Security Review Report
NM-0113 Polygon Id

NETHERMIND

(Sep 13, 2023)

Contents

1 Executive Summary 2

2 Audited Files 3

3 Assumptions 3

4 Summary of Issues 3

5 System Overview 4
5.1 Key Concepts . . . 4
5.2 Audited Contracts . . . 4

6 Analysis of the Probability of Collision for Identities 6

7 Risk Rating Methodology 7

8 Issues 8
8.1 [Medium] Unable to set version for a revocations . . . 8
8.2 [Low] Collision of claims' indices . . . 9
8.3 [Low] IdentityBase may return data based on non-published state . . . 9
8.4 [Low] IdentityBase may return data not synchronized with StateV2 contract . . . 9
8.5 [Low] Incorrect documentation about ID creation . . . 10
8.6 [Low] On-chain ID genesis state can't follow the specification . . . 10
8.7 [Low] State transition security properties differ for ID types . . . 11
8.8 [Low] `OnChain` identities can be left unusable . . . 11
8.9 [Low] `bytesToAddress(...)` casts to incomplete address . . . 12
8.10 [Info] Incorrect nat-spec documentation . . . 12
8.11 [Best Practice] Change names of `int256ToAddress(...)` and `int256ToBytes(...)` . . . 12
8.12 [Best Practice] Checks-effects-interactions pattern is not applied on `transitState(...)` function . . . 13
8.13 [Best Practice] Mapping variable name does not match with documentation . . . 13
8.14 [Best Practice] `IdentityBase.sol` contract can be deployed in its current state . . . 13
8.15 [Best Practice] `transitStateGeneric(...)` calls itself externally . . . 14

9 Documentation Evaluation 15
9.1 ID creation in the documentation is not updated . . . 15
9.2 Mapping variable name does not match with documentation . . . 15

10 Test Suite Evaluation 16
10.1 Tests Output . . . 16
10.2 Coverage Test . . . 27
10.3 Slither . . . 27

11 About Nethermind 28

1 Executive Summary

This document outlines the security review conducted by Nethermind for the Polygon ID protocol. Polygon ID is a blockchain-native identity system. It gives the power to build trusted and secure relationships between users and dApps, following the principles of self-sovereign identity and privacy by default.

This assessment focuses on a newly added feature allowing users to create identities managed by Ethereum accounts (EOA and smart contracts). It comprehends changes to the State contract to allow transitions from these identities and a set of contracts that will act as a base for users who want to create on-chain identities.

The audited code comprises 693 lines of code in Solidity with 83.79% of test coverage. The `PolygonID` team has provided detailed documentation explaining the protocol summary and the newly added feature. It has also communicated to clarify existing doubts and validate our understanding of the technology.

The audit was performed using: (a) manual analysis of the codebase, (b) automated analysis tools, and (c) simulation of the smart contracts. **Along this document, we report** 15 points of attention, where one are classified as `Medium`, eight are classified as `Low`, and six are classified as `Informational` or `Best Practice`. The issues are summarized in Fig. 1.

This document is organized as follows. Section 2 presents the files in the scope of this audit. Section 3 presents the assumptions for this audit. Section 4 summarizes the issues. Section 5 presents the system overview. Section 6 discusses the probability of collisions for identities. Section 7 discusses the risk rating methodology adopted for this audit. Section 8 details the issues. Section 9 discusses the documentation provided by the client for this audit. Section 10 presents the compilation, tests, and automated tests. Section 11 concludes the document.

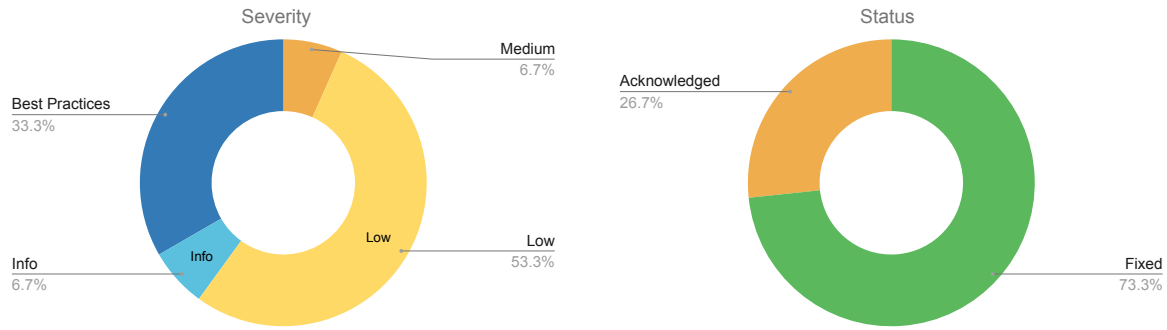

Fig 1: a) Distribution of issues: Critical (0), **High** (0), **Medium** (1), **Low** (8), **Undetermined** (0), **Informational** (1), **Best Practices** (5). **b) Distribution of status: Fixed** (11), **Acknowledged** (4), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	August 7, 2023
Final Report	September 13, 2023
Methods	Manual Review, Automated Analysis
Repository	iden3
Commit Hash	c62afb072026194289b0da7672ddbf3b816cc677
Final Hash	45eaace3de34b43fe3ecf338dc3c934b3a57d066
Documentation	PolygonID tutorials, iden3 documentation
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	contracts/state/StateV2.sol	240	170	70.8%	47	457
2	contracts/lib/IdentityBase.sol	111	101	91.0%	26	238
3	contracts/lib/GenesisUtils.sol	81	40	49.4%	26	147
4	contracts/lib/OnChainIdentity.sol	179	113	63.1%	31	323
5	contracts/interfaces/IOnchainCredentialStatusResolver.sol	27	30	111.1%	4	61
6	contracts/interfaces/IState.sol	55	70	127.3%	10	135
	Total	**693**	**524**	**75.6%**	**144**	**1361**

3 Assumptions

The prepared security review basis on the following assumptions:

- The off-chain code which interacts with the on-chain contracts is safe;
- The `Poseidon` hash library is implemented correctly, and it holds properties of hashing function;
- Circuits used to generate the zero-knowledge proof are correct;
- The `verifier` of zero-knowledge proof is responsible for checking the validity of the genesis state for a regular identity;
- The `verifier` of the zero-knowledge proof is responsible for checking the validity of state transitions from the old state to the new one for a regular identity;

4 Summary of Issues

	Finding	Severity	Update
1	Unable to set version for a revocations	Medium	Acknowledged
2	Collision of claims' indices	Low	Acknowledged
3	IdentityBase may return data based on non-published state	Low	Fixed
4	IdentityBase may return data not synchronized with StateV2 contract	Low	Acknowledged
5	Incorrect documentation about ID creation	Low	Fixed
6	On-chain ID genesis state can't follow the specification	Low	Fixed
7	State transition security properties differ for ID types	Low	Fixed
8	`OnChain` identities can be left unusable	Low	Acknowledged
9	`bytesToAddress(...)` casts to incomplete address	Low	Fixed
10	Incorrect nat-spec documentation	Info	Fixed
11	Change names of `int256ToAddress(...)` and `int256ToBytes(...)`	Best Practices	Fixed
12	Checks-effects-interactions pattern is not applied on `transitState(...)` function	Best Practices	Fixed
13	Mapping variable name does not match with documentation	Best Practices	Fixed
14	`IdentityBase.sol` contract can be deployed in its current state	Best Practices	Fixed
15	`transitStateGeneric(...)` calls itself externally	Best Practices	Fixed

5 System Overview

This audit consists of two contracts, two interfaces, and two libraries:

a) **IdentityBase**

b) **StateV2**

c) **IState**

d) **IOnchainCredentialStatusResolver**

e) **GenesisUtils**

f) **OnChainIdentity**

5.1 Key Concepts

The protocol defines several key concepts, including Identity Types and Identity State. This subsection provides a summary of these concepts to facilitate the comprehension of the contracts and libraries.

Identity Types - There are two types of Identities:

- Regular Identity - It is created from roots of three merkle trees (Claim Tree, Revocation Tree, and Roots Tree).
- Ethereum-controlled Identity - This type of identity uses Ethereum accounts to authenticate and perform state transitions.

Identity State - An Identity State consists of three SMT (Sparse Merkle Trees):

- Claims Tree - It contains the claims issued by an identity.
- Revocations Tree - It contains revocation nonces of the submitted claims that were revoked by an identity.
- Roots Tree - It contains the history of the tree roots from the Claims tree.

When an identity issues a claim, it is added to the Claims Tree. If the Claims Tree Root is changed, it is also added to the Roots Tree. The Revocation Tree is changed when an identity invalidates a claim. The revocation is done by adding the claim revocation nonce to the Revocation Tree. The action of changing Claims and Revocations Trees modifies the identity state, and it must be updated on-chain by executing a state transition. The identity state represents the status of an identity at a specific point in time. The initial state of an identity is known as the Genesis State.

5.2 Audited Contracts

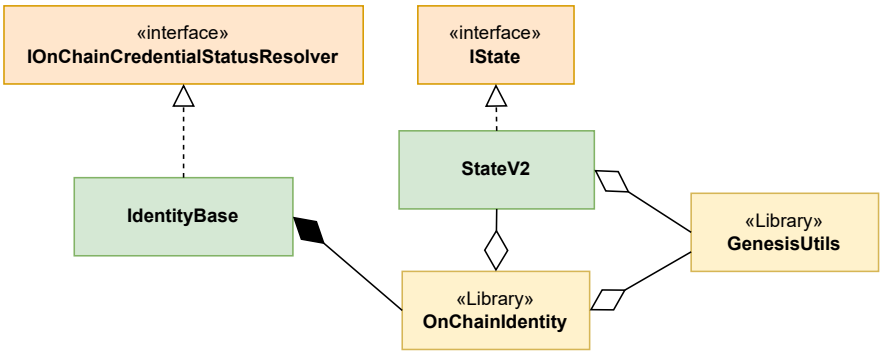

Fig. 2: Structural Diagram of Contracts

Fig. 2 presents the structural diagram of the contracts. The contract **StateV2** , which is a part of the Polygon ID's core component and its main goal is to store information about identities. The contract saves the state for each identity. Basically, it has a SMT and a state variable to store identities with their respective state in a mapping. This SMT works as a `GIST` (Global Identities State Tree) that register the state of all identities using the protocol. The `GIST` is stored in the **StateV2** contract, and every time a user executes one state transition function, the new state of an identity is added to the `GIST`. This feature allows users to prove ownership of an Identity by validating that it is included in the `GIST` without revealing their `Genesis ID`.

The contract implements 23 public / external functions, and the main functions are presented below:

- transitState(...) - Changes the state of an identity (i.e., transits to the new state) with ZKP ownership check.
- transitStateGeneric(...) - Changes the state of an identity under a specific method id ownership check.

- setDefaultIdType(...) - Only the owner can call this function to set the defaultIdType external wrapper.
- setVerifier(...) - Only the owner can call it to set ZKP verifier contract address.

The getter functions retrieve several distinct pieces of information, such as the last state info for a given identity, state info for a particular identity, GIST latest root, GIST root history, and GIST inclusion or non-inclusion proof for a given identity by the block number or timestamp.

The contract **IdentityBase** is a base contract to build OnChain Identity or Issuer Contracts with required public interfaces implemented. **OnChainIdentity** is a library used by **IdentityBase** to create identity, manage the three trees (Claims Tree, Revocations Tree, and Roots Tree), add / revoke claims, and execute state transitions by calling the State Contract. It consists of 16 external / public functions, and the main functions are:

- addClaim(...) and addClaimHash(...) - These functions are used to add a claim.
- revokeClaim(...) - Revokes a claim using its revocation nonce.
- transitState(...) - Do the state transition. If at least one of the Identity trees has changed, it updates the Roots Tree if needed, performs the function transitStateGeneric in State Contract, and updates the internal variables.
- calcIdentityState(...) - Calculates IdentityState by computing the hash of the roots (Claims Tree Root, Revocations Tree Root, and Roots Tree Root).

Finally, the **GenesisUtils** is a library used to generate ID from an Ethereum address or identity state and also verify it. The library contains functions to calculate on chain ID from an address and to check if an ID is Genesis State.

6 Analysis of the Probability of Collision for Identities

This section discusses the collision probability between two Identities. The newly added feature introduces two possible scenarios to discuss: i) Collisions between two On-chain Identities and ii) Collisions between an On-chain Identity and a Regular Identity.

Collisions between two On-chain Identities - On-chain identities are generated in the following way:

$$\text{genesisState} = \text{zeroPadding} + \text{ethAddress}$$
$$\text{genesisId} = \text{idType} + \text{genesisState} + \text{checksum}$$

where,
***idType** - consists of 2 bytes. It represents the identifier of DID method and blockchain network;*
***zeroPadding** - 7 zero bytes;*
***ethAddress** - This is the Ethereum address of controlling Ethereum account, 20 bytes;*
***checksum** - consists of 2 bytes. It represents the control checksum of idType + zeroPadding + ethAddress.*

All components presented above (**idType**, **zeroPadding**, and **checksum**) except for **ethAddress** can have collisions. For this scenario, the mentioned exception makes it impossible to have two identical Identities derived from different Ethereum Addresses.

Collision between an On-chain Identity and a Regular Identity - Identity generation for both, **Regular** and **Ethereum controlled** identities, can be specified as:

$$\text{genesisId} = \text{idType} + \text{genesisStateTransformation} + \text{checksum}$$

The differences arise on the `genesisStateTransformation` value. For **Regular** identities, it is generated as: **the first 27 bytes of the genesis state**, where:

$$\text{genesisState} = \text{Hash}(\text{ClaimsTreeRoot}||\text{RevocationsTreeRoot}||\text{RootsTreeRoot})$$

For **Ethereum controlled** identities, the `genesisStateTransformation` value is the concatenation of 7 zero bytes with the 20 bytes of a valid Ethereum Address:

$$\text{genesisStateTransformation} = 7\text{BytesOfZeroPadding} + 20\text{BytesOfAValidEthereumAddress}$$

For this analysis, let's assume that we are able to control the preimage of the hash for the **genesisState** of **Regular** identities at will (which is not true). A collision for the bytes of a given Ethereum ID happens with probability $2^{-8\times20} = 2^{-160}$. **If you have N Ethereum IDs already in the system, this probability scales linearly, i.e. $N \times 2^{-160}$.**

This is without accounting for the proof-of-work that has to happen to make the first 7 bytes equal to zero, i.e. $2^{-7\times8} = 2^{-56}$. All in all, we get a probability of $N \times 2^{-216}$ for a single of the regular IDs to collide with one of the Ethereum IDs.

From that, we get a binomial distribution for whether one out of M GenesisIDs can collide with one out of N EthereumIDs:

$$P = 1 - (1 - \frac{N}{2^{216}})^M$$

If $N = 2^{30}$ (roughly a trillion ETH IDs), then a probability of collision of one in a billion (roughly $P = 2^{-20}$) would take roughly $2^{-20} \times 2^{186} = 2^{166}$ attempts in M.

Note that this analysis assumes full control over the preimage of the hash from which the Regular ID is created. This is false for this implementation since the preimage is the hash of the three merkle tree roots, with the added requirement that at least one BJJ public key must be added to the Claims tree.

7 Risk Rating Methodology

The risk rating methodology used by Nethermind follows the principles established by the OWASP Foundation. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood measures how likely an attacker will uncover and exploit the finding. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to Motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploits the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Severity Risk		
Impact	**High**	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Info/Best Practices	Low	Medium
	Undetermined	Undetermined	Undetermined	Undetermined
		Low	**Medium**	**High**
		Likelihood		

To address issues that do not fit a High/Medium/Low severity, Nethermind also uses three more finding severities: **Informational**, **Best Practices**, and **Undetermined**.

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to formally pass to the client;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

c) **Undetermined** findings are used when we cannot predict the impact or likelihood of the issue.

8 Issues

8.1 [Medium] Unable to set version for a revocations

File(s): OnChainIdentity.sol

Description: Based on the documentation, an updatable claim can be updated by adding a leaf to the Claim Tree by incrementing the version(32 bits) and changing the claim value part.

```
Index:
 i_0: [ 128  bits ] claim schema
      [ 32 bits ] option flags
          [3] Subject:
            000: A.1 Self
            001: invalid
            010: A.2.i OtherIden Index
            011: A.2.v OtherIden Value
            100: B.i Object Index
            101: B.v Object Value
          [1] Expiration: bool
          [1] Updatable: bool
          [3] Merklized: data is merklized root is stored in the:
            000: none
            001: C.i Root Index (root located in i_2)
            010: C.v Root Value (root located in v_2)
          [24] 0
      [ 32 bits ] version (optional?)
      [ 61 bits ] 0 - reserved for future use
 i_1: [ 248 bits] identity (case b) (optional)
      [  5 bits ] 0
 i_2: [ 253 bits] 0
 i_3: [ 253 bits] 0
Value:
 v_0: [ 64 bits ]  revocation nonce
      [ 64 bits ]  expiration date (optional)
      [ 125 bits] 0 - reserved
 v_1: [ 248 bits] identity (case c) (optional)
      [  5 bits ] 0
 v_2: [ 253 bits] 0
 v_3: [ 253 bits] 0

```

Then, adding a leaf to the Revocation Tree containing the revocation nonce(64 bits) and the highest invalid version(32 bits) (that is, all the claims with that nonce and version equal or lower to the one in the leaf are invalid).

```
Revocation Tree Leaf:
leaf: [ 64 bits ] revocation nonce
      [ 32 bits ] version
      [157 bits ] 0
```

When an updatable claim needs to be revoked completely, without the possibility of updating, the highest version and the revocation nonce should be added to the ReT. But the `revokeClaim` function takes only revocationNonce as an argument. No version argument is included.

```
/**
 * @dev Revoke claim using it's revocationNonce
 * @param revocationNonce - revocation nonce
 */
// @audit based on documentation, missing version argument
function revokeClaim(Identity storage self, uint64 revocationNonce) external {
    self.trees.revocationsTree.addLeaf(uint256(revocationNonce), 0);
}
```

This creates an issue if an identity wants to update a claim:

1. add a new leaf(with increased version and new claim value) to claim tree;
2. add a new leaf(revocation nonce) to revocation tree;

This will revoke the claim completely instead of just updating and revoking the old claim.

Recommendation(s): Consider adding a version(32 bits) argument to the `revokeClaim` function as mentioned in the docs.

Status: Acknowledged.

Update from the client: The claim versioning currently exists in the documentation only. There is no version implemented either in the core Golang libraries or in the contracts. Actually, we just put zero bits in place of a version, so it looks like this approach is fine, and a version can be added later as a backward-compatible feature.

8.2 [Low] Collision of claims' indices

File(s): `OnChainIdentity.sol`

Description: Claims are added to the claims tree in the `OnChainIdentity` contract. The trees can be initialized with a depth up to a maximum of `40`. Depending on the number of claims an issuer needs to publish, this depth may be insufficient, leading to a probability of index collision in the Sparse Merkle Trees (SMTs). For example, if the number of claims inserted in the tree is around 1 million, a tree with a depth `40` has a collision probability of around `60%`. If the number of claims rises to approximately 10 million, the collision probability effectively becomes `100%`.

Recommendation(s): Consider raising the maximum permissible tree depth. Also, it would be beneficial to inform users about the necessary tree depth for their specific use case when setting up an on-chain identity contract. This information should be prominently featured in user guides or any related documentation.

Status: Acknowledged.

Update from the client: We are limiting the depth of the trees to 40 as the optimal depth, which balances the collision / ZK-proof-generation-performance ratio. We'll consider including it in the documentation.

8.3 [Low] IdentityBase may return data based on non-published state

File(s): `IdentityBase.sol`, `OnChainIdentity.sol`

Description: The getter functions in the IdentityBase contract may return data not synchronized with the StateV2 contract. The functions `getClaimProof(...)`, `getClaimsTreeRoot(...)`, `getRevocationProof(...)`, and `getRevocationsTreeRoot(...)` return data using the newest root of a claim/revocation tree. However, those roots may refer to changes to the tree that were not yet committed to the StateV2 contract with the state transition. This may result in, e.g., generating proof for a claim that was not yet confirmed on the StateV2 contract.

Recommendation(s): Consider using the `self.lastTreeRoots.claimsRoot` and `self.lastTreeRoots.revocationRoot` in the OnChainIdentity library instead of the latest root for returning roots and generating proofs for claims and revocations

Status: Fixed.

Update from the client: `getClaimProof(...)` and `getRevocationProof(...)` return proofs based on the latest published roots but not the current tree roots. If there is a need to get proof for non-published roots or any roots in the history, users can use `getClaimProofByRoot(...)` and `getRevocationProofByRoot(...)` functions. The `getClaimsTreeRoot(...)` and `getRevocationsTreeRoot(...)` didn't change. They return current roots, which may not be published yet.

8.4 [Low] IdentityBase may return data not synchronized with StateV2 contract

File(s): `IdentityBase.sol`, `OnChainIdentity.sol`

Description: The getter functions in the IdentityBase contract may return data not synchronized with the StateV2 contract. The `getRevocationStatus(...)` function in `IdentityBase.sol` checks if a particular nonce has been revoked, returning an inclusion or non-inclusion proof of the specified nonce in the `revocationTree`. However, this function does not verify if the data stored in the `Identity` struct is consistent with the data in the `State` contract. As indicated in the documentation, state transitions can be executed through Zero-Knowledge Proofs (ZKP) if a Baby Jubjub (BJJ) key has been added to the identity. If a state transition occurs this way, the change may not be reflected in the information stored in the contract using the `OnChainIdentity` library, leading to stale data being used by `getRevocationStatus(...)`.

Recommendation(s): Consider ensuring that the latest state saved in the contract using the `OnChainIdentity` library matches the data in the `StateV2` contract to avoid potential data inconsistencies

Status: Acknowledged.

Update from the client: We decided not to introduce any restriction from the Onchain Identity side or the State contract side (at least for now). Yes, this may result in the Onchain Identity state being stale. However, the risk of that is close to zero and still can't even impact more than one identity, which harmed itself on purpose in this case. The warning of the data inconsistency was added to the documentation here.

8.5 [Low] Incorrect documentation about ID creation

File(s): StateV2.sol

Description: There is a misalignment between the documentation and the actual implementation regarding creating an ID for Ethereum-controlled Identity. According to the documentation, the process for ID creation should be as follows:

```
idType + zeroPadding + ethAddress + checksum
```

The documentation gives an example:

```
idType:         0212 // DID method: PolygonID; Network: Polygon Mumbai
ethAddress: 0x0dcd1bf9a1b36ce34237eeafef220932846bcd82
```

The corresponding ID creation is presented as:

```
id: 021200000000000000dcd1bf9a1b36ce34237eeafef220932846bcd82450a (bytes)
id: 2qCU58EJgrELSJT6EzT27Rw9DhvwamAdbMLpePztYq (base58)
```

However, the actual implementation in the `GenesisUtils.calcOnchainIdFromAddress(...)` function uses a different process:

```
reverse(idType + zeroPadding + ethAddress + reverse(checksum))
```

Applying the same arguments to this process results in a different ID:

```
id: 0A4582CD6B84320922EFAFEE3742E36CB3A1F91BCD0D000000000000001202 (bytes)
id: A5tDcNxacVgBQ4yHRvqv1FMR7cqNG74xGDhBWMidaq (base58)
```

Another issue resides in the documentation's representation of checksum calculation. The example presents the checksum as:

```
checksum: 450a // checksum of idType + zeroPadding + ethAddress
```

However, following the algorithm from the documentation, the checksum should not be reversed, resulting in:

```
checksum 0a45 // checksum of idType + zeroPadding + ethAddress (actual, non-reverted)
```

Recommendation(s): It is recommended to synchronize the implementation of the ID creation algorithm with its documentation. This alignment will help to ensure clarity and accuracy.

Status: Fixed.

Update from the client: Client smart contracts expect identifiers in little-endian byte order, so the resulting identifier is reversed, which we've fixed in the documentation. The same we've done regarding the checksum, we've added a note to the docs that the checksum is uint16 sum of all the bytes of idType + zero-padding + ethAddress, but the bytes of the uint16 are in reversed order, e.g., if the sum is 0x0a45 then the checksum is 0x450a. The above will be reflected in the online docs as soon as the changes are processed.

8.6 [Low] On-chain ID genesis state can't follow the specification

File(s): OnChainIdentity.sol

Description: The documentation states that the genesis state of an Ethereum-controlled identity is:

```
genesisState = zeroPadding + ethAddress
```

However, the OnChainIdentity library does not provide an option for defining a genesis state. It isn't possible to set it during initialization, and the first state transition, its value remains unset, defaulting to `0`.

Recommendation(s): Consider allowing for providing a genesis state during initialization and checking if it corresponds with ID.

Status: Fixed.

Update from the client: Fixed docs and the code so that the genesis state of Ethereum-controlled identity is always zero, which is the desired behavior.

8.7 [Low] State transition security properties differ for ID types

File(s): `StateV2.sol`

Description: The `StateV2` contract allows authorized users of an identity to execute state transitions through the `transitState(...)` and `transitStateGeneric(...)` functions. The former requires a zero-knowledge proof of the correctness of the provided data, while the latter validates that the caller is the identity owner. In the `transitStateGeneric(...)` function, two checks are properly conducted in `transitState(...)` but not adequately executed:

1. Verifying that the new user state (`newUserState`) corresponds to the hash of the new claims tree root (`newClaimsTreeRoot`), the revocation tree root (`newRevTreeRoot`), and the roots tree root (`newRootsTreeRoot`);
2. Ensuring that the old user state (`oldUserState`) and `newUserState` are distinct. Currently, `transitStateGeneric(...)` allows `oldUserState` to be equal to `newUserState` if `oldUserState` is the genesis state. This condition usually gets checked by confirming that the new state does not exist. However, if `oldUserState` is the genesis state, both states are added after the existence check, bypassing the validation;

Recommendation(s): To maintain the integrity of state transitions, it's recommended to introduce the following changes to the `transitStateGeneric(...)` function:

- Explicitly check that `oldUserState != newUserState`;
- Verify that `newUserState` is the hash of the three tree roots (`newClaimsTreeRoot`, `newRevTreeRoot`, `newRootsTreeRoot`). To achieve this, these tree roots need to be passed as arguments to the function;

Status: Fixed.

Update from the client: We've fixed the `oldUserState != newUserState`, a case we missed for the unique state's restriction for an identity. But there is no reason to check if the state is derived from tree roots. It is not State contract responsibility.

Update from Nethermind: Checking if the state is derived from the three roots is one of the properties ensured by the ZKP transition. Documenting the properties ensured by the `onChain` transitions could be helpful.

Update from the client: The onChain transition properties where documented in this page

8.8 [Low] `OnChain` identities can be left unusable

File(s): `OnChainIdentity.sol`

Description: PolygonID supports several types of identities. This includes "regular" identities managed by Baby Jubjub (BJJ) keys and Ethereum-controlled identities managed by Ethereum accounts. An Ethereum-controlled identity can gain additional functionality by adding at least one BJJ key as authorized, acting as a hybrid of the two primary types. However, this can cause issues with state transitions.

When a BJJ key is used to transition the identity state, a discrepancy between the `latestState` in the contract using `OnChainIdentity` and the last state recorded in the `StateV2` contract arises. This discrepancy means that the `transitState(...)` function in the contract using `OnChainIdentity` will no longer function correctly, as it relies on the on-chain recorded state being the latest one. The `StateV2` contract will then revert upon attempting a state transition as the latest state recorded by the contract controller identity differs from the new state set via the BJJ key.

In summary, once a BJJ key transition has been made, the identity will no longer be able to update its state using the smart contract, leading to a partially operable identity.

Recommendation(s): Consider either preventing Ethereum-controlled identities from making state transitions using BJJ keys or document this issue thoroughly. It's crucial to inform users not to perform state transitions with BJJ keys if they use the `OnChainIdentity` library to manage their identities.

Status: Acknowledged.

Update from the client: Yes, the scenario is feasible. However, we decided not to disable using BJJ keys for Ethereum-controlled identities but rather document this issue and recommend users avoid such scenarios. Plus, always create their onchain-identities as upgradable to handle "disasters".

8.9 [Low] `bytesToAddress(...)` casts to incomplete address

File(s): `GenesisUtils.sol`

Description: The function `bytesToAddress(...)` receives bytes and returns its respective address. However, the function does not return the full address.

```
function bytesToAddress(bytes memory bys) internal pure returns (address addr) {
    assembly {
        // @audit it's not returning the full address
        addr := mload(add(bys, 20))
    }
}

function int256ToAddress(uint256 input) internal pure returns (address) {
    return bytesToAddress(int256ToBytes(reverse(input)));
}
```

Proof-of-Concept:

1. Let the address equal to `0x5B38Da6a701c568545dCfcB03FcB875f56beddC4`;
2. Now we cast to `uint256` and apply `reverse(...)` function:
   `output = 89045105184058068735626723095753784881446600650024375522886434599973815320576`;
3. Then we call the function `int256ToAddress(...)` with the `output` value (this function uses `bytesToAddress(...)`;
4. The returned address is `0x0000000000000000000000005b38Da6A701c5685`;

Recommendation(s): Consider reviewing the intended use of this function and checking correctness based on that review.

Status: Fixed.

Update from the client: The function is not used by any other component, so it was removed from the repo.

8.10 [Info] Incorrect nat-spec documentation

File(s): `IdentityBase.sol`, `OnChainIdentity.sol`

Description: There are a few incorrectly commented functions:

- `getRootProof(...)` and `getRootProofByRoot(...)` have incorrectly described the return value as "The RevocationsTree inclusion or non-inclusion proof for the claim", which does not match the returned value in both `IdentityBase` contract and `OnChainIdentity` library;
- `getLastRevocationsRoot(...)` returns `revocationsRoot`, but the comment states "@return claimsRoot";

Recommendation(s): Rewrite nat-spec for listed functions.

Status: Fixed.

8.11 [Best Practice] Change names of `int256ToAddress(...)` and `int256ToBytes(...)`

File(s): `GenesisUtils.sol`

Description: The `GenesisUtils` library incorporates the functions `int256ToAddress(...)` and `int256ToBytes(...)`. However, these function names can be confusing as they convert a `uint256` type variable, not an `int256`. In Solidity, `int256` and `uint256` are distinct types. Consequently, the current function names may misleadingly imply that the input value should be of type `int256`.

Recommendation(s): To enhance code readability and prevent potential misunderstandings, we recommend renaming `int256ToAddress(...)` to `uint256ToAddress(...)` and `int256ToBytes(...)` to `uint256ToBytes(...)`.

Status: Fixed.

8.12 [Best Practice] Checks-effects-interactions pattern is not applied on `transitState(...)` function

File(s): `OnChainIdentity.sol`

Description: The `transitState(...)` function computes the new state for the Identity and executes a state transition on the `State` contract. The function makes an external call to the `State` contract, made before the contract's storage is updated.

```
    function transitState(Identity storage self) external {
        ...
        // @audit - Execute state transition in the State contract. This will be done through an external call.
        // do state transition in State Contract
        self.stateContract.transitStateGeneric(
            self.id,
            self.latestState,
            newIdentityState,
            self.isOldStateGenesis,
            1, // state transition method id: 1 - using ethereum auth
            new bytes(0) // empty method params
        );

        // @audit - Update contract's storage
        // update internal state vars
        self.latestState = newIdentityState;
        ...
    }
```

The `State` contract is upgradeable, and its logic could change in the future in a way that would pass control of the call to the original caller or a different contract. This would open a possibility for a reentrancy attack vector.

Recommendation(s): It is recommended to apply the checks-effects-interactions pattern to mitigate potential reentrancy attack vectors. The contract's storage should be updated before making the external call to the `State` contract.

Status: Fixed

Update from the client:

8.13 [Best Practice] Mapping variable name does not match with documentation

File(s): `StateV2.sol`

Description: The `State` contract stores identities and their respective states in `_stateData`.

```
StateLib.Data internal _stateData;
```

According to the documentation, the mapping that stores identities is named `identities` as described in the text below:

> Hereafter, this identity is represented as a mapping: `ID => IdState`. This gets published, together with all other identities, inside the `identities` mapping, which is part of the State contract.

Recommendation(s): Consider updating the documentation as is defined in the code to increase code comprehension.

Status: Fixed.

Update from the client: Fixed in the docs and will be updated once the relevant PR in the docs repo is merged and processed.

8.14 [Best Practice] `IdentityBase.sol` contract can be deployed in its current state

File(s): `IdentityBase.sol`

Description: The `IdentityBase` contract, intended to be inherited by contracts that manage on-chain identities, only provides some of the basic functionalities required for such management. More complex mechanisms, such as state transitions and claim additions, are expected to be implemented by the inheriting contracts. The contract could be denoted as abstract to make this requirement more explicit and promote good programming practices.

Recommendation(s): Consider designating the `IdentityBase` contract as an abstract contract to signal that it only provides base functionality and requires further implementation.

Status: Fixed.

8.15 [Best Practice] `transitStateGeneric(...)` calls itself externally

File(s): `StateV2.sol`

Description: The function `transitStateGeneric(...)` currently makes an external call to `this.getDefaultIdType(...)`. In the context of Solidity, this pattern leads to the contract calling itself externally. It's noteworthy that in this specific instance, such an external call to `getDefaultIdType(...)` is unnecessary, given that the function's visibility is set to `public`.

Recommendation(s): Avoiding external calls is a best practice. Therefore, consider calling `getDefaultIdType(...)` without a `this` keyword.

Status: Fixed.

9 Documentation Evaluation

Software documentation refers to written or visual information describing software's functionality, architecture, design, and implementation. It provides a comprehensive overview of the software system and helps users, developers, and stakeholders understand how the software works, how to use it, and how to maintain it. Software documentation can take different forms, such as user manuals, system manuals, technical specifications, requirements documents, design documents, and code comments. Software documentation is critical in software development, enabling effective communication between developers, testers, users, and other stakeholders. It helps to ensure that everyone involved in the development process has a shared understanding of the software system and its functionality. Moreover, software documentation can improve software maintenance by providing a clear and complete understanding of the software system, making it easier for developers to maintain, modify, and update the software over time. Smart contracts can use various types of software documentation. Some of the most common types include:

- Technical whitepaper: A technical whitepaper is a comprehensive document describing the smart contract's design and technical details. It includes information about the purpose of the contract, its architecture, its components, and how they interact with each other;
- User manual: A user manual is a document that provides information about how to use the smart contract. It includes step-by-step instructions on how to perform various tasks and explains the different features and functionalities of the contract;
- Code documentation: Code documentation is a document that provides details about the code of the smart contract. It includes information about the functions, variables, and classes used in the code, as well as explanations of how they work;
- API documentation: API documentation is a document that provides information about the API (Application Programming Interface) of the smart contract. It includes details about the methods, parameters, and responses that can be used to interact with the contract;
- Testing documentation: Testing documentation is a document that provides information about how the smart contract was tested. It includes details about the test cases that were used, the results of the tests, and any issues that were identified during testing;
- Audit documentation: Audit documentation includes reports, notes, and other materials related to the security audit of the smart contract. This type of documentation is critical in ensuring that the smart contract is secure and free from vulnerabilities.

These types of documentation are essential for smart contract development and maintenance. They help ensure that the contract is properly designed, implemented, and tested, and they provide a reference for developers who need to modify or maintain the contract in the future.

The `Polygon` team has provided extensive documentation on Polygon ID, including developer-friendly documentation:

- An initial tutorial describing several concepts, such as Identity holder, Issuer, Verifier, off-chain and on-chain verification. This documentation can be found in https://0xpolygonid.github.io/tutorials/.
- The another one includes in detail core concepts, such as Identity Types, Identity State, and OnChain Identity. This documentation can be accessed in https://docs.iden3.io/getting-started/getting-started/.
- Their README explains the StateV2 smart contract, how to run tests, and the deployment script.

Use of code comments is present in some files, explaining the goal and behavior of multiple functions. Moreover, the Polygon team was available to address any inquiries or concerns raised by the Nethermind auditors.

9.1 ID creation in the documentation is not updated

File(s): `StateV2.sol`

ID creation for Ethereum-controlled Identity is not implemented the same way presented in the documentation. More information about it can be found in Issue 8.4.

9.2 Mapping variable name does not match with documentation

File(s): `StateV2.sol`

In the contract `StateV2` there is a mismatch on the state variable that stores identities and their respective state in:

```
1 StateLib.Data internal _stateData;
```

More information about it can be found in Issue 8.13 and the documentation.

10 Test Suite Evaluation

10.1 Tests Output

```
$ npx hardhat test
Warning: Function state mutability can be restricted to pure
   --> contracts/lib/SmtLib.sol:566:5:
    |
566 |     function _getNodeHash(Node memory node) internal view returns (uint256) {
    |     ^ (Relevant source part starts here and spans across multiple lines).


Generating typings for: 3 artifacts in dir: typechain for target: ethers-v5
Successfully generated 9 typings!
Compiled 3 Solidity files successfully


GenesisUtilsWrapper deployed to: 0x5FbDB2315678afecb367f032d93F642f64180aa3
  generate ID from genesis state and idType
     testVector: 0 (43ms)
     testVector: 1
     testVector: 2
     testVector: 3
     testVector: 4
     testVector: 5
     testVector: 6
     testVector: 7
     testVector: 8
     testVector: 9
     testVector: 10
     testVector: 11
     testVector: 12
     testVector: 13
     testVector: 14

  check provided IDs in the genesis state
     testVector: 0 (41ms)
     testVector: 1 (40ms)
     testVector: 2 (43ms)
     testVector: 3 (40ms)
     testVector: 4 (39ms)
     testVector: 5 (39ms)
     testVector: 6 (38ms)
     testVector: 7 (39ms)
     testVector: 8 (39ms)
     testVector: 9 (39ms)
     testVector: 10 (39ms)
     testVector: 11 (39ms)
     testVector: 12 (38ms)
     testVector: 13 (39ms)
     testVector: 14 (38ms)

  test is genesis state with base 10 bigint
     base 10 bigint test

  Claim builder tests
ClaimBuilderWrapper deployed to: 0x9fE46736679d2D9a65F0992F2272dE9f3c7fa6e0
     validate buildClaim (254ms)
     validate buildClaim errors (122ms)
25
     validate buildClaim from file (555ms)

  Next tests reproduce identity life cycle
Poseidon1Elements deployed to: 0x92d9dd5Db15d8fb4B9F6F7d6655A4f70200F5487
Poseidon2Elements deployed to: 0x41d982994F7A0341A90076cB2b40109dC85e9400
Poseidon3Elements deployed to: 0xBB80dFE07aF6c69DBa4b056de735c03AF6c3bbe8
Poseidon4Elements deployed to: 0xA72B1c240505DE217e8Df4DB42Bd468bB388e777
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2
```

```
    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.
Warning: Potentially unsafe deployment of contracts/test-helpers/IdentityExample.sol:IdentityExample

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

    create identity
       deploy state and identity
0xA8c517D83e224bbb05F93226F25A60fc2837Ae5B
93c5bae3728fc605af22632f905bb4b223ed817c5a8000000000000001202
       validate identity's id
    validate initial identity
       trees should be empty
       last roots should be empty
       since the identity did not perform the transition - the isGenesis flag should be true
       latest identity state should be empty
    add claim
       we should have proof about claim existing (75ms)
       after insert identity should update only claims tree root
       another trees should be empty
       latest roots should't be change
       computes state should be different from latest saved state (54ms)
    make transition
       latest roots for ClaimsTree and RootOfRoots should be updated
       Revocation root should be empty
       Root of roots and claims root should be updated
       calculatet and saved status should be same (50ms)
    revoke state
       revoced index should exists in Revocation tree (75ms)
       transit of revocation tree shouldn't update root of roots tree
       Root of Roots and Claims Root should be changed
    make transition after revocation
       state should be updated

  Claims tree proofs
Poseidon1Elements deployed to: 0x6d0Da1f9Ed02dBD12E52A8C5810b45696DF0ea13
Poseidon2Elements deployed to: 0x5e3877F619A7e38C319e0f1d2bb53527dF1467CE
Poseidon3Elements deployed to: 0xB2a5E59151F889348003b8BBf6dB65b1b455f181
Poseidon4Elements deployed to: 0xB35981BeD3cA180907307a0c84DFe95303Cfd999
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

Warning: Potentially unsafe deployment of contracts/test-helpers/IdentityExample.sol:IdentityExample

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

     Insert new claim and generate proof (133ms)
     Get proof for claim by root (307ms)

  Revocation tree proofs
Poseidon1Elements deployed to: 0x411a01F9DF89D8C9CcbeBAf48d614fFE639b5A8d
Poseidon2Elements deployed to: 0x6C1320fB5439d39f2303649C9E8C46d42AB105f9
Poseidon3Elements deployed to: 0xB59fed0737C9541E9322F3f46D944B4D14070A09
Poseidon4Elements deployed to: 0x07bea70fE739ABfCfcF94e24b84Ae6b3C8f43F17
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

Warning: Potentially unsafe deployment of contracts/test-helpers/IdentityExample.sol:IdentityExample

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

     Insert new record to revocation tree and generate proof (133ms)
     Get proof for revocation by root (309ms)
```

```
  Root of roots tree proofs
Poseidon1Elements deployed to: 0xA5F77Aac4C90e3Fc9ece5F924036a61875f3eA74
Poseidon2Elements deployed to: 0x7Aba43DfCf469387CE307825E1e734ddb4a5942F
Poseidon3Elements deployed to: 0x8806D82799e4206E52c1EfAc82F91f43a92814aA
Poseidon4Elements deployed to: 0xfBFc6bAEccC09C8D239a0d5951bd6b34559A3737

Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2
    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

Warning: Potentially unsafe deployment of contracts/test-helpers/IdentityExample.sol:IdentityExample

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

    Insert two claims and make transtion state
       Check that root of roots not empty
    Get current Claims tree root and check that is root exists on Root of Roots tree
       Check that Root of Roots tree contains old Claims tee root (76ms)
    Check historical Claim tree root in claims tree root
       Check that Root of Roots tree not contains latest claims tree root (81ms)
       Check that current Root of roots contains latest claims tree root (81ms)

  Compare historical roots with latest roots from tree
Poseidon1Elements deployed to: 0x9Ce00050e1aEcCaC2B1775f288Dd0F34Eb245E16
Poseidon2Elements deployed to: 0xBE08b5c213e700CbA359bf35513410527c0a7a54
Poseidon3Elements deployed to: 0x974BaA3259a240735dC76ae6c9032b08Fa8B8bb1
Poseidon4Elements deployed to: 0x5aFDccB04577168c73c0Dd910cAC8725E34b2e4A
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

Warning: Potentially unsafe deployment of contracts/test-helpers/IdentityExample.sol:IdentityExample

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

    Insert and revoke claims
       Compare latest claims tree root
       Compare latest revocations tree root
       Compare latest roots tree root

  Compare historical roots with latest roots from tree
Poseidon1Elements deployed to: 0xc9Fb99AbBBED2ea1086b07cc2BF318B856Cc1d3F
Poseidon2Elements deployed to: 0xFfE5290faf572729C551C69cca1499689def4051
Poseidon3Elements deployed to: 0x73F3Ffc08C80D3c1c6dDD75f864386029821B828
Poseidon4Elements deployed to: 0x599e71E357ca448Cb186dA942c7537A90666C710
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

Warning: Potentially unsafe deployment of contracts/test-helpers/IdentityExample.sol:IdentityExample

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

    Check prev states
       Compare latest claims tree root
       Compare latest revocations tree root
       Compare latest roots tree root
    Check next states
       Check historical claims tree root
       Check historical revocations tree root
       Check historical roots tree root

  Genesis state doens't have history of states
Poseidon1Elements deployed to: 0xa58f2674552cee734Ad4b8D5fc8713a800d83308
Poseidon2Elements deployed to: 0x9856499A34Db324F6Ae91Ff1Ba61F15E8f06218D
Poseidon3Elements deployed to: 0x4A1D894CA51fa1Fe2CffA378174606dc5d37e9ce
Poseidon4Elements deployed to: 0xaFD5e2AB5dcd651341cE7BD4B28C0705c39523B1
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2
```

```
    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

Warning: Potentially unsafe deployment of contracts/test-helpers/IdentityExample.sol:IdentityExample

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

    Empty history map
       Got an error (55ms)
  poseidon

Poseidon1Elements deployed to: 0x19DBB915B0be0D535D7Aea36882b0E1EfF12e506
Poseidon2Elements deployed to: 0x111538E763C3d6e31a530f6aBA22E6F33900ddEC
Poseidon3Elements deployed to: 0x17547A6D3497b985F75904B5533366Ca5E725e03
Poseidon4Elements deployed to: 0x20c0958C0a5ac8F49f2564f563D62ed771A91939
Poseidon5Elements deployed to: 0xb4Df840c1D477bdB1362916FcbdFB38c25a0f2f5
Poseidon6Elements deployed to: 0x9af8A3E74F6529bb631BdE51759103AbDbaA5fAf
SpongePoseidon deployed to: 0x7751c981F30D037e8e85A2d0B14b0950689F8675
PoseidonFacade deployed to: 0x532B40A1c809CE8e52A94CAe8935426ebF59ed82
     check poseidon hash function with inputs [1, 2]
     check poseidon hash function with inputs [1, 2, 3] (38ms)
     check sponge poseidon hash function with inputs (22047ms)
  Merkle tree proofs of SMT
    SMT existence proof
      keys 4 (100), 2 (010)
Poseidon2Elements deployed to: 0x834529304Fb7f96B82A391aeebbCE1F6A7D88641
Poseidon3Elements deployed to: 0x649216Aa4B56C36B4F66D503B7800aE780Bd3b66
         add 1 leaf and generate the proof for it (124ms)
Poseidon2Elements deployed to: 0xFD9f46f270a0DDE6Ad7C3d0D1e0137727e8502F1
Poseidon3Elements deployed to: 0xCa4c0319eB12EA64AeEfDCb1e2Ecd0E6744D2884
         add 2 leaves (depth = 2) and generate the proof of the second one (301ms)
Poseidon2Elements deployed to: 0x24fd577091010C21a19358A4cECa1875B766Cb17
Poseidon3Elements deployed to: 0x526D59c3Dca44b077e8a41ba5f3063588c79A3fd
         add 2 leaves (depth = 2) update 2nd one and generate the proof of the first one (408ms)
Poseidon2Elements deployed to: 0xd5320Eb30DD849E1A51567Eb3e9A22F7a3bB9530
Poseidon3Elements deployed to: 0xB38fE774330c3283cD58C690f3A772AB68368bdd
         add 2 leaves (depth = 2) update the 2nd leaf and generate the proof of the second one (409ms)
Poseidon2Elements deployed to: 0xF98140D9113383497B7114f51A6b68A3F9F8E70A
Poseidon3Elements deployed to: 0x602Cf679b97647acd6BA279cfE634D3d2ACC2EA0
         add 2 leaves (depth = 2) update the 2nd leaf and generate the proof of the first one for the previous root state
         ↪  (414ms)
Poseidon2Elements deployed to: 0x4ecFAb2ABB713088cC340c4754965441bF344f2A
Poseidon3Elements deployed to: 0x8E8A0955a3b0b459a0528E6b632B0Ad96436eFE1
         add 2 leaves (depth = 2) update the 2nd leaf and generate the proof of the second one for the previous root
         ↪  state (404ms)
      keys 3 (011), 7 (111)
Poseidon2Elements deployed to: 0x54b7BC15b0F66dc53f6f55a6770be2b90e5697ed
Poseidon3Elements deployed to: 0xAc609f206B5C4Aa80A49bA4c60B6127A54f0b16c
         add 1 leaf and generate the proof for it (122ms)
Poseidon2Elements deployed to: 0x74B85eEc26a1772ecaf802260A64b9b72FC5c875
Poseidon3Elements deployed to: 0x692d5227bC7a7c0Fc39a2315B1117DD2C91ADd55
         add 2 leaves (depth = 2) and generate the proof of the second one (327ms)
Poseidon2Elements deployed to: 0x7a8320Ba627a675fd7F824770399C5D221841589
Poseidon3Elements deployed to: 0xc04F335Fec14c63a21fD9a09D9D87CF8c39f49C9
         add 2 leaves (depth = 2) update 2nd one and generate the proof of the first one (462ms)
Poseidon2Elements deployed to: 0x3E332a12199d5e0947FAfbF92674cD82cfB566e7
Poseidon3Elements deployed to: 0x504E5D623DFDdf7a298D7078262221E0237Abf86
         add 2 leaves (depth = 2) update the 2nd leaf and generate the proof of the second one (463ms)
Poseidon2Elements deployed to: 0x4a5E6388995831f96a9F2A9Af706fDff0c0B02bC
Poseidon3Elements deployed to: 0x14D734087430c0f5AC1B3002af94713276E11D6B
         add 2 leaves (depth = 2) update the 2nd leaf and generate the proof of the first one for the previous root state
         ↪  (464ms)
Poseidon2Elements deployed to: 0xbF1A6ca2E32C89a61118F89Ac88064b6a85f1fA6
Poseidon3Elements deployed to: 0x4144C2F08d86f5F11BeE5cb467f4eCA1Ce79A06f
         add 2 leaves (depth = 2) update the 2nd leaf and generate the proof of the second one for the previous root
         ↪  state (460ms)
      big keys and values
Poseidon2Elements deployed to: 0x402bA89DF96EE92A05fbA7A7A9020D7445e07d1A
Poseidon3Elements deployed to: 0x90B6d4ea36e0C771715d5E278B2650b27dC17A1d
         add 10 big keys and values and generate the proof of the last one (2031ms)
    SMT non existence proof
      keys 4 (100), 2 (010)
```

```
Poseidon2Elements deployed to: 0x93Fe0868ad77F35fD522bfe44d298E3EBF02fdf4
Poseidon3Elements deployed to: 0xe5444D312A29963f5E8eBa9F0abC9F15dD302442
        add 1 leaf and generate a proof on non-existing leaf (124ms)
Poseidon2Elements deployed to: 0x7D437A3E076Db34CD42BF3Fee708781f08458aBe
Poseidon3Elements deployed to: 0x03026BBd18936c9b3B95d816166c98CE5264bac7

        add 2 leaves (depth = 2) and generate proof on non-existing leaf WITH aux node (307ms)
Poseidon2Elements deployed to: 0x90988994efF887581DAC97BbB3FEf28805021846
Poseidon3Elements deployed to: 0x8AA35a5edDD7c37fc86610aA07A2ee3c8ba2F6d4
        add 2 leaves (depth = 2) and generate proof on non-existing leaf WITHOUT aux node (304ms)
Poseidon2Elements deployed to: 0x175970D3F72F1fAFE410590282DF30F0F0ecfd07
Poseidon3Elements deployed to: 0x27a7F0e96AcCe219491D59fA35810d2418e8cbf8
        add 2 leaves (depth = 2), update the 2nd leaf and generate proof of non-existing leaf WITH aux node (which
        ↪  existed before update) (408ms)
Poseidon2Elements deployed to: 0x450Ebf9503ae0Ef5E485478c1c9Af2371a7Ab359
Poseidon3Elements deployed to: 0x1AAE946244F3fb9564EE7CD82A5b6A285b8e78Dc
        add 2 leaves (depth = 2), update the 2nd leaf and generate proof of non-existing leaf WITHOUT aux node (407ms)
Poseidon2Elements deployed to: 0xEb41c483EA4e0797B67bbf4bc82C6247d1d72F16
Poseidon3Elements deployed to: 0x5CCD1B29B23363072a0B3088028991bf79847043
        add 2 leaves (depth = 2), add 3rd leaf and generate proof of non-existence for the 3rd leaf in the previous
        ↪  root state (385ms)
      keys 3 (011), 7 (111)
Poseidon2Elements deployed to: 0x541b42E5B8B4845bc1D9C2aeb96D45C3C48ccC85
Poseidon3Elements deployed to: 0xDd0EaC8569d1244648e04F2117E0Aca38aC2C6a1
        add 1 leaf and generate a proof on non-existing leaf (124ms)
Poseidon2Elements deployed to: 0xa23DE7B375Cc971068dc4A0F9C3f50719DE33Fac
Poseidon3Elements deployed to: 0xC447632aC0dCA599117B174Bd6A0e8b5c6fef46E
        add 2 leaves (depth = 2) and generate proof on non-existing leaf WITH aux node (336ms)
Poseidon2Elements deployed to: 0x763eeFb63c87dd65d97242217E6f1E412f8368FC
Poseidon3Elements deployed to: 0x5e24F7085B3d03Ae20ca98Fee810E901a25191eF
        add 2 leaves (depth = 2) and generate proof on non-existing leaf WITHOUT aux node (432ms)
Poseidon2Elements deployed to: 0xD7F7963D576CDF6f643a501591924C65152A0494
Poseidon3Elements deployed to: 0xf5c7146246cE44eC3d141b057d2C89f5a91dF86D
        add 2 leaves (depth = 2), update the 2nd leaf and generate proof of non-existing leaf WITH aux node (which
        ↪  existed before update) (462ms)
Poseidon2Elements deployed to: 0xd8Dc7B4ED2004605dB0903A57cBd291063a9FdC6
Poseidon3Elements deployed to: 0xbf9f76553Af6904d760CC09Ca2f82c6B69DbA8e0
        add 2 leaves (depth = 2), update the 2nd leaf and generate proof of non-existing leaf WITHOUT aux node (457ms)
Poseidon2Elements deployed to: 0x24898C07c149AAA8E85DEB78FB141B0e285ba348
Poseidon3Elements deployed to: 0x56DeE572c735bd0CE320Db67641f177d607030BF
        add 2 leaves (depth = 2), add 3rd leaf and generate proof of non-existence for the 3rd leaf in the previous
        ↪  root state (556ms)
      big keys and values
Poseidon2Elements deployed to: 0xD9a9C4baAFd0fF8394D3dFDd50B62161e26837a6
Poseidon3Elements deployed to: 0x8C65C5107F59f4C611A80864f73B76bB3e359B0a
        add 10 leaves and generate a proof on non-existing WITH aux node (1998ms)
Poseidon2Elements deployed to: 0x1eD78475d41155641AE4C7249aA7c67a3edd0603
Poseidon3Elements deployed to: 0x38B3dd1C66706A668Cd17738d02290D96e829929
        add 10 leaves and generate a proof on non-existing WITHOUT aux node (2003ms)
      empty tree
Poseidon2Elements deployed to: 0x61F87e4D03bAEA1E8899De9855dB0f7071Fb04a4
Poseidon3Elements deployed to: 0x3BF2AAd4375B545c2C6F22cB6C3798007CBf0c73
        generate proof for some key (75ms)
Poseidon2Elements deployed to: 0xA849aa84d6ea0eb1c6396A17aFE65867540D9DEE
Poseidon3Elements deployed to: 0x138f5d50FF2800B2626B5092f10626F087eF0DfC
        generate proof for some key and zero historical root (124ms)
    SMT add leaf edge cases
Poseidon2Elements deployed to: 0xca356971Ce01FcC2304D69d8422f9147ad398aAc
Poseidon3Elements deployed to: 0x9E76b6f57C40F2e7F4643976F4CE6C0A0DbdFb3A
       Positive: add two leaves with maximum depth (less significant bits SET) (2135ms)
Poseidon2Elements deployed to: 0x5E9cB429439225445B1D9991bAa25e1eFaa1aBe2
Poseidon3Elements deployed to: 0x2Cd710A886928766c29C15Cab70c1EdE0Db4bA12
       Positive: add two leaves with maximum depth (less significant bits NOT SET) (2092ms)
Poseidon2Elements deployed to: 0xe5d5D3b068638D008175ff6C556eABb48a83759A
Poseidon3Elements deployed to: 0xcc6FA8e1F4cdbe268EBDe6c9504acaD74aCa7351
       Positive: add two leaves with maximum depth (less significant bits are both SET and NOT SET) (1124ms)
Poseidon2Elements deployed to: 0x4b6456705891A64884Ab1DB7A950b10e8AAf401c
Poseidon3Elements deployed to: 0x2F9AFA809810E075D64AD79a047D9Df258Cf4e3b
       Negative: add two leaves with maximum depth + 1 (less significant bits SET) (53ms)
Poseidon2Elements deployed to: 0xc54b75bdb5350b60Cb374b98EF2957a4d8c84d97
Poseidon3Elements deployed to: 0xDf003Ae2e2f86cC7bE641928684a922a46c0925b
       Negative: add two leaves with maximum depth + 1 (less significant bits NOT SET) (50ms)
Poseidon2Elements deployed to: 0xD7ef914ed4796fAB7671616cBaB6BB7608053528
```

```
Poseidon3Elements deployed to: 0x564c17637D6b5632C2dEf314B9F5a494158f5B71
       Negative: add two leaves with maximum depth + 1 (less significant bits are both SET and NOT SET (51ms)

  Root history requests
Poseidon2Elements deployed to: 0x5E08b66787e87e4A28236F0caEC73312D20cb06f
Poseidon3Elements deployed to: 0x23189eAeB545D6353F7e100769302322Bd83e1Ef
     should return the root history
     should revert if length is zero
     should revert if length limit exceeded
     should revert if out of bounds
     should NOT revert if startIndex + length >= historyLength
  Root history duplicates
Poseidon2Elements deployed to: 0xCc5BCf7f7664898Fa1b8B7DEa49E27Eac51dd2F8
Poseidon3Elements deployed to: 0x86b8b5468e3F01c7b12de480143aF58f15e1d874
     comprehensive check (421ms)
Poseidon2Elements deployed to: 0xeBbDaBFc342FDBD2be0df61DB7a31095d6C2bF7A
Poseidon3Elements deployed to: 0x7A8B48f5De2f742e6587298ef6e5A745F307E387
     should revert if length is zero
Poseidon2Elements deployed to: 0x09f3684424520ec1E1971078007876c05b38791E
Poseidon3Elements deployed to: 0xa2cbe2648C26BAC21C3854F4d4F7FBA49a1B6D3c
     should revert if length limit exceeded
Poseidon2Elements deployed to: 0xEC6B0B0b040933811C4255A112C4618fb64bEE82
Poseidon3Elements deployed to: 0x9147a9919cA4283b34cBF34B740ACffeE3d30fe0
     should revert if out of bounds (79ms)
Poseidon2Elements deployed to: 0xF9a0fc3E8abcca5761cB3DF809E351f2e935bb4c
Poseidon3Elements deployed to: 0xb07F2587BEeECaa49F39F7F66Cf538AFa38e4635
     should NOT revert if startIndex + length >= historyLength (95ms)
Poseidon2Elements deployed to: 0x5D82F5eF05044c51090130a04743216D1C8322Fd
Poseidon3Elements deployed to: 0x787eE89D488fAcF55fd311C570F01B07EdB3fD96
     should return correct list and length just after init
  Binary search in SMT root history
    Empty history
Poseidon2Elements deployed to: 0xc31A67dA9954F2C5274B93190d6F523D2e6EB10d
Poseidon3Elements deployed to: 0x6169bC86A98C26fb304fa5bdD27474d8DA1F23c8
       Should return zero root for some search
    One root in the root history
Poseidon2Elements deployed to: 0x25379583A1ABD77E99dc93B3c69973a38ed142b1
Poseidon3Elements deployed to: 0xB31A30B3f5453D97b8E263ED6782bf30e6465125
       Should return the first root when equal
Poseidon2Elements deployed to: 0x15ec312aD23Fae7Ecfc4635B74622624ba6f881A
Poseidon3Elements deployed to: 0xC48C15Ab61E1F72979a677225886Cc5BB1562E52
       Should return zero when search for less than the first
Poseidon2Elements deployed to: 0x30c33DCCb7B15FBc7ADa9d08B3F6f6888489f6eA
Poseidon3Elements deployed to: 0x9EE3DD6184e842fd9686b145Afe85e0A01431c69
       Should return the last root when search for greater than the last
    Two roots in the root history
Poseidon2Elements deployed to: 0xc3ca1c396725D675E7D9c1B6E378Dd0865cEB5ca
Poseidon3Elements deployed to: 0xbE1f06d3E5e4ab8E0A8b5fae48eef1f287D186A6
       Should return the first root when search for equal
Poseidon2Elements deployed to: 0xCde0058591fAC040B1bbf8d95E0Cb9836f0A8a6E
Poseidon3Elements deployed to: 0xC3db8dd34751e5c4E164E1dA9a76caEC7F872134
       Should return the second root when search for equal
Poseidon2Elements deployed to: 0xb867b7B80Df96FaAAeD812EECb31F73Ecc217824
Poseidon3Elements deployed to: 0x72F463C4e1B8E0af596e44d97D10670675a23c0e
       Should return zero when search for less than the first
Poseidon2Elements deployed to: 0x7C646bbaF7265d35BD21E0292199bB00Dc10DD50
Poseidon3Elements deployed to: 0x7875181FC8D1fB84dfec03a416084E799D35e1f3
       Should return the last root when search for greater than the last
    Three roots in the root history
Poseidon2Elements deployed to: 0x8e273D0D0F93e1ADb264317aFe52998a54C2f785
Poseidon3Elements deployed to: 0x7F493D74D265b3362B903AF4638f13cf6b12c634
       Should return the first root when equal
Poseidon2Elements deployed to: 0x99AE517041EFEF76401525a09bE85671c0b3D776
Poseidon3Elements deployed to: 0x18E27876a65C7966c79376e88e39D2936EAD62C9
       Should return the second root when equal
Poseidon2Elements deployed to: 0xe1c713c81EB7efB3eC868b0dd9B01Ae81eb4eef0
Poseidon3Elements deployed to: 0xAeBA59C5791AD3dc935343047E5fbDe904bD808A
       Should return the third root when equal
Poseidon2Elements deployed to: 0x515FC766d93e1BdBBaaED176b69D912b4dC86D0E
Poseidon3Elements deployed to: 0x6900Ca1258F2D38Fb93847791cd998fBA1Db353E
       Should return zero root when search for less than the first
Poseidon2Elements deployed to: 0x3A76e5ffcfe4552d1e2954C3eC018E473B699a95
Poseidon3Elements deployed to: 0x05185c3D6C2d04A0D1Ee28F33aafdb63821106e8
```

```
      Should return the last root when search for greater than the last
    Four roots in the root history
Poseidon2Elements deployed to: 0x990F32f0dD1862375AdEC34ff96eb07fDAd506f2
Poseidon3Elements deployed to: 0x83Ac812785e56D3F6148bbB786c419cea27cBC3D
      Should return the first root when equal
Poseidon2Elements deployed to: 0x2181E312C4F68eC7f5e3382aaf806449B499F595
Poseidon3Elements deployed to: 0x41eaE179A167EBC0b72333C62f87195C62f6e24B
      Should return the fourth root when equal
Poseidon2Elements deployed to: 0x35Cf3374A2b327b9De94Ae3137d1Ce356ac613D5
Poseidon3Elements deployed to: 0xaeAbb8ACb44BE08294ACDAFf12064423B1c77193
      Should return zero when search for less than the first
Poseidon2Elements deployed to: 0xdeCF72D452329d392aEaFbd5c489BFc785C8dB31
Poseidon3Elements deployed to: 0xDb4fB4861e5C45c17fE09fAb220C50238e38a573
      Should return the last root when search for greater than the last
    Search in between the values
Poseidon2Elements deployed to: 0x8F77E5fb6dF63d622A3691FD3cc3D5b6fB9B11b5
Poseidon3Elements deployed to: 0xC240e09833717D2ABFadA18Dedcc7c1e18190d2D
      Should return the first root when search in between the first and second
Poseidon2Elements deployed to: 0xA6e68499aFEd2E33566890A56E072b7FdA91b94b
Poseidon3Elements deployed to: 0xe924Eda52E4e49633150443724d095d489ebB22E
      Should return the fourth root when search in between the fourth and the fifth
    Search in array with duplicated values
Poseidon2Elements deployed to: 0x057f2d199D5583337423024Fb538BFA96F79944A
Poseidon3Elements deployed to: 0x050658D8A6cca8324587c84a6Bf4A082939d42C0
      Should return the last root among two equal values when search for the value (39ms)
Poseidon2Elements deployed to: 0x89d4091702776917F4EBbE28783d6872BAeE9A05
Poseidon3Elements deployed to: 0x10a3fD81Fa8341310F1E8BAdB90B6632A2F8B8aC
      Should return the last root among three equal values when search for the value
    Search in array with duplicated values and in between values
Poseidon2Elements deployed to: 0x487A726AE72b10F049dBC058860272e2c52B6591
Poseidon3Elements deployed to: 0xf4543339b914093aB3C53c23E7f0d4e6440f4870
      Should search in between the third (1st, 2nd, 3rd equal) and fourth values and return the third (49ms)
Poseidon2Elements deployed to: 0xc345d3218168522Dddabe4CaCd72BA8C7b3c8C58
Poseidon3Elements deployed to: 0x6cEC2602c6BA2957F9bA67d0b8420354Dc769FC2
      Should search in between the fifth (4th, 5th equal) and sixth values and return the fifth (44ms)

  Binary search in SMT proofs
    Zero root proofs
Poseidon2Elements deployed to: 0xCbe670b201b714158e50e97D7F47f8C9D08BF925
Poseidon3Elements deployed to: 0xa2EA240c6b4DfB56788B88b05C8e15dB150BF27a
      Should return zero proof for some search (40ms)
Poseidon2Elements deployed to: 0x1511245421d039a7bb9C9e33cc70d6Cc3EB58E3c
Poseidon3Elements deployed to: 0x33Cf99b8990efDaaB197cC506F5535596De7eC82
      Should return zero proof for some search back in time (65ms)
    Non-zero root proofs
Poseidon2Elements deployed to: 0xCac4aAc48fcbD33ED8ae651166c2992F643e252D
Poseidon3Elements deployed to: 0xE6864E47353F80Ae385f0f4Ebe5a8A72A703F55c
      Should return zero proof for some search current time (64ms)
Poseidon2Elements deployed to: 0xf542C74bfA6563815296A0b89f3B81427c1f3937
Poseidon3Elements deployed to: 0x652a67488557Cb29143fA01cC7D85a25b0AD835f
      Should return zero proof for some search current block (65ms)

  Edge cases with exceptions
Poseidon2Elements deployed to: 0x94C27176c5a35827dd0207C2f4f52EB4b27A8456
Poseidon3Elements deployed to: 0x4921e05c71ca2406C7bd94cCeBB3a3466D6E38c1
    getRootInfo() should throw when root does not exist
Poseidon2Elements deployed to: 0x60F77D4C4b161d1542d4BDe48338A7230e6a9608
Poseidon3Elements deployed to: 0xfeC8F10Cc27f8a31Dda314adA4FDdDb613359b69
    getProofByRoot() should throw when root does not exist (136ms)

  maxDepth setting tests
Poseidon2Elements deployed to: 0x6A58C4C0f358882e1E2e6e787bCF01A8b5F53453
Poseidon3Elements deployed to: 0xB6627665152307b92381Ea2d8365d1479170cF37
    Max depth should be 64
    Should increase max depth
    Should throw when decrease max depth
    Should throw when max depth is set to the same value
    Should throw when max depth is set to 0
    Should throw when max depth is set to greater than hard cap

  State transition with real verifier
Poseidon1Elements deployed to: 0x6B9806730a9F7Bd777221661Fb723b90b19bd0f1
Poseidon2Elements deployed to: 0x91329C35003E4b6aa200ba6ec8935076689 62d2B
Poseidon3Elements deployed to: 0xd0d3fBf8056b30bcC5725091Fd8f3264B8bb87E8
```

```
Poseidon4Elements deployed to: 0xf73478891477B297f86c500f543f823F83B4C400
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

     Zero-knowledge proof of state transition is not valid (285ms)
     Initial state publishing (358ms)
     Subsequent state update (366ms)

  State transition negative cases
Poseidon1Elements deployed to: 0x24C4312DbE5CA48E80bc907B3D9F7C7c6c07B3dc
Poseidon2Elements deployed to: 0xc1559a9d6086E18979fdDCA8aBA41E701cf38cdA
Poseidon3Elements deployed to: 0x964DeEbCbaDe3E054f467C4a7636aB76B77D15d5
Poseidon4Elements deployed to: 0x60EABB77F0F656d02480738ff2b988A78Ae7A6B0
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2
    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

     Old state does not match the latest state (66ms)
Poseidon1Elements deployed to: 0x498d0cc82fb1Eb789348C40c444A1a09bD70Bf23
Poseidon2Elements deployed to: 0x1E25E841D6a1dfa02863cDf654421fEc308291E1
Poseidon3Elements deployed to: 0x00F2b351820C96A62A9545f46f651ceB2a0Cc702
Poseidon4Elements deployed to: 0x92Ed31EaC79169c7903f461da102538D5c8Fa08A
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2
    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

     Old state is genesis but identity already exists (62ms)
Poseidon1Elements deployed to: 0x72Ff916A81bbe16DD80298968AbF78fb8BB4CF30
Poseidon2Elements deployed to: 0xa21bfb9398Ccd08ac5d947149A001F250FB0fC1b
Poseidon3Elements deployed to: 0x414CA05dCE5a595b11B91d6a56376948a69a636E
Poseidon4Elements deployed to: 0x9Af9C6291c313C5eDeb3a8751CA7a631FA60Abd8
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

     Old state is not genesis but identity does not yet exist
Poseidon1Elements deployed to: 0x98E7286AA72c3bE7184Fa66a48E7F894076b6Fe5
Poseidon2Elements deployed to: 0x2C57b91c3B7243b64a6AAc5B81cE72f9095a4048
Poseidon3Elements deployed to: 0x3E125a04c17146dcD14E58d6f4CDF499b9cF831a
Poseidon4Elements deployed to: 0x3502E685d1277fb6829A04a51550FB8E92708895
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

     ID should not be zero
Poseidon1Elements deployed to: 0x44F4c3accDBEfcf2dB687E1CC3f2682eb6a45cDF
Poseidon2Elements deployed to: 0x2799f26D35a4FD114bd489Bab026953C8eC06585
Poseidon3Elements deployed to: 0xa368C736FA269b52e243a8D108319EAed7EBD19D
Poseidon4Elements deployed to: 0x5fBAA10B4f7e0bCDC778d1Cb4d7d5CB2ef4636C9
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

     New state should not be zero
Poseidon1Elements deployed to: 0xFc716c638E49C555b45eCf719ff922f6Af1c4545
Poseidon2Elements deployed to: 0x2de5f06ae61aD3Df99FE41897dc44D63731bf4a0
Poseidon3Elements deployed to: 0xcCf599c804386f16c80fc457B37CC95FB128e01F
Poseidon4Elements deployed to: 0xf89e5a0F759b3b7E5435230237F29CaBE33ad5c4
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

     Should allow only one unique state per identity (106ms)

  StateInfo history
Poseidon1Elements deployed to: 0x1D6e726aDdFeDEA85856d67Bb3c3e9C7D82a78c2
Poseidon2Elements deployed to: 0xe60993305C5168b9C7934a01e503330fdC26268b
```

```
Poseidon3Elements deployed to: 0x422F99b793e913c9A5ebA97F2b130815bbb056d3
Poseidon4Elements deployed to: 0xb3CF50D16EAABE2c8C7c639D18B91773BE4cB8F8
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

     should return state history
  GIST proofs
Poseidon1Elements deployed to: 0x0e5D06416Da847D1660a4184f75b315127b28422
Poseidon2Elements deployed to: 0xB56Cd17Fb1300Ce34Ff875dd0DCbD49e912f4e91
Poseidon3Elements deployed to: 0x36dF5648822a93e5DB427239b21485251722d9Cd
Poseidon4Elements deployed to: 0xbA41968387e58eeF11c0bd0c1fb28c3F42090B25
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.
     Should be correct historical proof by root and the latest root (327ms)
Poseidon1Elements deployed to: 0xbCE8eF7324775a2796DD66AE2DF617B5Cd5E6c0C
Poseidon2Elements deployed to: 0x57956f12dCa518bA6A086106F9055A494e9F2289
Poseidon3Elements deployed to: 0x5C96A1d07824e656660EACcca33D5EfAf1399DD9
Poseidon4Elements deployed to: 0x593e900f88c52b79b703c4D2D91aac994C213B6C
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2
    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

     Should be correct historical proof by time (221ms)
Poseidon1Elements deployed to: 0x369C35F0C67a70943E552a01C8A59Cc149a06bf8
Poseidon2Elements deployed to: 0xe51B2Af6106722ead011AE2175671f01FE0D5C1d
Poseidon3Elements deployed to: 0x80F537846664F4C440e48833bdb4E796b9431885
Poseidon4Elements deployed to: 0x164b341b3C622a7211ED6ea2f8150061bE7f2A0b
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

     Should be correct historical proof by block (218ms)

  GIST root history
Poseidon1Elements deployed to: 0xC3C17b5c68272720A0774bac8980e711d9eCa411
Poseidon2Elements deployed to: 0xb95C156c710406F73beFb6464Cb1631A6146Fe16
Poseidon3Elements deployed to: 0x7374947f4992F480b28ae6f50097f86Eb79b2446
Poseidon4Elements deployed to: 0x39Eb869Da7910282e60E5539Dd7CB037a7f1B813
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

     Should search by block and by time return same root (215ms)
Poseidon1Elements deployed to: 0xb0bC99F24560e66914EEd19573c26fe6733c9FB2
Poseidon2Elements deployed to: 0xCf7115125Ec7b4a13Eb6Ba000eE21BB1558De3b3
Poseidon3Elements deployed to: 0xAf1ED42528f21b9c5C5d9E6D3C782FF5D96246eE
Poseidon4Elements deployed to: 0xAb8417fc62aAd588615eD995C5f1cbA80a71c773
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

     Should have correct GIST root transitions info (113ms)

  Set Verifier
Poseidon1Elements deployed to: 0xE6275C3b51b78Feb10a2127c992e8d97174b15ef
Poseidon2Elements deployed to: 0x0AB6fC6Af4C6cD9e7Fe4492b51DCCc3cE465D484
Poseidon3Elements deployed to: 0xc676AB39cf90C6D2452964F87d10fd093Ced87C7
Poseidon4Elements deployed to: 0xEE8Cf3f5Add2868A827c894b02feb362Bd8B24e9
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.
    Should set verifier (146ms)
Poseidon1Elements deployed to: 0x25BC4dB806284AC8e0e84de1abD72909bd1B5D70
Poseidon2Elements deployed to: 0x8Cc433169E8ABcFfa75365E06C2c698FbE83bc31
Poseidon3Elements deployed to: 0x1285a8533AD61718B9337Eca7a7B9c6c9FdBAaDC
Poseidon4Elements deployed to: 0x4360C3d87b39826fc50703D0d12b44c8b445C55D
```

```
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2
    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

     Should not set verifier if not owner (144ms)
Poseidon1Elements deployed to: 0x4018AaC72b92872bDc1117EA3B046ABf3B39276b
Poseidon2Elements deployed to: 0x79D33C1Fabd19197D13D9cEEa294839a766b5eCD
Poseidon3Elements deployed to: 0xf907eEb74FD8Bac40d05c2cC14ce1BFa7584Fe40
Poseidon4Elements deployed to: 0x0AED87875EE10BFc23519a37b113945E5fc5930b
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2
    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

     Should allow verifier zero address to block any state transition (162ms)
  Get StateV2 old Contract and migrate to latest version
    - Check migration
  Negative tests
     getStateInfoByID: should be reverted if identity does not exist
     getStateInfoHistoryById: should be reverted if identity does not exist
     getStateInfoHistoryLengthById: should be reverted if identity does not exist
     getStateInfoByIdAndState: should be reverted if state does not exist
     Zero timestamp and block should be only in the first identity state (39ms)
 StateInfo history
     should return state history (49ms)
     should be reverted if length is zero
     should be reverted if length limit exceeded
     should be reverted if startIndex is out of bounds
     should not revert if startIndex + length >= historyLength

  State history duplicates
     comprehensive check (147ms)
     should revert if length is zero
     should revert if length limit exceeded
     should revert if out of bounds

  migration test automated
no output.json file found for migration test
upgrade test skipped (no old contract address found)
     test state contract migration

  Atomic MTP Validator
Poseidon1Elements deployed to: 0x1925e621f3e4dc6386C98Bf7195041010fa662B9
Poseidon2Elements deployed to: 0xEe5bEDB9Ed80b35a8d3279A6Ba09942E0e173c3f
Poseidon3Elements deployed to: 0x73bEa1Ed02B705092BD57c02b3bc863f8354192e
Poseidon4Elements deployed to: 0xB1B8ca64D381fB290eCE8051b3f548B6ad2FE72c
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

Validator Verifier Wrapper deployed to: 0x4a5b24A16be05b93f33B95f92c09D45B9875A746
CredentialAtomicQueryMTPValidator deployed to: 0x3E9aCf759F5008478926b3e454fddC2478957C58
     Validate Genesis User State. Issuer Claim IdenState is in Chain. Revocation State is in Chain (649ms)
Poseidon1Elements deployed to: 0xa128F47514E078d121e02602C07722eE10a5BbA5
Poseidon2Elements deployed to: 0xe2E590924f89c4c76575431fbaa71F3393a9eb45
Poseidon3Elements deployed to: 0x009686Dd50925C2554734104636b799b31B9bE62
Poseidon4Elements deployed to: 0x016129fD9767f6bD296e46E464CcBddfe8Ce57B7
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2
    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

Validator Verifier Wrapper deployed to: 0x9516B54C07fBa7AaBF139388d6ec43759d0b691C
CredentialAtomicQueryMTPValidator deployed to: 0x1471ADdCbE6Db6A5a6e8997bcEF372C4a0834fb5
     Validation of proof failed (322ms)
Poseidon1Elements deployed to: 0xcD8311cb1EB97582E543962B360600F0fB14a5f5
Poseidon2Elements deployed to: 0xD3624829750dC601F66ca8020485711108D9EAcC
Poseidon3Elements deployed to: 0xA1cA92deF3629Acde4982435B99690FDF07BE6DC
Poseidon4Elements deployed to: 0x72cA51E345f5f56156B61A117Ac870a1b0fd7B79
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.
```

```
Validator Verifier Wrapper deployed to: 0xc5b0BDce544e2fDEB8f3aAf15014D4434390a452
CredentialAtomicQueryMTPValidator deployed to: 0x880819ff3C5cF86C05c33E808c38ce19A6b2f9Fb
     User state is not genesis but latest (1088ms)
Poseidon1Elements deployed to: 0xDF5c32c0d5A82fC3cDE4B609D4a85e05FC31525A
Poseidon2Elements deployed to: 0x7B8042866D02bae8878b5462A7702e211685e440
Poseidon3Elements deployed to: 0xec6D6Da6015f9F133bF029959f92FA078526F031
Poseidon4Elements deployed to: 0x7A3d255CB5471488aE485dC635bfA39cd0d63158
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

Validator Verifier Wrapper deployed to: 0x8E5cCA5f2058041Aac1D3e7E9d0ec4F5cC9eB9c6
CredentialAtomicQueryMTPValidator deployed to: 0x226582f6432f41910b4Ff9eB6816e41957C4140c
     The non-revocation issuer state is not expired (is not too old) (1437ms)
Poseidon1Elements deployed to: 0xe11EBc54344A29375C1825bb2708F78721484c46
Poseidon2Elements deployed to: 0xE6061892C479eBaC54234AcB6b4Ef880E3eC5f7F
Poseidon3Elements deployed to: 0xEe72Abc172b407B1fa06E8527F30FC49C9A25bB4
Poseidon4Elements deployed to: 0xd98336043E140FD0B5c2E9364eB16834fB7A1a3E
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

Validator Verifier Wrapper deployed to: 0xB3Fcbe50040d6D82fa55476f91cA7BAbE0FAD07A
CredentialAtomicQueryMTPValidator deployed to: 0x6eb879920a3d85Cc69E333CB47397B2d151FAfA6
     The non-revocation issuer state is expired (old enough) (1140ms)

  Atomic Sig Validator
Poseidon1Elements deployed to: 0xBD018fdF65A5b0DbDACb988494AdAd807cFe6D60
Poseidon2Elements deployed to: 0x6Ab803565C8C970987D6f7978E0B3189C70749Db
Poseidon3Elements deployed to: 0x9Fe0267384B1874307A1B8331dF816a756D38E69
Poseidon4Elements deployed to: 0x8a1b2E7f0589F77e437527eC2699eB4caE6B49A8
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

Validator Verifier Wrapper deployed to: 0xE27fe0359C5b675D6b3b4a3dc011c23D9F1621e3
CredentialAtomicQuerySigValidator deployed to: 0x9225686297925fb12500d5DccB3740ca70052181
     Validate Genesis User State. Issuer Claim IdenState is in Chain. Revocation State is in Chain (653ms)
Poseidon1Elements deployed to: 0x4C22c23cF94742Bd5d205e0740f4994F06e495fC
Poseidon2Elements deployed to: 0xF46CC4E15dc5D05BcD6B02CAea357CE5aB6aA48C
Poseidon3Elements deployed to: 0xAe23268301D5E350DafC3f72F63e8E6345136a84
Poseidon4Elements deployed to: 0x2Ad11d6B51272a930cdCc31274B075aEA3a2db1c
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.
Validator Verifier Wrapper deployed to: 0x0075176e0352E9D1Df281709C0B314b731D8c335
CredentialAtomicQuerySigValidator deployed to: 0x85171299fB8dD90E1DE1D83177c98a523C00C996
     Validation of proof failed (322ms)
Poseidon1Elements deployed to: 0x36058b450b2aB78D4ce3d0ABe21c3f145361996c
Poseidon2Elements deployed to: 0x3aE0Bc70f8b46f0A27c4666F5A19DF5056B10E79
Poseidon3Elements deployed to: 0xC2bC40f6eA7D0ffE81Ca46a013E87Ba8C7877C09
Poseidon4Elements deployed to: 0xb7766b758AC5329BB77f435ef50Fb3bCc55BCeCB
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

Validator Verifier Wrapper deployed to: 0x19dEC0735334fd8294c0FE27A5c6999BfaaF4baF
CredentialAtomicQuerySigValidator deployed to: 0x11B8bE0eF622EE0e6F9377d67ABb3a9C42c994Db
     User state is not genesis but latest (1053ms)
Poseidon1Elements deployed to: 0x1d1490d7386DEF2435099Abd2daE046EcE55237C
Poseidon2Elements deployed to: 0x288f09328B087d4751cA78Fc9f6DFb8e5F4b9e6D
Poseidon3Elements deployed to: 0x3e5563385Dd0a71c9FC0E6b38321736a47Dada6e
Poseidon4Elements deployed to: 0xBdEd7A40a7088eef12392E9bcdD387AAC8bF9972
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2
```

```
    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

Validator Verifier Wrapper deployed to: 0x25b5C4A3110139337A699892A3D321159ACf7C21
CredentialAtomicQuerySigValidator deployed to: 0x6BF29Bb14E44f9E0E6946D1C21144BcEbE69d55d
     The non-revocation issuer state is not expired (is not too old) (1431ms)
Poseidon1Elements deployed to: 0xd293f0dEECdaEf060a50eF44f936b53AD1E1998E
Poseidon2Elements deployed to: 0xE3d6cD97E7c19381D8cf774F8570d51B33ada9Ed
Poseidon3Elements deployed to: 0x095DaBe9e17916AF5924822e569F314E07d935F3
Poseidon4Elements deployed to: 0x9c346441419008d0d0Dc34696A20f55F4E504155
Warning: Potentially unsafe deployment of contracts/state/StateV2.sol:StateV2

    You are using the `unsafeAllow.external-library-linking` flag to include external libraries.
    Make sure you have manually checked that the linked libraries are upgrade safe.

Validator Verifier Wrapper deployed to: 0xe2de18FEc790C4D61EBDB30b9609E5F4C807Aa8F
CredentialAtomicQuerySigValidator deployed to: 0xd31B2168149Fa599666F7FfE75fec2682B2F510f
     The non-revocation issuer state is expired (old enough) (1102ms)
  198 passing (1m)
  1 pending
```

10.2 Coverage Test

```
$ npx hardhat coverage

---------------------------------------|----------|----------|----------|----------|----------------|
File                                   |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
---------------------------------------|----------|----------|----------|----------|----------------|
 interfaces/                           |      100 |      100 |      100 |      100 |                |
  ICircuitValidator.sol                |      100 |      100 |      100 |      100 |                |
  IOnchainCredentialStatusResolver.sol |      100 |      100 |      100 |      100 |                |
  IState.sol                           |      100 |      100 |      100 |      100 |                |
  IStateTransitionVerifier.sol         |      100 |      100 |      100 |      100 |                |
  IVerifier.sol                        |      100 |      100 |      100 |      100 |                |
  IZKPVerifier.sol                     |      100 |      100 |      100 |      100 |                |
 lib/                                  |    85.75 |    75.74 |    83.72 |    87.29 |                |
  ArrayUtils.sol                       |      100 |      100 |      100 |      100 |                |
  ClaimBuilder.sol                     |      100 |    93.75 |      100 |      100 |                |
  GenesisUtils.sol                     |       80 |     37.5 |    72.73 |    84.85 |... ,75,136,145 |
  IdentityBase.sol                     |       72 |        0 |    94.74 |       72 |... 221,225,236 |
  OnChainIdentity.sol                  |    82.35 |    61.11 |    94.12 |    82.22 |... 95,97,98,99 |
  Pairing.sol                          |       70 |       50 |    55.56 |    71.43 |... 179,180,181 |
  Poseidon.sol                         |    73.33 |      100 |    28.57 |    83.33 |    59,71,75,79 |
  SmtLib.sol                           |    95.83 |    82.89 |      100 |    96.69 |... 647,675,699 |
  StateLib.sol                         |      100 |    93.75 |      100 |      100 |                |
  verifierMTP.sol                      |    53.33 |     37.5 |    66.67 |    72.97 |... 174,175,177 |
  verifierMTPWrapper.sol               |       75 |       50 |      100 |    85.71 |             36 |
  verifierSig.sol                      |    53.33 |     37.5 |    66.67 |    72.97 |... 174,175,177 |
  verifierSigWrapper.sol               |       75 |       50 |      100 |    85.71 |             36 |
  verifierV2.sol                       |      100 |       75 |      100 |      100 |                |
 state/                                |    92.98 |       75 |    89.66 |    93.75 |                |
  StateV2.sol                          |    92.98 |       75 |    89.66 |    93.75 | 89,148,328,339 |
 validators/                           |    81.58 |    57.14 |    66.67 |    86.54 |                |
  CredentialAtomicQueryMTPValidator.sol |       50 |      100 |    33.33 |    77.78 |          11,15 |
  CredentialAtomicQuerySigValidator.sol |       50 |      100 |    33.33 |    77.78 |          11,15 |
  CredentialAtomicQueryValidator.sol    |       90 |    57.14 |      100 |    91.18 |       81,95,99 |
 verifiers/                            |        0 |        0 |        0 |        0 |                |
  ZKPVerifier.sol                      |        0 |        0 |        0 |        0 |... 102,103,107 |
---------------------------------------|----------|----------|----------|----------|----------------|
All files                              |    83.73 |    70.44 |    80.23 |    85.02 |                |
---------------------------------------|----------|----------|----------|----------|----------------|

> Istanbul reports written to ./coverage/ and ./coverage.json
```

10.3 Slither

All the relevant issues raised by Slither have been incorporated into the issues described in this report.

11 About Nethermind

Nethermind is a Blockchain Research and Software Engineering company. Our work touches every part of the web3 ecosystem - from layer 1 and layer 2 engineering, cryptography research, and security to application-layer protocol development. We offer strategic support to our institutional and enterprise partners across the blockchain, digital assets, and DeFi sectors, guiding them through all stages of the research and development process, from initial concepts to successful implementation.

We offer security audits of projects built on EVM-compatible chains and Starknet. We are active builders of the Starknet ecosystem, delivering a node implementation, a block explorer, a Solidity-to-Cairo transpiler, and formal verification tooling. Nethermind also provides strategic support to our institutional and enterprise partners in blockchain, digital assets, and decentralized finance (DeFi). In the next paragraphs, we introduce the company in more detail.

Blockchain Security: At Nethermind, we believe security is vital to the health and longevity of the entire Web3 ecosystem. We provide security services related to Smart Contract Audits, Formal Verification, and Real-Time Monitoring. Our Security Team comprises blockchain security experts in each field, often collaborating to produce comprehensive and robust security solutions. The team has a strong academic background, can apply state-of-the-art techniques, and is experienced in analyzing cutting-edge Solidity and Cairo smart contracts, such as ArgentX and StarkGate (the bridge connecting Ethereum and StarkNet). Most team members hold a Ph.D. degree and actively participate in the research community, accounting for 240+ articles published and 1,450+ citations in Google Scholar. The security team adopts customer-oriented and interactive processes where clients are involved in all stages of the work.

Blockchain Core Development: Our core engineering team, consisting of over 20 developers, maintains, improves, and upgrades our flagship product - the Nethermind Ethereum Execution Client. The client has been successfully operating for several years, supporting both the Ethereum Mainnet and its testnets, and now accounts for nearly a quarter of all synced Mainnet nodes. Our unwavering commitment to Ethereum's growth and stability extends to sidechains and layer 2 solutions. Notably, we were the sole execution layer client to facilitate Gnosis Chain's Merge, transitioning from Aura to Proof of Stake (PoS), and we are actively developing a full-node client to bolster Starknet's decentralization efforts. Our core team equips partners with tools for seamless node set-up, using generated docker-compose scripts tailored to their chosen execution client and preferred configurations for various network types.

DevOps and Infrastructure Management: Our infrastructure team ensures our partners' systems operate securely, reliably, and efficiently. We provide infrastructure design, deployment, monitoring, maintenance, and troubleshooting support, allowing you to focus on your core business operations. Boasting extensive expertise in Blockchain as a Service, private blockchain implementations, and node management, our infrastructure and DevOps engineers are proficient with major cloud solution providers and can host applications in-house or on clients' premises. Our global in-house SRE teams offer 24/7 monitoring and alerts for both infrastructure and application levels. We manage over 5,000 public and private validators and maintain nodes on major public blockchains such as Polygon, Gnosis, Solana, Cosmos, Near, Avalanche, Polkadot, Aptos, and StarkWare L2. Sedge is an open-source tool developed by our infrastructure experts, designed to simplify the complex process of setting up a proof-of-stake (PoS) network or chain validator. Sedge generates docker-compose scripts for the entire validator set-up based on the chosen client, making the process easier and quicker while following best practices to avoid downtime and being slashed.

Cryptography Research: At Nethermind, our Cryptography Research team is dedicated to continuous internal research while fostering close collaboration with external partners. The team has expertise across a wide range of domains, including cryptography protocols, consensus design, decentralized identity, verifiable credentials, Sybil resistance, oracles, and credentials, distributed validator technology (DVT), and Zero-knowledge proofs. This diverse skill set, combined with strong collaboration between our engineering teams, enables us to deliver cutting-edge solutions to our partners and clients.

Smart Contract Development & DeFi Research: Our smart contract development and DeFi research team comprises 40+ world-class engineers who collaborate closely with partners to identify needs and work on value-adding projects. The team specializes in Solidity and Cairo development, architecture design, and DeFi solutions, including DEXs, AMMs, structured products, derivatives, and money market protocols, as well as ERC20, 721, and 1155 token design. Our research and data analytics focuses on three key areas: technical due diligence, market research, and DeFi research. Utilizing a data-driven approach, we offer in-depth insights and outlooks on various industry themes.

Our suite of L2 tooling: Warp is Starknet's approach to EVM compatibility. It allows developers to take their Solidity smart contracts and transpile them to Cairo, Starknet's smart contract language. In the short time since its inception, the project has accomplished many achievements, including successfully transpiling Uniswap v3 onto Starknet using Warp.

- **Voyager** is a user-friendly Starknet block explorer that offers comprehensive insights into the Starknet network. With its intuitive interface and powerful features, Voyager allows users to easily search for and examine transactions, addresses, and contract details. As an essential tool for navigating the Starknet ecosystem, Voyager is the go-to solution for users seeking in-depth information and analysis;
- **Horus** is an open-source formal verification tool for StarkNet smart contracts. It simplifies the process of formally verifying Starknet smart contracts, allowing developers to express various assertions about the behavior of their code using a simple assertion language;
- **Juno** is a full-node client implementation for Starknet, drawing on the expertise gained from developing the Nethermind Client. Written in Golang and open-sourced from the outset, Juno verifies the validity of the data received from Starknet by comparing it to proofs retrieved from Ethereum, thus maintaining the integrity and security of the entire ecosystem.

Learn more about us at nethermind.io.

General Advisory to Clients

As auditors, we recommend that any changes or updates made to the audited codebase undergo a re-audit or security review to address potential vulnerabilities or risks introduced by the modifications. By conducting a re-audit or security review of the modified codebase, you can significantly enhance the overall security of your system and reduce the likelihood of exploitation. However, we do not possess the authority or right to impose obligations or restrictions on our clients regarding codebase updates, modifications, or subsequent audits. Accordingly, the decision to seek a re-audit or security review lies solely with you.

Disclaimer

This report is based on the scope of materials and documentation provided by you to Nethermind in order that Nethermind could conduct the security review outlined in **1. Executive Summary** and **2. Audited Files**. The results set out in this report may not be complete nor inclusive of all vulnerabilities. Nethermind has provided the review and this report on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. This report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on this report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, Nethermind disclaims any liability in connection with this report, its content, and any related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Nethermind does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Nethermind will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.