of the identifiers, the identity provider generates a proof:
   (a) Public inputs: zkhash(salt —— hash (DG2)), CDS, hash(salt).
   (b) Private inputs: SOD signature, SOD, salt, hash(DG2).
   (c) Proofs:
      i. Proof of eligibility (generated by the user)
      ii. Proof that the hash of the salt == hash(salt)
      iii. Proof that the hash of DG2 is included in the SOD and zkhash(salt —— hash (DG2)) == unique identifier
      iv. Proof that signature (CDS, SOD) is correct.
5. The auditor checks the body of evidence:
   (a) If at least one of the proofs is not formed correctly, the identity provider fails the check (successful discreditation).
   (b) If all the evidence is correct, the identity provider did not violate the rules of the protocol, and all voters are valid.

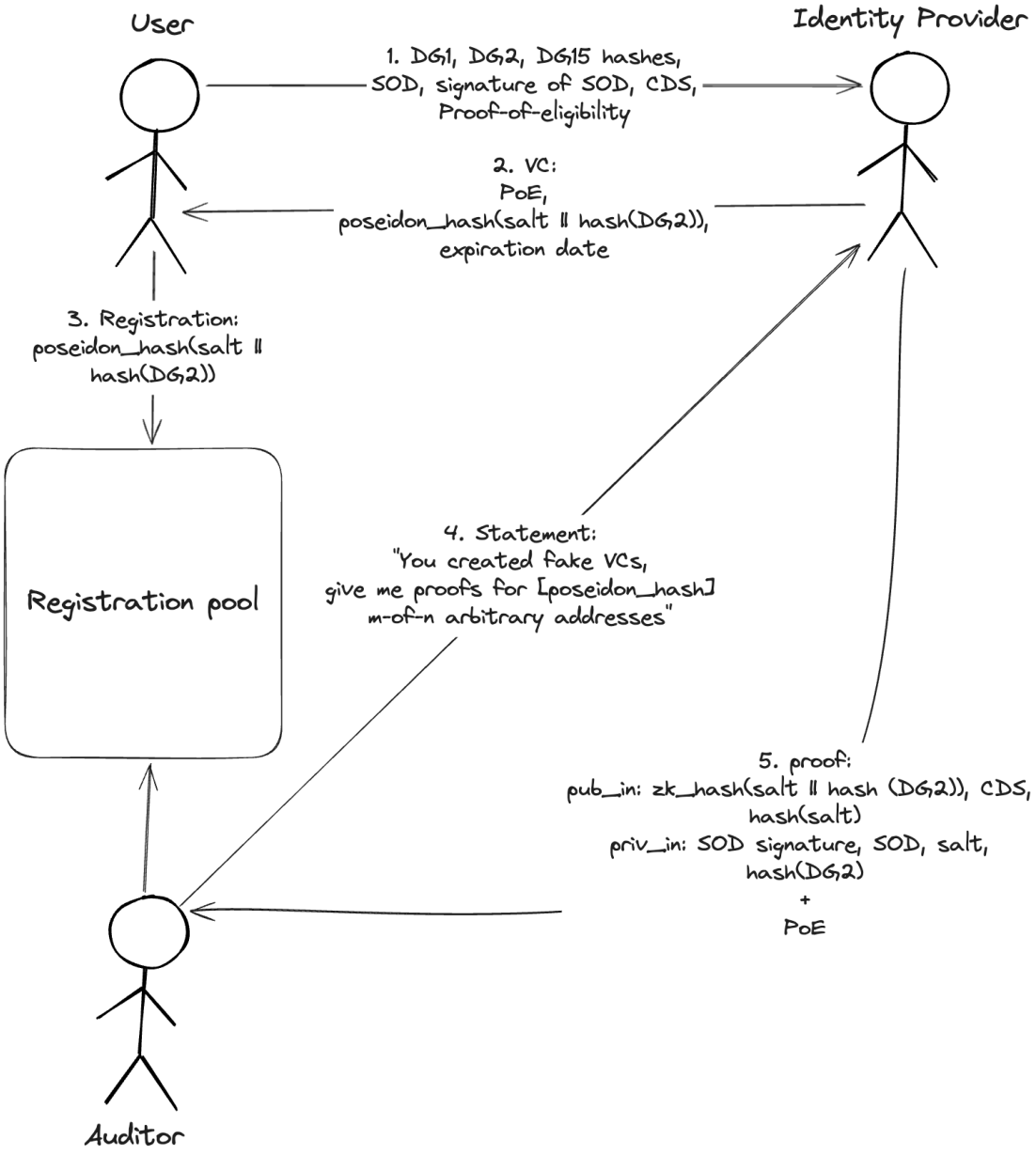

Figure 10: The proof of claims validity.

10 Voting process

To send the vote, the user needs to perform the following actions:

1. To form the ballot body representing the selected voting option (Vote).
2. Calculate the Merkle branch for the leaf added due to user registration.
3. Calculate zero-knowledge proof:
   (a) Nullifier, Merkle Root, and Vote are public inputs.
   (b) Identifier, Merkle Branch, and Secret are private inputs.
   (c) Verification logic is:
      i. Identifier == Poseidon(Nullifier, Secret);
      ii. Identifier inclusion into the Merkle Tree with the corresponding Root using the Merkle Branch.
4. Aggregates the proof along with the body of the ballot and sends it to one of the relayers. The relayer packages the proof ballot into a transaction and sends it to the network.
5. The VOT contract verifies the proof and takes into account the vote if it is correct.

Note: the Vote parameter does not participate directly in the verification process. It adds a constraint and binds the Vote to the proof. Changing the vote will make the whole proof invalid. Otherwise, malicious parties could intercept a proof and replace a Vote with a new one.

10.1 Calculation of results

The corresponding smart contract state stores the voting results. Suppose the permissionless blockchain is used as an infrastructure for pool deployment. In that case, it means that anyone can check it and be sure that the final result corresponds to all performed transactions.

There is a mode that allows hiding voting results before it finishes. This approach requires a trusted setup between the quorum of participants. This quorum forms the public key for encrypting votes by the end users. When the voting procedure is finished, these parties can reveal their secrets, and anyone (if the threshold is met) can decrypt votes and calculate the final voting result.

If results are open constantly, the additional logic can be built based on the final state of the voting contract, for example, launching some action. An encryption option is also possible but requires a more complex solution, like homomorphic encryption.

11 Security claims and assumptions

The protocol includes a list of security claims and assumptions:

- Only the person who physically owns a biometric passport can confirm the ownership

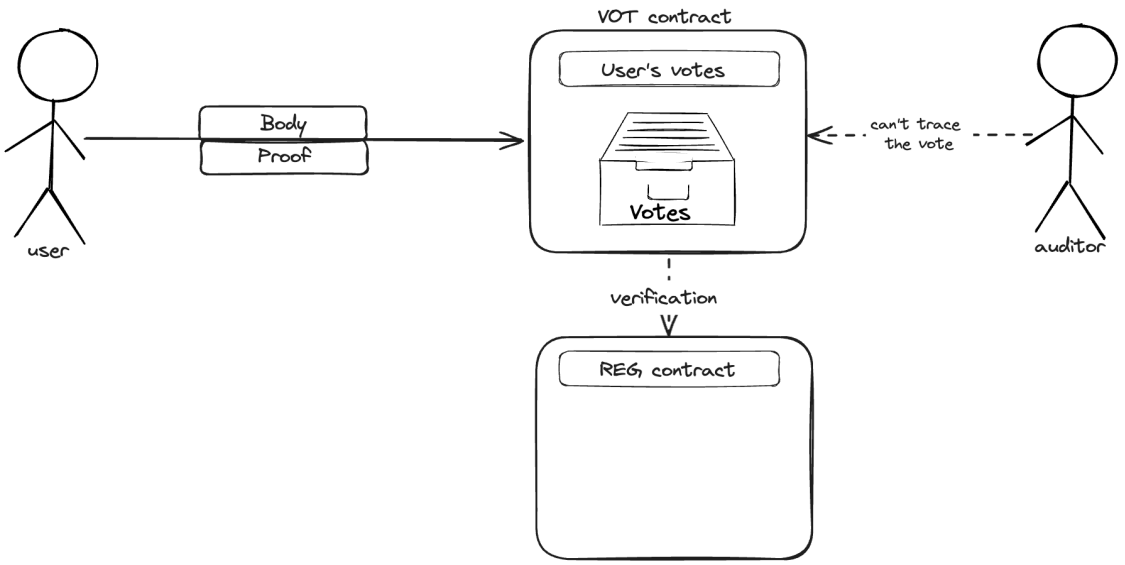

Figure 11: The voting process.

of DID.

- Only registered users on the REG contract (with appropriate tree leaves) can participate in the voting.
- Registering and voting can be performed on separate devices (through exporting Nullifier and Secret). In this case, the terminal for voting knows nothing about the person except they have a right to vote.
- Users can't vote twice using the same leaf in the tree.
- Users can't be registered in the pool using the same salted hashes of any biometry credential or with the same signature from the certificate authority.
- If the authentication key was replaced in the passport duplicate - the real owner can prove it and discredit the certificate authority.
- The vote can be sent only with the knowledge of the corresponding Nullifier and Secret. If the user lost it - there is no way to recover the vote.
- The vote can't be sent if the Nullifier or Secret is unknown.
- It's impossible to replace the ballot body in the vote while keeping the consistency of the proof.
- There is no way to track a connection between registration and voting if different EOAs / relayers were used.
- Identity providers can not recover users' personal data without having access to appropriate databases (dictionary attack).
- Identity providers can be compromised only when they fake VCs (or lose the salt value).
- The probability of identity provider correctness is calculated as follows:

Formula

References

[Ide24] Iden3. Iden3 standard, 2024. Accessed on 03.06.2024.

[INT21] INTERNATIONAL CIVIL AVIATION ORGANIZATION. Doc 9303, machine readable travel documents part 3 — specifications common to all mrtds, 2021. Accessed on 03.06.2024.

[INT23] INTERNATIONAL CIVIL AVIATION ORGANIZATION. The icao master list and icao health master list, 2023. Accessed on 03.06.2024.

[INT24] INTERNATIONAL CIVIL AVIATION ORGANIZATION. The icao pkd data, 2024. Accessed on 03.06.2024.

[Rar24] Rarimo. Rarimo, 2024. Accessed on 03.06.2024.

[Vit23] Matthias Nadler Fabian Schär Ameen Soleimani Vitalik Buterin, Jacob Illum. Blockchain privacy and regulatory compliance: Towards a practical equilibrium. *SSRN*, 2023.

[W3C24] W3C. World wide web consortium, 2024. Accessed on 03.06.2024.