IMOK: A compact connector for non-prohibition proofs to privacy-preserving applications

Oleksandr Kurbatov[1], Lasha Antadze[2], Ameen Soleimani[3],
Kyrylo Riabov[1], Artem Sdobnov[1]
[1]Distributed Lab
[2]Rarilabs
[3]0xbow

October 2024

Abstract

This article proposes an extension for privacy-preserving applications to introduce sanctions or prohibition lists. When initiating a particular action, the user can prove, in addition to the application logic, that they are not part of the sanctions lists (one or more) without compromising sensitive data. We will show how this solution can be integrated into applications, using the example of extending Freedom Tool (a voting solution based on biometric passports). We will also consider ways to manage these lists, versioning principles, configuring the filter data set, combining different lists, and using the described method in other privacy-preserving applications.

1 Introduction

Many privacy-focused applications and financial solutions built on decentralized blockchains overlook the existence of globally recognized prohibition and sanctions lists, which identify individuals who should be restricted from misusing privacy solutions. The current trend often advocates for privacy without addressing eligibility concerns. However, we believe that privacy and eligibility can be effectively combined to comply with regulatory requirements. As privacy technologies continue to advance, traditional regulatory practices may become outdated, highlighting the need for new approaches that ensure the exclusion of unauthorized individuals while maintaining the highest level of privacy for everyone else.

Such lists can be public (like the Office of Foreign Assets Control(OFAC) sanction list[Trea]) and, if properly transferred, can be a good anchor for applications that must comply with the legal environment. Naturally, the decision regarding the need to use these lists (and which specific lists) lies with the decentralized application's initiator. This article proposes a way to create such lists in an open and provable form and the effective use of such constructions by applications that need it.

One of the applications that can apply this approach today is Freedom Tool[OL24][Rar24], which has introduced a way to vote on top of public blockchain-based accounting systems, taking data from users' biometric passports as a source of eligibility criteria. To prove the right to vote, the user must generate a zero-knowledge proof that the passport is signed by one of the existing and valid issuer authorities and that the specific data stored in the passport's **DG1** (the data group that includes the person's general data) meets the criteria of the voting pool: citizenship, age, expiration date, etc. We can reuse these data and create prohibition filters for users who are included in such lists.

The paper is organized as follows. Section 2 examines the basic components needed to build sanctions lists. Section 3 is devoted to a brief overview of the Freedom Tool protocol as an example of an application that can consume information from these lists. Section 4 details the principle of building sanctions lists and combining them for end applications. Section 5 will showcase building sanctions lists based on the OFAC list. Section 6 contains some security considerations regarding the approach described in this paper. Section 7 is devoted to issues of practical application of these lists and the parties responsible for their formation.

2 Background

2.1 Zero-knowledge proofs

Zero-knowledge proving is a technology that allows a user to prove knowledge of the secret data(witness w), which, together with the public data, form a valid instance x in the relation. This proof itself does not reveal anything about the witness. Freedom Tool uses zk-SNARKs Groth16 construction[Gro16] to prove users' eligibility. zk-SNARK consists of three algorithms:

1. $\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{vp})$ — the setup algorithm that takes the security parameter λ and outputs the public parameters: proving and verification keys.
2. $\mathsf{Prove}(\mathsf{pp}, x, w) \to \pi$ — the proving algorithm that takes the prover parameters pp, statement x, and witness w, and outputs a proof π.
3. $\mathsf{Verify}(\mathsf{vp}, x, \pi) \to \{\mathsf{accept}, \mathsf{reject}\}$ — the verification algorithm that takes the verification key, statement x, and proof π, and outputs a bit indicating whether the proof is valid.

and requires completeness, knowledge soundness, zero-knowledge, and succinctness over relation $\mathcal{R}$. We refer to the zero-knowledge proofs with the following notation:

Algorithm 1 Proof construction example

Inputs:

- *Private:*

  array of private inputs

- *Public:*

  array of public inputs

Proving:

Generate proof π for relation:

$$\mathcal{R} = \{(x, w) \in \mathcal{X} \times \mathcal{W} : \phi_1(x, w) \wedge \phi_2(x, w) \wedge \cdots \wedge \phi_m(x, w)\}$$

2.2 Merkle Trees. Inclusion and exclusion proofs

Merkle Tree[Car09] is an efficient data aggregator that allows the representation of the collection of data objects to the short value with the following properties:

1. **Verification efficiency**. It's possible to prove that a particular object is included in the full set with $O(\log n)$ verification complexity.

2. **Succinctness**. The size of the root value doesn't depend on the number of leaves in the tree.

3. **Soundness**. If the piece of data isn't included in the initial set, the user can't create valid proof except with a negligible probability.

Some Merkle Tree constructions[Scr24][ide24b] additionally allow the forming of efficient exclusion proofs. These constructions are a core element we use to organize exception lists. In the following subsections, we will present the construction of the Sparse Merkle Tree.

2.2.1 Tree components

We are referring to different components of trees in the paper. To be consistent, let's start with several definitions:

- Set — a data collection, which Merkle Tree will represent.

- Leaf — a tree node that has no children. Leaves consist of three values: (h, k, v), where k determines the position of the leaf in the tree, v represents the data that we want to put in a certain position(for our purpose we can consider $v = k$) and $h = \mathsf{Hash}(k, v, 1)$. 1 is added to the h definition to disable the possibility of making proofs starting with the internal node.

- Root — the tree node that doesn't have a parent; it's the only node in the top level of the tree.

- Empty node — a tree node, representing that subtree does not include any keys generated from set elements. All tree empty nodes have $k = v = 0$ and, therefore, $h = \mathsf{Hash}(0, 0, 1)$.

- Internal node — a tree node with at least one non-empty child. Such nodes keep one value $h = \mathsf{Hash}(h_1, h_2)$, where h_1, h_2 are hash values of the children of the internal node. In the case of the empty node, we put as $h = \mathsf{Hash}(0, 0, 1)$.

2.2.2 The process of adding a leaf to the tree

This subsection specifies how to add a new leaf to the Merkle Tree. First, we calculate $k_x = \mathsf{Hash}(\mathsf{data}_1, \mathsf{data}_2, ...)$ for the data, representing a certain set member, that should be added to the tree. Let's present k_x in binary form. Then, we go through the tree's nodes, starting from the root. We step by the order of bits of binary representation of the k_x starting from the least significant bit. 0 would be a step to the left child, and 1 would be a step to the right one. After each step, there could be three possibilities:

- We are in the empty node. This empty node becomes a leaf that keeps all the needed data (h, k, v) and recalculates all the nodes we've met on the way to that node. Recalculation is made by the following rule: $h = \mathsf{Hash}(h_1, h_2)$, where h_1, h_2 are hash values of the children of the internal node.

- We are in the internal node. We make a step to one of the children depending on the corresponding bit of the binary representation of the k_x, as described above.

- We are in the leaf. This leaf would become an internal node. We will create new leaves and empty nodes until the bit of the k_x won't differ from the corresponding bit of the k value of the leaf. Here, if both bits were equal to 0, we would create an empty node as the right sibling and an internal node as the left sibling, an empty node as the left sibling, and an internal node as the right sibling. When bits become different, we create two leaves as children. One of the siblings would be a leaf with the same values h, k, v as a leaf we deleted. The other leaf has a k value, which equals k_x. Leaves are added in the order of the bits. After that, we recalculate all the nodes we've met on the path.

For clearer logic, you can give values to the edges of the tree in the following way: 0 if the edge between the left child and parent and 1 if the edge between the right child and parent. With such indexation, every path from the leaf to the root would present the last bits of the k_x binary representation.

Let us introduce an example to illustrate the algorithm above better. Consider the tree with the structure depicted in Figure 1 below. The read leaf has a $k = (\dots 1101100)_2$.

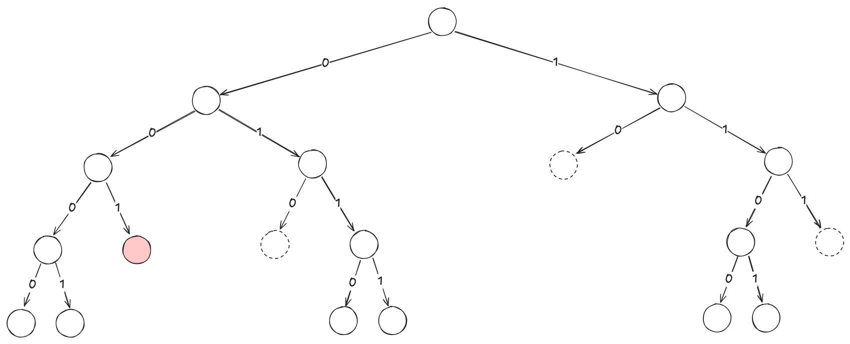


Figure 1: Example SMT (Sparse Merkle Tree). Via dotted nodes, we denote the empty nodes, while the red-colored leaf is the one we are currently considering

Then we add a leaf with $k = ...0101100$. After some steps, we will appear in red leaf. After that, we are in the third case, and we need to push both leaves down until their bits in k would differ. In the end, we have the structure presented in Figure 2.

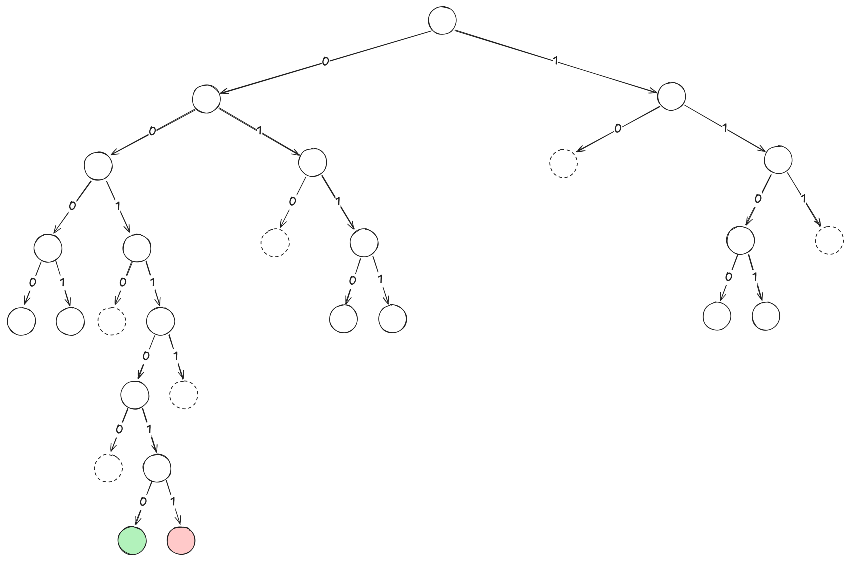


Figure 2: Structure of the SMT after adding green leaf

2.2.3 Deleting a leaf

Consider we want to delete a leaf from the tree. For each leaf, there could be two possibilities:

- If the leaf's sibling is an internal node, we just replace a leaf with an empty node. After that, we recalculate the path to that empty node.
- If the leaf sibling is not an internal node, then it's a leaf node(it can't be an empty node, or otherwise, the leaf should have been at least one level above). An empty node also replaces the sibling leaf. Their parent becomes that sibling leaf. Now, if the sibling to the parent node is an internal or leaf node, we should recalculate the path to it. Otherwise, we replace the parent with an empty node, and we should bring the sibling leaf one more level up. By the same logic, we bring up that sibling leaf until its sibling won't be an empty node(or we will be in the root). After that, we recalculate the path to that node.

Let us introduce an example of deleting a leaf from a tree. In Figure 3 we present tree structure before deleting a leaf

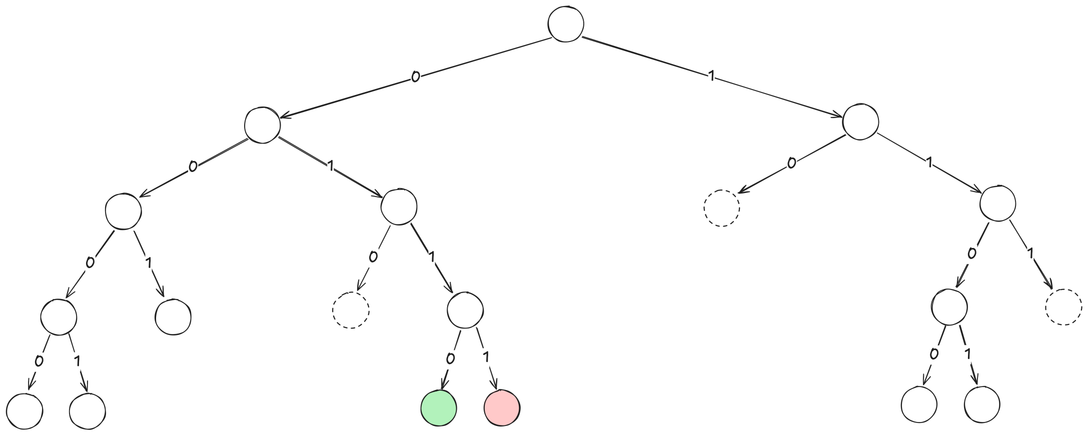


Figure 3: SMT from we are deleting a leaf. The red leaf presents the leaf that should be deleted, and the green node presents its sibling, which will change its position in the tree

We have the second case, so we should bring up a green leaf till the moment its sibling won't be an empty node as shown in Figure 4

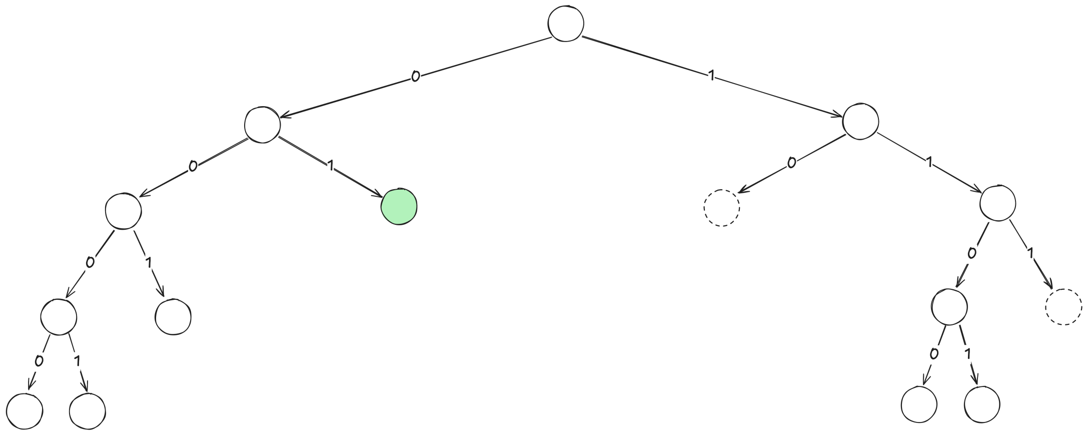


Figure 4: Result of deleting a leaf from SMT. Shows the change of the green-colored leaf position

2.2.4 Inclusion proof

Consider that Prover wants to prove inclusion to the data set represented by k_x (the key is generated based on included data). With the binary representation of k_x, Prover finds a leaf representing his data and the path to that leaf. Then, the root is deleted from the path, and Prover creates a Merkle path, an array of siblings on the path. The proof of inclusion is Merkle path, data, root, and leaf data: (h, k, v).

The verifier could check the proof in the following way:

- Verifier calculates k_x from the data and checks if $k_x = k$.
- Verifier calculates $\mathsf{current} = \mathsf{Hash}(k, v, 1)$ and last m bits of binary representation of k_x, where m is the size of Merkle path.
- Verifier calculates $\mathsf{current} = \mathsf{Hash}(\mathsf{current}, \mathsf{Merklepath_i})$ if $(m - i)$-th last bit of binary representation of k_x equals 0, and $\mathsf{current} = \mathsf{Hash}(\mathsf{Merklepath_i}, \mathsf{current})$ if $(m - i)$-th last bit of binary representation of k_x equals 1 for $i \in [0, 1, ..., m - 1]$.
- Verifier checks and accepts the proof if $\mathsf{current} = \mathsf{root}$.

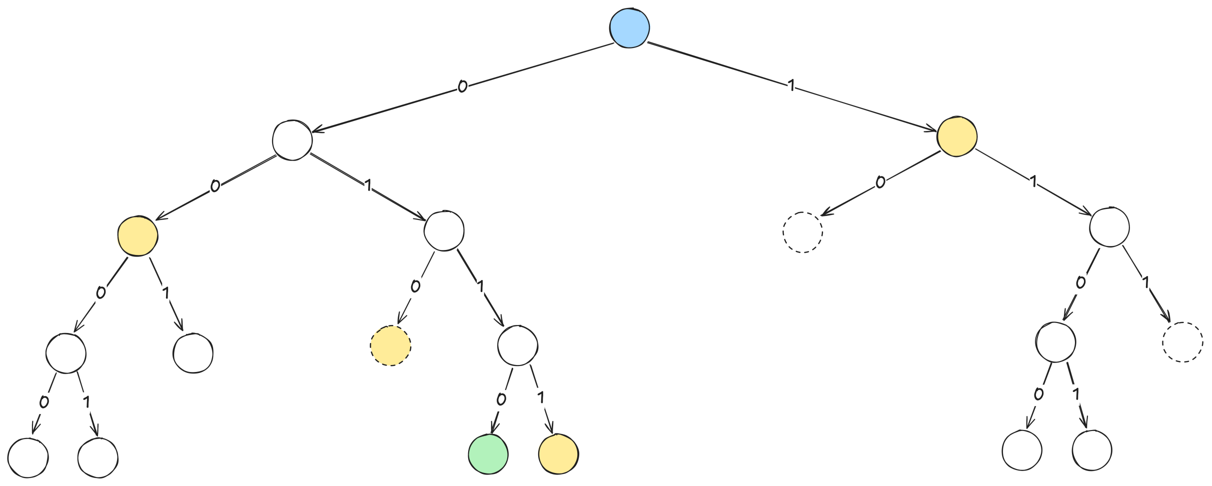


Figure 5: Example of Merkle path for inclusion proof. Green-colored leaf presents a leaf in which inclusion should be proven, yellow-colored nodes present nodes included to the Merkle path, and blue-colored node present a Merkle root of the tree

2.2.5 Exclusion proof

Consider we want to prove exclusion from the data set represented by k_x. Prover makes steps with the binary representation of k_x and finds a leaf or empty node. After that, they create a Merkle path for the found leaf or empty node. The proof of exclusion would be Merkle path, k_x, root, and found node data: (h, k, v).

The verifier could check the proof in the following way:

- Verifier calculates k_x from the data and checks if $k_x \neq k$.
- Verifier calculates $\mathsf{current} = \mathsf{Hash}(k, v, 1)$ and last m bits of binary representation of k_x, where m is the size of Merkle path.
- Verifier calculates $\mathsf{current} = \mathsf{Hash}(\mathsf{current}, \mathsf{Merklepath_i})$ if $(m - i)$-th last bit of binary representation of k_x equals 0, and $\mathsf{current} = \mathsf{Hash}(\mathsf{Merklepath_i}, \mathsf{current})$ if $(m - i)$-th last bit of binary representation of k_x equals 1 for $i \in [0, 1, ..., m - 1]$.
- Verifier checks and accepts the proof if $\mathsf{current} = \mathsf{root}$.

As you can see, exclusion proof is an inclusion proof for another element, verifying that the keys of those have the same last bits in binary representation but are still different.

3 Freedom Tool proof construction

As we mentioned in the introduction, it's nice to have an example of how existing privacy-preserving applications can be extended with prohibition lists. For this purpose, first of all, let us describe the construction of such an application with the example of the Freedom Tool.

Freedom Tool[Rar24][OL24] introduced an anonymous voting platform in which eligibility is proved with data extracted from biometric passports. The Freedom Tool protocol consists of 3 parts: mapping passport data to the user's public key, registration in the voting pool (which can be skipped), and voting.

3.1 Passport data

Biometric passports serve an essential role in proving a user's eligibility. Passports store all the necessary information about the particular person in the form of a set of data groups (**DG**s): name and surname, citizenship, date of birth, biometric data (if presumed), and issuing authority, which represents passport validity (we will not philosophize on the subject of the legitimacy of various states and their issuance authorities in this article). Fortunately, all this information can be read by phone with an RFID (radio-frequency identification).

In this section, we want to describe information written on the biometric passport chip that is needed for voting verification.

- **DG1** — a data group that stores the basic information that is presented in all biometric passports: personal information, passport number, expiration date, and issuing authority(citizenship).
- **SOD** — Document Security Object has an encapsulated content, which contains hashes of all the **DG**s presented in the passport. The signature of the hash of the encapsulated content, generated by the issuance authority, is also stored in **SOD**. Public keys of the issuance authorities that can sign encapsulated data are officially published. For easier verification of the authenticity of the public key that signed **SOD** was made an SMT of verified public keys, published by the International Civil Aviation Organization(ICAO)[ICAb].
- **DG15** — a data group which keeps information about Active Authentication public key $\mathsf{PK}_{\mathsf{pass}}$. The passport manages an appropriate secret key for signing challenges to prove passport control.

3.2 Associating the passport with an identity key

First of all, the user needs to fetch data from the passport (as we described using their device with an RFID reader). The reading process consists of the following steps:

1. Receive the basic passport data from MRZ (Machine Readable Zone)[ICAa] and generate authentication keys.
2. Scan the data from the passport in encrypted form.
3. Decrypt the passport data and verify them (under the hood, the device verifies the signature of the SOD[Raj] and the branch of certificates up to Trust Anchor; then it verifies that the data is stored in data groups is valid).
4. Store the data on the device.

So, within this process, data transfers only from the passport to the user's device; no other parties are involved.

Then, the user generates a keypair $(\mathsf{sk}_{\mathcal{I}}, \mathsf{PK}_{\mathcal{I}})$ for identity management and creates a blinder for passport data as $\mathsf{blinder} = \mathsf{Hash}(\mathsf{sk}_{\mathcal{I}})$. Blinder is designed to hide **DG1** data in the commitment. Then, the user must authenticate their identity key $\mathsf{PK}_{\mathcal{I}}$ by signing it with the passport secret key.

Then, the user generates a sub-proof that the passport is signed by the government. An additional tree includes the list of authorities' public keys. The user proves that:

1. The public key for signature verification is included in the tree.
2. The signature is valid.

The last step is to commit to the passport data as $\mathsf{Hash}(\mathsf{DG1}, \mathsf{blinder})$. When all these steps are completed, the user creates the transaction to connect the passport to the identity key. The transaction includes the following set of data:

- Identity public key $\mathsf{PK}_{\mathcal{I}}$
- Passport's public key for active authentication $\mathsf{PK}_{\mathsf{pass}}$
- DG commit $\mathsf{Hash}(\mathsf{DG1}, \mathsf{blinder})$
- proof π generated according to Algorithm 2.

Algorithm 2 Proof of passport authenticity for connecting it to identity key

Inputs:

- *Private:*

  $\mathsf{sk}_{\mathcal{I}}$ – user's identity private key

  $\mathsf{DG1}$ – passport's **DG1**

  SOD – Document Security Object

  $\mathsf{PK}_{\mathsf{iss}}$ – public key of the passport issuer

  MP – the Merkle proof for inclusion of $\mathsf{PK}_{\mathsf{iss}}$ in $\mathsf{Root}_{\mathsf{ICAO}}$

- *Public:*

  G – generator of a cyclic subgroup of elliptic curve [Bar20]

  $\mathsf{PK}_{\mathcal{I}}$ – user's identity public key

  $\mathsf{PK}_{\mathsf{pass}}$ – passport's public key

  $\mathsf{Root}_{\mathsf{ICAO}}$ – the root of SMT with issuer public keys from the ICAO list

  $\mathsf{DG}_{\mathsf{commit}}$ – commitment on passport's data from **DG1**

Proving:

Generate proof π for relation:

$$\begin{gathered}
\mathcal{R} = \{(\mathsf{sk}_{\mathcal{I}}, \mathsf{DG1}, \mathsf{SOD}, \mathsf{PK}_{\mathsf{iss}}, \mathsf{MP}, \mathsf{PK}_{\mathcal{I}}, \mathsf{PK}_{\mathsf{pass}}, \mathsf{Root}_{\mathsf{ICAO}}, \mathsf{DG}_{\mathsf{commit}}, G):\\
\mathsf{VerifyMP}(\mathsf{PK}_{\mathsf{iss}}, \mathsf{Root}_{\mathsf{ICAO}}, \mathsf{MP}) = \mathsf{true} \wedge\\
\mathsf{sig\_ver}(\mathsf{PK}_{\mathsf{iss}}, \mathsf{DG1}, \mathsf{SOD.sig}) = \mathsf{true} \wedge\\
\mathsf{Hash}(\mathsf{sk}_{\mathcal{I}}) = \mathsf{blinder} \wedge\\
\mathsf{sk}_{\mathcal{I}} \cdot G = \mathsf{PK}_{\mathcal{I}} \wedge\\
\mathsf{PK}_{\mathsf{pass}} = \mathsf{SOD.PK}_{\mathsf{pass}} \wedge\\
\mathsf{Hash}(\mathsf{DG1}, \mathsf{blinder}) = \mathsf{DG}_{\mathsf{commit}}\}
\end{gathered}$$

If the proof is valid and there is not the same passport public key in the tree — the association $\mathsf{PK}_{\mathcal{I}} \leftrightarrow \mathsf{PK}_{pass}$ is being added to the identity registry as the leaf of Sparse Merkle Tree with the following construction:

$$\begin{aligned}
\mathsf{leaf}.k &:= \mathsf{Hash}(\mathsf{Hash}(\mathsf{PK}_{\mathcal{I}}), \mathsf{Hash}(\mathsf{PK}_{\mathsf{pass}}))\\
\mathsf{leaf}.v &:= \mathsf{Hash}(\mathsf{DG}_{\mathsf{commit}}, \mathsf{timestamp})
\end{aligned}$$

3.3 Voting procedure

Now, when the passport is mapped to the user's identity keys, the voting pool can be launched. Participation in the voting pool requires the user to create proof of eligibility.

Algorithm 3 Voting eligibility proof

Inputs:

- *Private:*

  $\mathsf{sk}_{\mathcal{I}}$ – user's identity private key

  Data – string, which represent the set of data included in **DG1**:

  - $\mathsf{Data.nat}$ – nationality of the user
  - $\mathsf{Data.DOB}$ – date of birth of the user
  - $\mathsf{Data.exp}$ – expiration date of the user's passport

  ts – timestamp of creating a leaf of identity SMT

  H – hash of the passport public key, $\mathsf{H} = \mathsf{Hash}(\mathsf{PK}_{\mathsf{pass}})$

  $\mathsf{MP}_{\mathsf{Reg}}$ – the Merkle proof for inclusion of $\mathsf{Hash}(\mathsf{Hash}(\mathsf{PK}_{\mathcal{I}})$ to $\mathsf{Root}_{\mathsf{Reg}}$

- *Public:*

  G – generator of a cyclic subgroup of elliptic curve

  ID_{v} – constant value, which represents voting event identifier

  null – user's nullifier, used for uniqueness

  $\mathsf{Root}_{\mathsf{Reg}}$ – Merkle root of identity SMT

  nat – the allowed nationality of the voters

  vote – user's vote represented by field value

  $\mathsf{min}_{\mathsf{DOB}}$ – threshold timestamp for age (only users with lower DoB can vote)

  $\mathsf{max}_{\mathsf{exp}}$ – threshold expiration timestamp

Proving:

Generate proof π for relation:

$$\begin{gathered}\mathcal{R} = \{(\mathsf{sk}_{\mathcal{I}}, \mathsf{Data}, \mathsf{ts}, \mathsf{H}, \mathsf{MP}_{\mathsf{Reg}}, \mathsf{ID}_{\mathsf{v}}, \mathsf{null}, \mathsf{Root}_{\mathsf{Reg}}, \mathsf{nat}, \mathsf{vote}, \mathsf{min}_{\mathsf{DOB}}, \mathsf{max}_{\mathsf{exp}}, G) : \\ \mathsf{Data.nat} = \mathsf{nat} \wedge \\ \mathsf{Data.DOB} \le \mathsf{min}_{\mathsf{DOB}} \wedge \\ \mathsf{Data.exp} \ge \mathsf{max}_{\mathsf{exp}} \wedge \\ \mathsf{Hash}(\mathsf{sk}_{\mathcal{I}}, \mathsf{Hash}(\mathsf{sk}_{\mathcal{I}}), \mathsf{ID}_{\mathsf{v}}) = \mathsf{null} \wedge \\ \mathsf{leaf}.k := \mathsf{Hash}(\mathsf{Hash}(\mathsf{sk}_{\mathcal{I}} \cdot G), \mathsf{H}) \\ \mathsf{leaf}.v := \mathsf{Hash}(\mathsf{Hash}(\mathsf{Data}, \mathsf{Hash}(\mathsf{sk}_{\mathcal{I}}), \mathsf{ts}) \\ \mathsf{VerifyMP}(\mathsf{leaf}, \mathsf{Root}_{\mathsf{Reg}}, \mathsf{MP}_{\mathsf{Reg}}) = \mathsf{true}\}\end{gathered}$$

To make a vote, the user sends a transaction with the following data:

- Pair of nullifier and chosen by the user vote $(\mathsf{null}, \mathsf{vote})$.
- Eligibility proof π generated according to Algorithm 3.

Then, the verification smart contract:

- Verifies that $\mathsf{Root}_{\mathsf{Reg}}$ is valid and relevant (voting solution should be oriented to specific historical root value).

- Verifies that $\mathsf{nationality}$ listed in the list of allowed.
- Verifies that $\mathsf{min}_{\mathsf{DOB}}$ and $\mathsf{max}_{\mathsf{exp}}$ correspond to defined by voting organiser.
- Verifies that null is not in the SMT of nullifiers
- Adds null to the SMT of nullifiers
- Adds pair $(\mathsf{null}, \mathsf{vote})$ to the voting list and vote is counted

If the user tries to vote with the nullifier that is already included in the tree, it's recognized as a double-voting attack and an appropriate transaction to be reverted.

4 Extension for prohibition list

Paper [But+24] introduces a way to support private actions with quorum-based eligibility criteria. In other words, when you use solutions like [Rom19][Kob20], you can additionally prove you are hidden in the particular "allowed" quorum or aren't hidden in the "disallowed" quorum. The proposed protocol inherits these properties.

For some purposes, it could be necessary to create a list of people who are forbidden to participate in the voting. For now, Freedom Tool ignores these limitations, and any person from a certain country older than 18 could be able to vote. We introduce the idea of how to extend Freedom Tool voting proof to make such prevention.

The main idea is to create a list of forbidden entities and then represent it in the form of a Sparse Merkle Tree. Furthermore, we define a collision-resistant function f, which allows us to represent a person by their personal data. Function f is essential for different Sanction Tree representations. So, the voting proof π would be extended by adding proof of exclusion from the generated sanction tree presented in Algorithm 4.

Our use case, which intends to prevent sanctioned people from being eligible to participate in any kind of event (e.g., voting, dApp, KYC process, etc.), requires a data structure that can provide non-membership proofs for the requested element. Additionally, this data structure should be zk-friendly so we can preserve user confidential information.

This implies that we are limited in our choice of data structures. There are a few options we can choose from:

1. **Indexed Merkle Trees:** [Net24]: This data structure is built upon a regular Merkle Tree with a modification of the Leaf Node structure to make Non-Membership proofs possible.
2. **Sparse Merkle Trees:** A modified Merkle Tree is constructed from the top to the bottom and extends each Leaf Node with a key field, allowing the deterministic definition of an element's place within the tree. It was described in a variety of papers [ide24b; Scr24; IAC18; Mic03], where the focus was on zk-compatibility.

Algorithm 4 Voting eligibility proof with proof of exclusion

Inputs:

- *Private:*

  $\mathsf{sk}_{\mathcal{I}}$ – user's identity private key

  Data – string, which represent the set of data included in **DG1**

  - $\mathsf{Data.nat}$ – nationality of the user
  - $\mathsf{Data.DOB}$ – date of birth of the user
  - $\mathsf{Data.exp}$ – expiration date of the user's passport

  ts – timestamp of creating a leaf of identity registry SMT

  H – hash of the passport public key, $\mathsf{H} = \mathsf{Hash}(\mathsf{PK}_{\mathsf{pass}})$

  $\mathsf{MP}_{\mathsf{Reg}}$ – the Merkle proof for inclusion of $\mathsf{Hash}(\mathsf{Hash}(\mathsf{PK}_{\mathcal{I}})$ to $\mathsf{Root}_{\mathsf{Reg}}$

  $\mathsf{EP}_{\mathsf{Sanc}}$ – the Merkle exclusion proof of $f(\mathsf{Data})$ from $\mathsf{Root}_{\mathsf{Sanc}}$

- *Public:*

  G – generator of a cyclic subgroup of elliptic curve

  ID_{v} – constant value, which represents voting event identifier

  null – user's nullifier used for uniqueness

  $\mathsf{Root}_{\mathsf{Reg}}$ – Merkle root of identity registration SMT

  $\mathsf{Root}_{\mathsf{Sanc}}$ – Merkle root of Sanction Tree

  nat – nationality of the voters

  vote – user's vote represented by field value

  $\mathsf{min}_{\mathsf{DOB}}$ – threshold timestamp for age (only users with lower DoB can vote)

  $\mathsf{max}_{\mathsf{exp}}$ – threshold expiration timestamp

Proving:

Generate proof π for relation:

$$\begin{gathered}\mathcal{R} = \{(\mathsf{sk}_{\mathcal{I}}, \mathsf{Data}, \mathsf{ts}, \mathsf{H}, \mathsf{MP}_{\mathsf{Reg}}, \mathsf{EP}_{\mathsf{Sanc}}, \mathsf{ID}_{\mathsf{v}}, \mathsf{null}, \mathsf{Root}_{\mathsf{Reg}}, \mathsf{Root}_{\mathsf{Sanc}}, \mathsf{nat}, \mathsf{vote}, \mathsf{min}_{\mathsf{DOB}}, \mathsf{max}_{\mathsf{exp}}, G) : \\ \mathsf{Data.nat} = \mathsf{nat} \wedge \\ \mathsf{Data.DOB} \leq \mathsf{min}_{\mathsf{DOB}} \wedge \\ \mathsf{Data.exp} \geq \mathsf{max}_{\mathsf{exp}} \wedge \\ \mathsf{Hash}(\mathsf{sk}_{\mathcal{I}}, \mathsf{Hash}(\mathsf{sk}_{\mathcal{I}}), \mathsf{ID}_{\mathsf{v}}) = \mathsf{null} \wedge \\ \mathsf{leaf}.k := \mathsf{Hash}(\mathsf{Hash}(\mathsf{sk}_{\mathcal{I}} \cdot G), \mathsf{H}) \\ \mathsf{leaf}.v := \mathsf{Hash}(\mathsf{Hash}(\mathsf{Data}, \mathsf{Hash}(\mathsf{sk}_{\mathcal{I}})), \mathsf{ts}) \\ \mathsf{VerifyMP}(\mathsf{leaf}, \mathsf{Root}_{\mathsf{Reg}}, \mathsf{MP}_{\mathsf{Reg}}) = \mathsf{true} \wedge \\ k := f(\mathsf{Data}) \\ \mathsf{VerifyEP}(k, \mathsf{Root}_{\mathsf{Sanc}}, \mathsf{EP}_{\mathsf{Sanc}}) = \mathsf{true}\}\end{gathered}$$

These structures have their pros and cons in different situations. Our main approach would be passing the Merkle Tree Proof into the circuit to prove that a specific entity has not been included in the tree.

As mentioned, both options are capable of providing proof of non-membership. However, as stated in the Aztec documentation [Net24], it could take a while to find the lowest_null, and to make this searching process efficient, we would have to store some auxiliary data. In the case of SMT, considering our usage, the data structure is self-consistent, meaning it has, by its structure, sufficient information to grow and provide Merkle Tree Proofs (MTPs).

A good example of this property is demonstrated by the efficient implementation of this structure in Solidity [Lib24]. Additionally, we could extend the previous implementation and store more information in the storage, allowing us to create Merkle Tree Proofs for historical roots [ide24a].

The self-consistent property allows us to make the data publicly available and fully decentralized by storing the full tree on-chain. Therefore, in this article, we will rely on the Sparse Merkle Tree data structure for building and maintaining the tree.

Our approach to integrating sanction lists into the voting system is the following: The sanction list will be presented in the form of the Sparse Merkle Tree (SMT). In the sections below, we will present how such a structure is built in relation to the sanction list, how users can prove their eligibility, versioning of the trees, and how to combine different lists.

4.1 Building the sanction tree

To build the sanction tree, we construct SMT. Leaves of this SMT are constructed in the following way:

- For each entity from the sanction list we calculate $\mathsf{Hash}(\mathsf{version}, \mathsf{parameters})$, if all the $\mathsf{parameters}$ are known. A version of the tree indicates based on which parameters we will check the sanction list exclusion. Parameters could differ for different purposes; then, we would describe the choice of parameters for the OFAC Sanction list.

- For each person from the sanction list, we add the leaf with $k = \mathsf{Hash}(\mathsf{version}, \mathsf{parameters})$ to the SMT sequentially.

A sanction list could contain different information about the people, e.g., first and last names, date of birth, etc. However, some information could be missing for some people. For that reason, we added a version that allows you to choose which parameters should be checked depending on how many sanction list members the tree should cover and how many eligible votes will not be restricted from voting.

4.2 Versioning and tree usage

As the OFAC list is actively maintained, it is necessary to have an efficient way to version the information in our data structure. The SMT allows us to add, remove, and update elements to maintain the data easily.

The bigger issue lies in the trust considerations, as there could be multiple sources of truth in the future, not only considering the OFAC list that was transferred to the blockchain but also other possible lists of the same type.

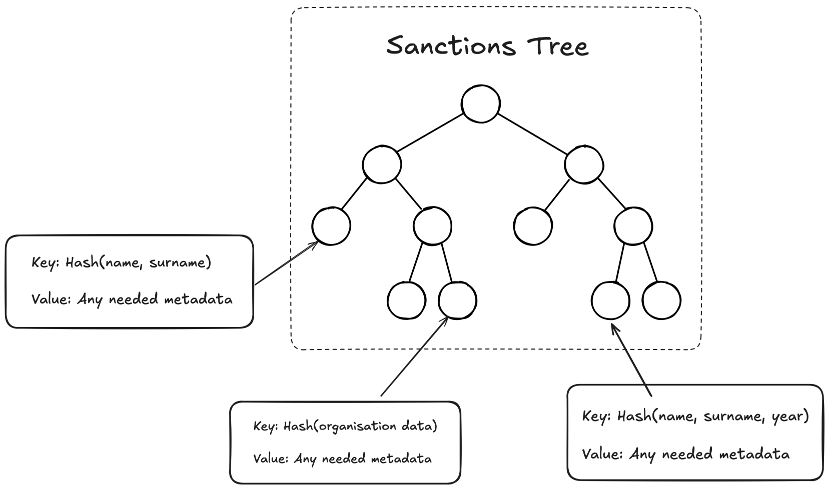
However, even in such a scenario, all of these sources are independent, and it is up to the consumer (i.e., voting platforms, DApps, etc.) to choose trusted ones and integrate them into their applications.

The most important thing for the end users is transparency and accessibility to the data. Therefore, the system's minimal setup would involve the Merkle Tree Root on-chain, which a trusted party can update. Another possible solution is to maintain the whole tree on-chain, but due to the high maintenance cost, this could be implemented on other, cheaper chains rather than on Ethereum Mainnet.

4.3 Conjunction and disjunction of trees

The next important consideration is using multiple data sources to increase the chance that an ineligible person cannot participate in the event.

In the simplest use case, all entries can be placed in one tree, as is shown in the picture below, providing a common way to generate and verify one or multiple MTPs.


Sanctions Tree
Key: Hash(name, surname)
Value: Any needed metadata
Key: Hash(organisation data)
Value: Any needed metadata
Key: Hash(name, surname, year)
Value: Any needed metadata

However, in real-world examples, multiple parties could create multiple trees. In this case, the application launcher's responsibility (DApps, voting platforms, etc.) is to choose which resources to trust and use. The cost of proof verification could increase linearly with the number of different trees used. A selector mechanism can be employed to manage the separation of what is being proven inside the circuit. In the selector, each bit could represent the exact option to verify against.

The selector mechanism allows service providers to choose which attributes to turn off/on depending on their use case.

Additionally, if the proof generation time can be neglected, the tree height can be set to 248, so there won't be a need to consider that the number of elements might overflow at some point. Otherwise, it is recommended to first assume how many elements will be in the tree and on which device the proof should be generated. In the most common use case, a tree height of 60 to 100 is recommended, given the trade-off between the possible number of elements in the tree and the time to generate the proof on the device.

In our case, we could choose the minimal setup, and as soon as we need to increase the tree capacity, we can do so through a contract upgrade and use a different circuit afterward.

5 An example of sanction tree for OFAC list

We constructed a sanction tree for the OFAC's SDN[Treb] sanction list version from 01.10.2024. To highlight the rationality of the tree-building mechanism, let's start by analyzing the data included in the OFAC list.

5.1 Sanction list data

While researching the data presented in the sanction list, we discovered that it contains individual data and data on sanctioned aircraft, vessels, and entities(corporations). We made a distribution of such data:

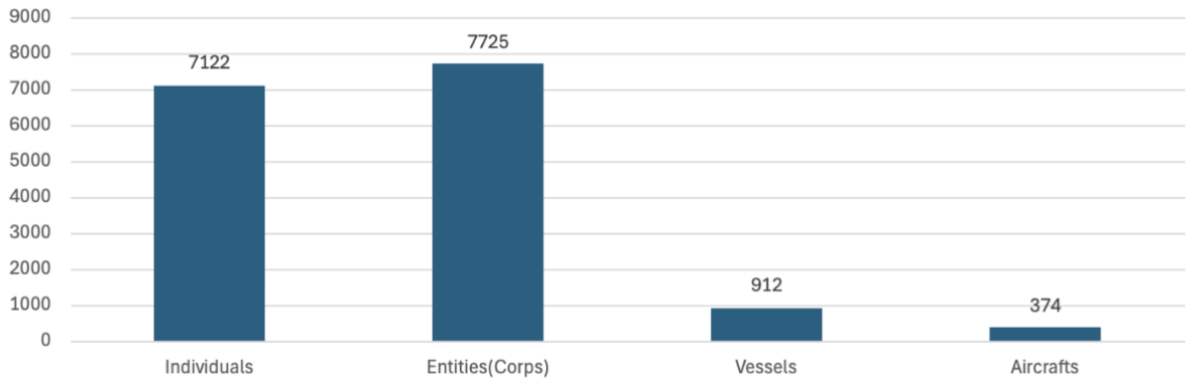


Figure 6: Sanction list data distribution

For voting purposes, only important information about individuals is considered. Unfortunately, passport data cannot provide information about a person's working place or ownership of property, so we can't determine whether a certain individual is part of the company or owner of the aircraft or vessel. That's why we will consider only information about individuals while building a sanction tree.

5.2 Individuals data

Then, we researched the individual data presented by many parameters. We will only consider the data presented in the passport **DG1** group because it is the only information presented in all biometric passports. Other information can't be checked and presented for at most half of the list members, so it can't be considered a factor in building the sanction tree.

DG1 information, given in the sanction list, is also not present for all the members. Here, you can find the percentage of individuals in the sanction list for whom each information was presented(probably not fully presented):

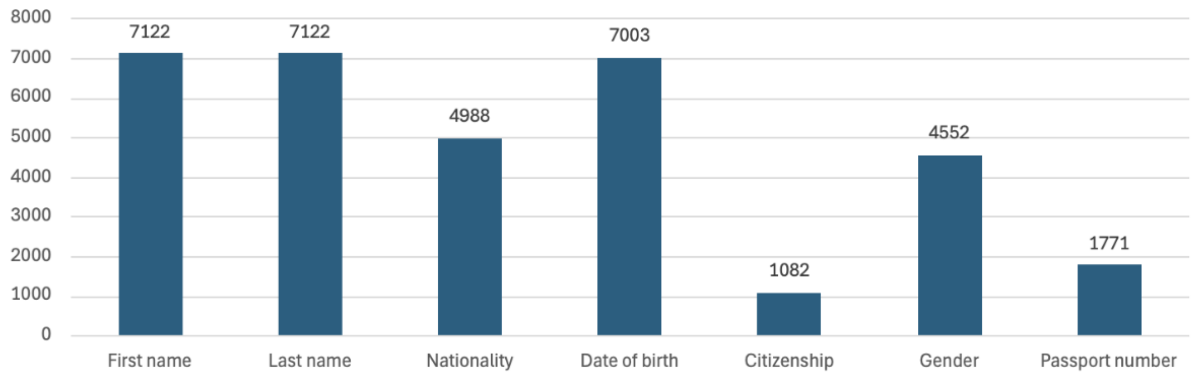


Figure 7: **DG1** information presented in OFAC list

As we can see, first and last names are the only two parameters presented for all the individuals in the list. They are perfect parameters for our purpose of preventing sanction list members from voting, but such a method causes too many collisions, and we've gone deeper into research. The next parameter, based on the inclusion rate, is the birthday date. Our further research was based precisely on the date of birth representation for members of the list. All the other parameters are presented for many people, which is not enough to start working with. So, for now, we've approached that there are two ways of determining leaves of the tree: with first and last names with or without date of birth. So, in the following section, we will introduce how we can deal with date of birth data.

5.3 Applying additional filters to DoB data

We looked closer at how the date of birth is presented in the list. Such data is presented in the OFAC list in different ways: full birthday date, full date of birth in range, approximate full date of birth, month and year in range, year in range, certain month and year, approximate year, and certain year. Below, we present statistics of how many people with certainly known parameters, and we also add a column with approximate date of birth:

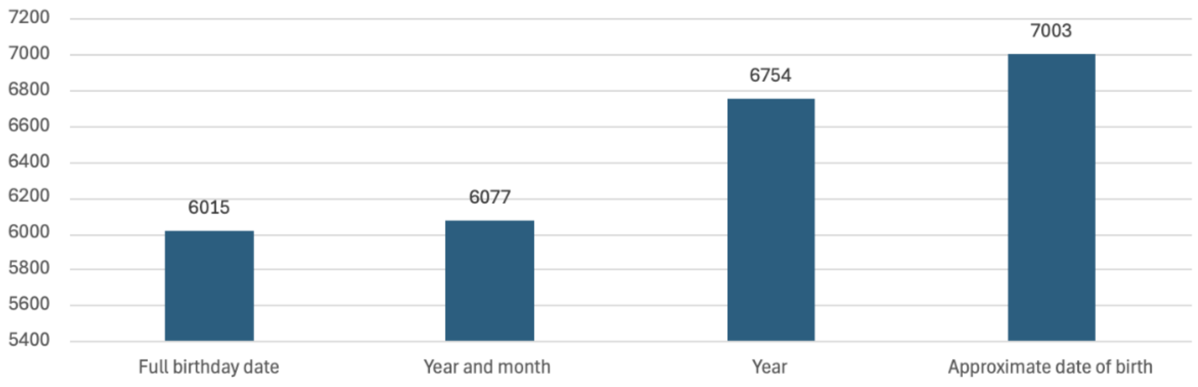


Figure 8: Statistics of date of birth representation in sanction list

While constructing a sanction tree with some data missing in the list, you should consider two things: provide the possibility for eligible users to vote and restrict sanction list members from voting. Our main purpose was to restrict **all** sanction list members from voting. As we can see, amount of people whose years are certainly known(6754) doesn't differ from all individual amounts (7122) as much as other representations. Also, adding a year to the leaf construction can hugely decrease the amount of collisions. The next section will describe how to maintain with other individuals whose year isn't known.

5.4 Tree building proposal

First, we will describe the leaf structure for building a sanction tree. Each leaf k would contain a $\mathsf{Hash}(\mathsf{name}, \mathsf{surname}, \mathsf{birth\ year})$. Such leaf structure verifies non-membership of the sanction list with high collision resistance. All of the sanction list members have their $\mathsf{name}, \mathsf{surname}$. However, $\mathsf{birth\ year}$ is unknown for some members. Moreover, some list members have $\mathsf{birth\ year}$ in some range or just approximate $\mathsf{birth\ year}$. Now we will describe how we deal with such members:

- $\mathsf{birth\ year}$ in range. For such people, we add a leaf for a year in the range, i.e., $k = \mathsf{Hash}(\mathsf{name}, \mathsf{surname}, \mathsf{year})$, for all year is in that range.

- $\mathsf{birth\ year}$ is approximate. We construct a range depending on parameter $\mathsf{appr\_range}$ for such people. Such range is given approximate $\mathsf{birth\ year}$ plus and minus $\mathsf{appr\_range}$. With the right choice of $\mathsf{appr\_range}$ parameter, it would cover all the possible options for the person in the sanction list.

- $\mathsf{birth\ year}$ is unknown. For such people we add 100 leafs, i.e. $k = \mathsf{Hash}(\mathsf{name}, \mathsf{surname}, \mathsf{year})$, where $\mathsf{year} \in [\mathsf{current\ year} - 99, \mathsf{current\ year}]$, for the time of writing this paper $\mathsf{current\ year} = 2024$.

6 Security considerations

6.1 Sanction list management

In the current setup (for the OFAC list), anybody can verify that the sanctions list was created according to specific data, ensuring that no additional data was added and all entries were included. However, the problem lies in the centralized nature of the list maintenance. At a critical moment, the centralized party can extend or destroy the list and essentially create a supply chain attack on the service that relies on this tree, causing reputational, legal, or technical consequences.

Essentially, the verifier or application chooses which tree to connect to; therefore, they are responsible for ensuring that a trustworthy party maintains the sanctions tree.

Another way to avoid this problem is to decentralize the maintenance of such a tree, but this is out of the scope of this paper.

6.2 Personal data leakage

Depending on the use case, the organization can construct the sanctions tree privately. In that case, to avoid a brute-force attack, the organization should introduce some blinder value so it will be impossible to check if an entry is within the tree by simply iterating through all possible names and birthdates.

There are different ways to generate and maintain blinders, but their detailed description is out of the scope of this paper.

6.3 Collision probability

Our scheme could be considered resistant to participation from sanction list members. However, it is still not fully collision-resistant. That's why we present a way for authorities to calculate collision probability. We would calculate the probability that we have less than m collisions for certain members of the list. We consider that the government knows how many people in the country have the same name as the given sanctioned authority - n, the same surname - s and the same birth year - y. Also, the government knows the total number of people in their country - t.

For certain citizens, the probability of having the same name as the chosen sanction list member is $\frac{n}{t}$. Here, t is the total number of people except for the individual in the sanction list, and n is the number of people with the same name as that individual except them (we could use $\frac{n-1}{t-1}$ relation but assume that 1 as a negligible value for these parameters). With the same logic, we can calculate probability for surname and birth year: $\frac{s}{t}$, $\frac{y}{t}$.

Then, we can calculate the probability that a certain person has the same data as a sanction list member. All the events above are independent, so we can calculate probability as their multiplication: $p = \frac{nsy}{t^3}$.

Then we will use Bernoulli distribution[08] to calculate probability that we have exactly k collisions: $\binom{t}{k} p^k (1-p)^{t-k}$. We denote a random variable ξ_i, which equals 1 if i-th person has a collision with sanctioned authority, and 0 if not. $\sum_{i=1}^{t} \xi_i = \xi$ is a total number of collisions. And so, finally, we have that probability that we have no more than m collisions

$$\mathsf{Pr}\left[\xi \leq m\right] \approx \sum_{k=0}^{m} \binom{t}{k} p^k (1-p)^{t-k}, \ p = \frac{nsy}{t^3}.$$

In case when we don't have a certain birthday year, we know that two events, i.e., a person was born in a certain year, where years are different, can't happen together; that's why for counting collision probability, we can add all the years in the range from which leafs to the tree were added:

$$\mathsf{Pr}\left[\xi \leq m\right] = \sum_{y \in range} \left[\sum_{k=0}^{m} \binom{t}{k} \cdot p^k (1-p)^{t-k}, \ p = \frac{nsy}{t^3} \right].$$

In practice, such Bernoulli distribution is not convenient to calculate, so via de Moivre-Laplace theorem[18], we can approximate it with a normal distribution(t is very large, p is very small):

$$\xi \sim \mathcal{N}(tp, tp(p-1))$$

$$\mathsf{Pr}\left[\xi \leq m\right] \approx \frac{1}{\sqrt{2\pi tp(p-1)}} \int_{-\infty}^{m} \exp\left(-\frac{(x-tp)^2}{2tp(p-1)}\right) dx, p = \frac{nsy}{t^3}$$

7 Practical usage

Additionally, this technology can be applied to the Freedom Tool solution and propagated to a wider spectrum of applications.

7.1 KYC vs IMOK

Nowadays, to provide KYC procedures, users are asked to provide their complete personal information: full name, photo, date of birth, passport scan, phone number, etc. However, keeping all the user's data can become a problem for users and service providers.

From the user's perspective, the problem is that the identity provider has much more data about the user than they need. If several providers agree, they can reveal some subgraph of the user's actions and make conclusions and predictions about the user's behavior.

From an identity provider's perspective, each data leak can lead to legal action, monetary losses, and a damaged reputation. Supporting GDPR and similar policies regarding personal data storage is a very sensitive and resource-heavy process.

Instead of the traditional KYC procedure, our proposed approach only allows to prove the non-inclusion of KYCed persons in some prohibition lists. The service provider doesn't know any personal information about the user but knows their claim "I'm OK" is valid.

7.2 Consuming prohibition lists by dApps

Each application that requires compliance with the existing legal environment can be similarly connected to prohibition lists. The single limitation is that this application should be connected to an identity infrastructure (Rarimo or similar) to be able to reuse passport commitments.

Implementing their own identity infrastructure with trees, passport verification logic, etc.:

1. Not the most efficient way from the development perspective.
2. Leads to identity layer segregation — users need to be separately registered in each solution before usage according to Section 3.2 of this paper.

3. Leads to decreasing of anonymity level. The bigger the identity infrastructure — the better because users are hidden in the quorum with other identity owners.

This case can be extended to the whole DeFi infrastructure, controlling the balance between privacy, permissions, and regulatory compliance [0xb].

7.3 Applying prohibition lists to other data

Additionally, we see the possibility of using this approach for arbitrary data that can and needs to be verified.

You can use the proposed approach, for example, to prove that the VIN code is not on the list of stolen ones, a specific address did not send transactions to the list of "gray" address, a mail is not signed by a key that is on the revoked list.

Arbitrary data can be used to build trees using the "rational" approach. The mechanism of the logic for checking the exclusion of this data and the mechanism for combining these trees allows you to create any arbitrary verification logic under the conditions of the existence of corresponding prohibition list infrastructure.

References

[18] "De Moivre–Laplace theorem". In: *The Doctrine of Chances.* 1718. URL: https://en.wikipedia.org/wiki/De_Moivre%E2%80%93Laplace_theorem.

[Mic03] Silvio Micali. *Zero-Knowledge Sets.* Accessed: 2024-09-19. 2003. URL: https://people.csail.mit.edu/silvio/Selected%20Scientific%20Papers/Zero%20Knowledge/Zero-Knowledge_Sets.pdf.

[08] "Bernoulli Distribution". In: *The Concise Encyclopedia of Statistics.* New York, NY: Springer New York, 2008, pp. 36–37. ISBN: 978-0-387-32833-1. DOI: 10.1007/978-0-387-32833-1_24. URL: https://doi.org/10.1007/978-0-387-32833-1_24.

[Car09] Barbara Carminati. "Merkle Trees". In: *Encyclopedia of Database Systems.* Ed. by LING LIU and M. TAMER ÖZSU. Boston, MA: Springer US, 2009, pp. 1714–1715. ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_1492. URL: https://doi.org/10.1007/978-0-387-39940-9_1492.

[Gro16] Jens Groth. "On the Size of Pairing-Based Non-interactive Arguments". In: *Advances in Cryptology – EUROCRYPT 2016.* Ed. by Marc Fischlin and Jean-Sébastien Coron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 305–326. ISBN: 978-3-662-49896-5.

[IAC18] IACR. *Origins of Zero-Knowledge Proofs.* Accessed: 2024-09-19. 2018. URL: https://eprint.iacr.org/2018/955.

[Rom19] Roman Semenov Roman Storm Alexey Pertsev. *tornado.cash privacy solution.* 2019. URL: https://berkeley-defi.github.io/assets/material/Tornado%20Cash%20Whitepaper.pdf.

[Bar20] Jordi Baylina (@jbaylina) Barry WhiteHat (@barryWhiteHat) Marta Bellés (@bellesmarta). *"ERC-2494: Baby Jubjub Elliptic Curve [DRAFT]" Ethereum Improvement Proposals, no. 2494.* Jan. 2020. URL: https://eips.ethereum.org/EIPS/eip-2494..

[But+24] Vitalik Buterin et al. “Blockchain privacy and regulatory compliance: Towards a practical equilibrium”. In: *Blockchain: Research and Applications* 5.1 (2024), p. 100176. ISSN: 2096-7209. DOI: https://doi.org/10.1016/j.bcra.2023.100176. URL: https://www.sciencedirect.com/science/article/pii/S2096720923000519.

[ide24a] iden3. *iden3 Sparse Merkle Tree Solidity Library*. Accessed: 2024-09-19. 2024. URL: https://github.com/iden3/contracts/blob/34bc4a7321c02cecb53725001d19ae2c1e6e9bee/contracts/lib/SmtLib.sol.

[ide24b] iden3. *Sparse Merkle Tree*. Accessed: 2024-09-19. 2024. URL: https://docs.iden3.io/publications/pdfs/Merkle-Tree.pdf.

[Lib24] Solidity Library. *Sparse Merkle Tree in Solidity*. Accessed: 2024-09-19. 2024. URL: https://github.com/dl-solarity/solidity-lib/blob/master/contracts/libs/data-structures/SparseMerkleTree.sol.

[Net24] Aztec Network. *Indexed Merkle Tree*. Accessed: 2024-09-19. 2024. URL: https://docs.aztec.network/aztec/concepts/storage/trees/indexed_merkle_tree.

[OL24] Kurbatov Oleksandr and Antadze Lasha. *Building ZK passport-based voting*. 2024. URL: https://mirror.xyz/0x90699B5A52BccbdFe73d5c9F3d039a33fb2D1AF6/1JSk1fvRwpfUQhLZt1OlaBpVDkXIutVYYXdYgaukh0c.

[Rar24] Rarimo. *Freedom Tool*. 2024. URL: https://freedomtool.org/#/doc.

[Scr24] Scroll.io. *zkTrie*. Accessed: 2024-09-19. 2024. URL: https://docs.scroll.io/en/technology/sequencer/zktrie/.

[0xb] 0xbow. *0xbow is the DeFi sector’s response to the growing need for a balanced approach to privacy and regulatory compliance*. URL: https://www.0xbow.io.

[ICAa] ICAO. *Doc 9303, Machine Readable Travel Documents*. URL: https://www.icao.int/publications/Documents/9303_p3_cons_en.pdf.

[ICAb] ICAO. *The ICAO Master List and ICAO Health Master List*. URL: https://www.icao.int/Security/FAL/PKD/Pages/ICAO-Master-List.aspx.

[Raj] R. Rajeshkumar. *E-Passport validation: A practical experience*. URL: https://www.icao.int/Meetings/TRIP-HongKong-2017/Documents/TRIP2017.HongKong.Rajesh(edited).pdf.

[Trea] U.S. Department of the Treasury. *OFAC lists*. URL: https://ofac.treasury.gov.

[Treb] U.S. Department of the Treasury. *Specially Designated Nationals List*. URL: https://sanctionslist.ofac.treas.gov/Home/SdnList.

[Kob20] Barry Whitehat Kobi Gurkan Koh Wei Jie. *Community Proposal: Semaphore: Zero-Knowledge Signaling on Ethereum*. March 31, 2020. URL: https://semaphore.pse.dev/whitepaper-v1.pdf.