Security Review Report
NM-0379 Privado ID

NETHERMIND SECURITY

(April 4, 2025)

Contents

1 Executive Summary

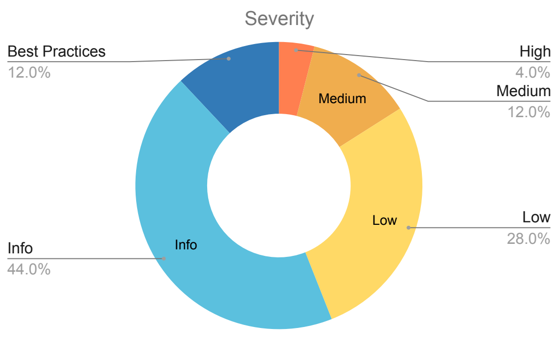
This document outlines the security review conducted by Nethermind for the Privado.iD protocol. Privado.iD is a blockchain-native identity system. It gives the power to build trusted and secure relationships between users and dApps, following the principles of self-sovereign identity and privacy by default.

The **Privado.iD** protocol has introduced **Version V3**, which incorporates new validators and enables cross-chain operations. This review builds on previous audits, namely `NM-0069` and `NM-0113`. The current assessment focuses primarily on verifiers, validators, and cross-chain interactions as well as the payment contracts. In addition, previously audited contracts and libraries were reviewed as part of a **difference audit**. The audit also included an evaluation of the **deployment scripts** used to implement the relevant contracts.

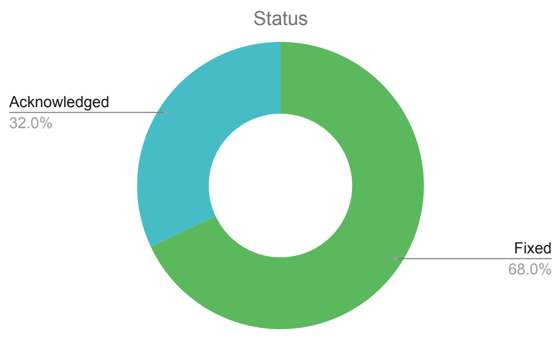
The audited code comprises 4,591 lines of code written in Solidity language.

The audit was performed using (a) manual analysis of the codebase, (b) automated analysis tools, and (c) creation of test cases. **Along this document, we report** 25 points of attention, where one is classified as `High`, three are classified as `Medium`, seven are classified as `Low`, and fourteen are classified as `Informational` or `Best Practice`. The issues are summarized in Fig. 1.

This document is organized as follows. Section 2 and Section 3 present the files in the scope and diff audit files, respectively. Section 4 summarizes the issues. Section 5 presents the system overview. Section 6 provides the BN254 security review. Section 7 discusses the risk rating methodology. 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 describing Nethermind services and products. Appendix describes the review of the deployment scripts. Section Reference lists all references cited in Section 6.



(a)



(b)

Fig. 1: Distribution of issues: Critical (0), **High** (1), **Medium** (3), **Low** (7), **Undetermined** (0), **Informational** (11), **Best Practices** (3).
Distribution of status: Fixed (17), **Acknowledged** (8), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	March 12, 2025
Response from Client	Regular responses during audit engagement
Final Report	April 4, 2025
Repository	iden3/contracts
Commit (Audit - Section 2-I)	f60a6a80c521f3052274c6be7c2753ff300c81bd
Final Commit	ae33c3175410bb3626628da7673afa7e4dac8afa
Documentation	README.md
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	cross-chain/CrossChainProofValidator.sol	189	27	14.3%	31	247
2	payment/MCPayment.sol	289	110	38.1%	41	440
3	payment/VCPayment.sol	217	41	18.9%	32	290
4	verifiers/UniversalVerifier.sol	131	73	55.7%	27	231
5	verifiers/Verifier.sol	677	182	26.9%	119	978
6	verifiers/RequestDisableable.sol	45	11	24.4%	10	66
7	verifiers/EmbeddedVerifier.sol	32	27	84.4%	7	66
8	verifiers/RequestOwnership.sol	35	10	28.6%	7	52
9	verifiers/ValidatorWhitelist.sol	62	11	17.7%	12	85
10	validators/auth/EthIdentityValidator.sol	57	31	54.4%	15	103
11	validators/auth/AuthV2Validator.sol	140	71	50.7%	30	241
12	validators/request/CredentialAtomicQuerySigV2Validator.sol	48	22	45.8%	9	79
13	validators/request/CredentialAtomicQueryValidatorBase.sol	181	85	47.0%	35	301
14	validators/request/CredentialAtomicQueryV2ValidatorBase.sol	141	38	27.0%	24	203
15	validators/request/CredentialAtomicQueryMTPV2Validator.sol	48	22	45.8%	9	79
16	validators/request/CredentialAtomicQueryV3Validator.sol	245	45	18.4%	33	323
17	validators/request/LinkedMultiQueryValidator.sol	172	43	25.0%	29	244
18	interfaces/INonMerklizedIssuer.sol	30	34	113.3%	7	71
19	interfaces/IIdentifiable.sol	4	7	175.0%	1	12
20	interfaces/IRHSStorage.sol	5	10	200.0%	2	17
21	interfaces/IState.sol	102	151	148.0%	19	272
22	interfaces/ICrossChainProofValidator.sol	11	14	127.3%	3	28
	Total	**2861**	**1065**	**37.2%**	**502**	**4428**

3 Diff Audited Files

Those files were audited in the previous audits: `NM-0069` and `NM-0113`. In the current audit only the changes to those files were checked.

	Contract	LoC	Comments	Ratio	Blank	Total
1	state/State.sol	379	225	59.4%	64	668
2	lib/SmtLib.sol	438	191	43.6%	73	702
3	lib/ArrayUtils.sol	18	10	55.6%	3	31
4	lib/PrimitiveTypeUtils.sol	84	73	86.9%	24	181
5	lib/IdentityLib.sol	261	138	52.9%	35	434
6	lib/StateLib.sol	170	135	79.4%	31	336
7	lib/IdentityBase.sol	151	124	82.1%	32	307
8	lib/GenesisUtils.sol	39	16	41.0%	13	68
9	lib/Poseidon.sol	68	2	2.9%	15	85
10	interfaces/IOnchainCredentialStatusResolver.sol	32	37	115.6%	5	74
11	interfaces/IVerifier.sol	90	148	164.4%	28	266
	Total	**1730**	**1099**	**63.5%**	**323**	**3152**

4 Summary of Issues

	Finding	Severity	Update
1	Issuer may burn another issuer's funds	High	Fixed
2	Request creation may be front-run	Medium	Fixed
3	The withdrawToAllIssuers(...) may be blocked	Medium	Fixed
4	Verifier can call nonactive AuthValidators	Medium	Fixed
5	Everyone can disable and enable an auth type in UniversalVerifier	Low	Fixed
6	Lack of check for V3 query in request group	Low	Fixed
7	Responses are valid for disabled requests	Low	Acknowledged
8	The `groupID` is not bound to the group contents	Low	Fixed
9	The creation of MultiRequest may be front-run	Low	Fixed
10	`setPaymentValue(...)` can reset totalValue	Low	Fixed
11	payERC20Permit is vulnerable to DOS via front-running	Low	Fixed
12	Checking the validity of the response group	Info	Acknowledged
13	LinkID may be shared between groups	Info	Acknowledged
14	LinkedMultiQuery proof may have `linkID=0`	Info	Fixed
15	Low Security from BN254 Curve	Info	Acknowledged
16	Redundant storage write	Info	Fixed
17	The `getRequestProofStatus(...)` does not account for group validity	Info	Acknowledged
18	The `withdrawToAllIssuers(...)` is not optimal	Info	Fixed
19	The cross-chain proofs may be provided to the same chain	Info	Acknowledged
20	Unused skipClaimRevocationCheck	Info	Acknowledged
21	Using CREATE2 on ZKsync Era	Info	Acknowledged
22	`MCPayment` can't be used with USDT	Info	Fixed
23	Apply reentrancy protection	Best Practices	Fixed
24	Disable the initializer of the implementation	Best Practices	Fixed
25	Lack of `_disableInitializers(...)`	Best Practices	Fixed

5 System Overview

In this section, we present the new parts of the protocol that were the main focus of the security review. For a more in-depth introduction to the protocol, please refer to the official documentation of Privado ID and iden3.

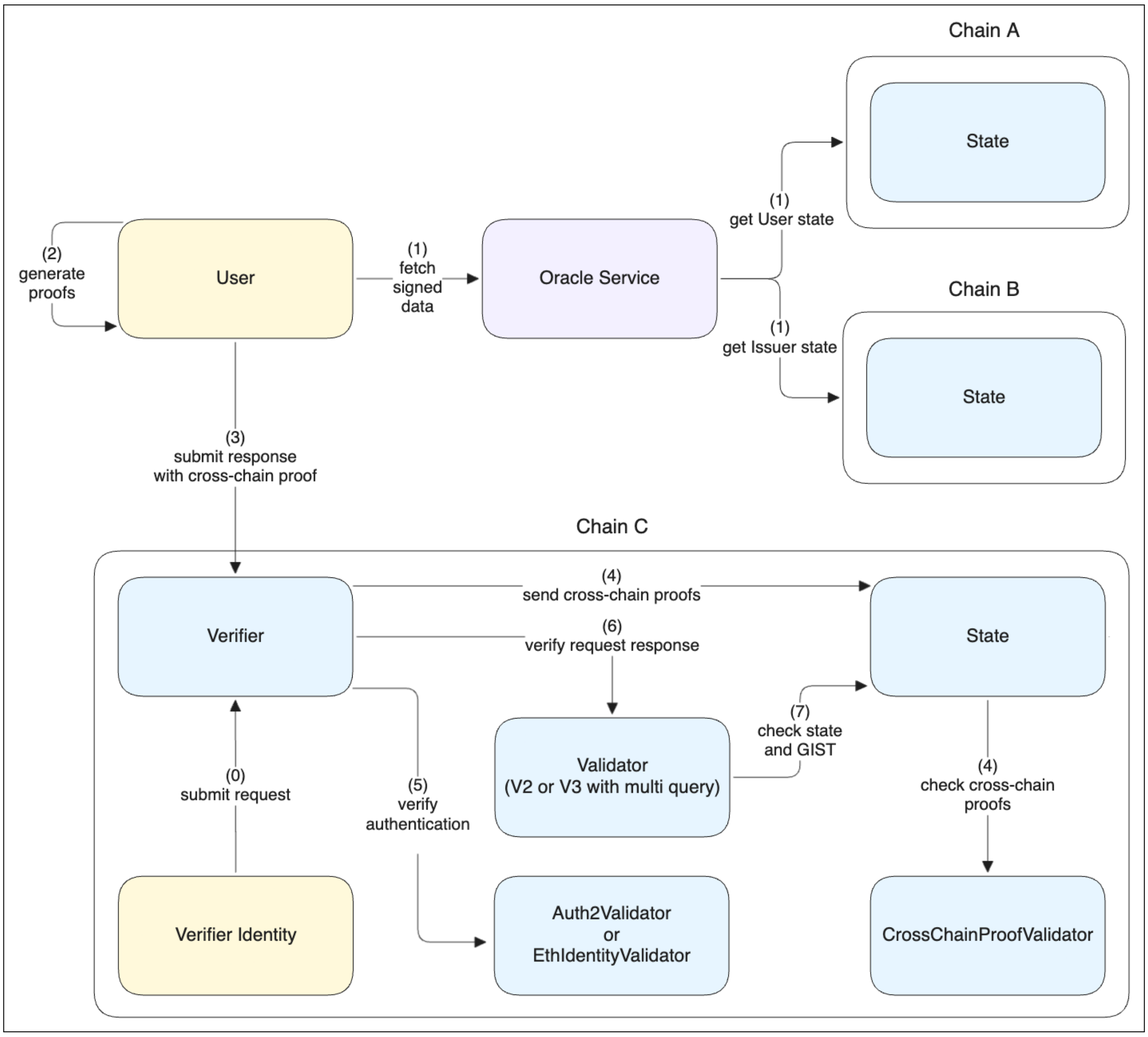

Chain A
State
(1) get User state
(2) generate proofs
User
(1) fetch signed data
Oracle Service
Chain B
(1) get Issuer state
State
(3) submit response with cross-chain proof
Chain C
Verifier
(4) send cross-chain proofs
(6) verify request response
State
(7) check state and GIST
Validator (V2 or V3 with multi query)
(0) submit request
(5) verify authentication
(4) check cross-chain proofs
Verifier Identity
Auth2Validator or EthIdentityValidator
CrossChainProofValidator

Fig. 2: Protocol flow diagram.

5.1 Cross-chain proofs

Cross-chain proofs allow the identities to prove the claims between chains. To issue a credential (prove the claim), the User can call the centralized oracle service that would fetch the data (state and GIST timestamps) of the User and the Issuer from the source chains, sign it, and return it to the User. The User then generates a zk-proof and submits it to the destination chain. On the destination chain, the Verifier contract would first check the oracle signature and submit the state and GIST to the local State contract. Next, the Verifier contract checks zk-proofs and additional properties using authentication and query validator contracts. If all requirements are satisfied, the proof of claim is submitted as an answer to a query.

5.2 V3 validator

The previous iteration of the protocol (V2) contained two different credentials query validators:

- `CredentialAtomicQueryMTPV2Validator` - validates that the claim is in the Claims Tree of the Issuer
- `CredentialAtomicQuerySigV2Validator` - validates that the claim was signed by the private key, which is in the Claims Tree of the Issuer

In the previous iteration, the protocol required two separate validator contracts because different circuits were used depending on the type of claim issuance. The `CredentialAtomicQueryV3Validator` was developed with the introduction of V3. This new validator enables proof verification regardless of the type of claim issuance, as a single circuit can now handle both cases.

In addition, V3 introduces the `LinkedMultiQueryValidator`, which allows for multiple queries (up to 10) on the same claim. However, this validator should only be used with `CredentialAtomicQueryV3Validator`, as it does not independently verify essential claim properties beyond the queried data.

5.3 Verifiers

The on-chain verification provided by Privado.iD enables applications to validate users' credentials directly within a Smart Contract. This verification process ensures privacy, allowing credentials to be verified without exposing personal information about the User (prover).

Their verification functionalities can be applied in several scenarios, such as:

- *Token Airdrops:* Distributing tokens exclusively to human-verified accounts.
- *DAO Governance:* Restricting voting rights to verified DAO members.
- *Geo-Restrictions:* Preventing airdrops for users from specific regions.
- *KYC-Based Access:* Enable trading only for accounts that have passed KYC verification.

5.3.1 Two Approaches to On-Chain Verification

Privado.iD provides two ways to integrate ZKP verification into smart contracts. Fig. 3 illustrates the main contracts for the verification process. `UniversalVerifier` inherits contracts `RequestOwnership`, `RequestDisableable`, `ValidatorWhitelist`, and the OpenZeppelin abstract contract Ownable2StepUpgradeable. `EmbeddedVerifier` inherits only Verifier and Ownable2StepUpgradeable.

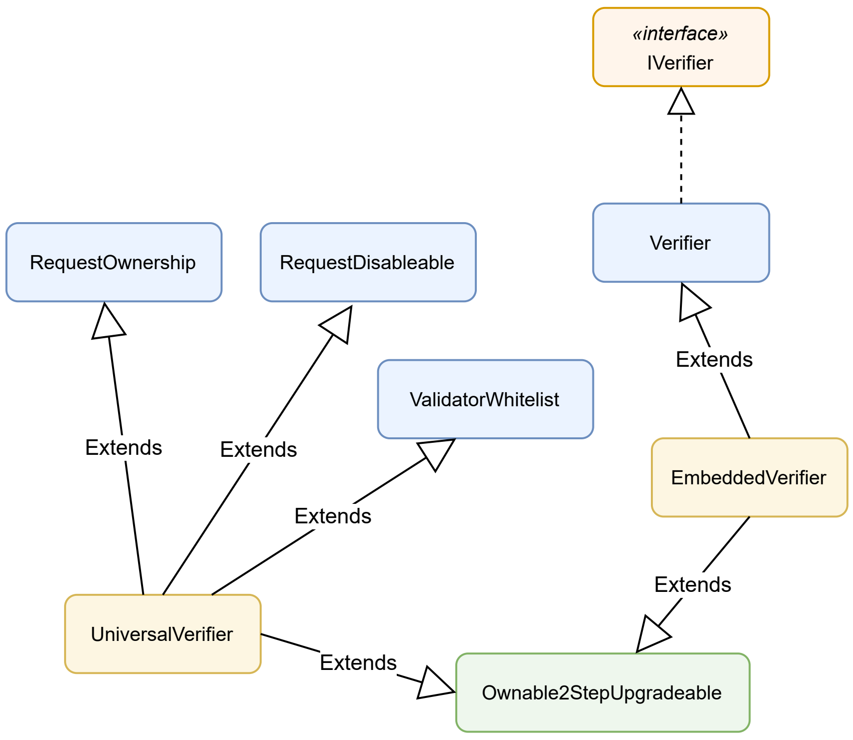


Fig. 3: Verifier contracts.

Ⓐ. `EmbeddedVerifier:` This contract is directly integrated into the client's smart contract. The verification results are stored within the state of the client contract. All proof responses must be resubmitted for each new contract requiring on-chain verification.

Ⓑ. `UniversalVerifier:` Deployed as a standalone contract, this contract acts as a central registry for verification of the proofs. Once a proof is submitted for a specific proof request, it can be reused across multiple client contracts without requiring re-submission.

Both approaches share the same parent class and implement the `IVerifier` interface, which defines methods for setting, retrieving, and submitting proof responses.

5.4 Payment contracts

The payment contracts can be utilized to compensate the Issuer for issuing the claims. There are two payment contracts:

– `VCPayment` - the contract owner defines the payment data (amount, withdrawal address) that can be used to compensate the Issuer. The User can pay with native currency, which is marked in the `payments` mapping. The protocol owner may set a percentage of the payment to the owner. The funds are stored in the contract until the Issuer or the owner withdraws.

– `MCPayment` - the `paymentData` that contains amount, recipient, nonce, and expiration timestamp are signed by the Issuer to ensure the correctness of the transaction. Similarly to `VCPayment`, the amount can be split between the Issuer and the protocol owner, the paid amount is marked in `isPaid` mapping, and the native currency is in the contract until the Issuer or owner withdraws. Apart from the native currency, the payment may be made in `ERC20`, which is directly transferred to the Issuer.

6 Analysis of the Security of the BN254 Elliptic Curve

The security of the **BN254** elliptic curve in cryptographic protocols has weakened from the initial 128-bit security level to between 96-bit and 110-bit [14, 4, 6] due to the improvement of computing discrete logarithms using the Tower Number Field Sieve (TNS) method [16].

Due to the increased ability to solve discrete logarithms in Finite Fields, and due to the Groth16 proof system using elliptic curve pairings (which rely on discrete logarithms to prove constraints), the overall security of the proof system utilized has been lowered. For this reason, and as it has been flagged on the safe curves list as false for security [3], we recommend the usage of a more secure curve and one that is compliant with NIST's 112-bit security level [15].

We recommend considering **BLS12_381** [1] due to its security level and native Circom/SnarkJS implementation.

6.1 Background

We present some preliminaries to better understand the following sections.

6.1.1 Number Theory and Cryptographic Hardness

In the following sections, we assume a basic understanding of cryptography, modular arithmetic, group theory, and elliptic curves. However, for clarity, we provide Lemma 7.13 from [12] to demonstrate the properties of a group:

> Let $\mathbb{G}$ be a group and $a, b, c \in \mathbb{G}$. If $ac = bc$, then $a = b$. In particular, if $ac = c$ then a is the identity in $\mathbb{G}$.

In particular, a cyclic group is a group that can be generated by a single element. This means that if there exists an element $g \in \mathbb{G}$ of order $m = |\mathbb{G}|$, we call $\mathbb{G}$ a cyclic group and g the generator. Then every element $h \in \mathbb{G}$ is equal to g^x for some $x \in \{0, \ldots, m-1\}$.

Ⓐ. **The Discrete Logarithm Problem:** An important intractable problem is known as the discrete logarithm problem (DLP). Many applications (including zero-knowledge proofs) are reduced to the ability (or feasibility) to solve this problem.

ⓐ.*Discrete Log:* Given a cyclic group $\mathbb{G}$ of order q with generator g, such that $\{g^0, g^1, \ldots, g^{q-1}\} = \mathbb{G}$, there exists a unique $x \in \mathbb{Z}_q$ such that $g^x = h$ for $h \in \mathbb{G}$. We call x the discrete logarithm of h with respect to g.

ⓑ. *Discrete Logarithm Problem (DLP):* The DLP in a cyclic group $\mathbb{G}$ with a given generator g and a random element $h \in \mathbb{G}$ is to compute:

$$x = \log_g h \tag{1}$$

Following the D.Log experiment $\text{DLog}_{\mathcal{A},\mathcal{G}}(n)$ from [12], we say that the DLP is a hard problem relative to a polynomial-time algorithm $\mathcal{G}$ if for a probabilistic polynomial adversary $\mathcal{A}$ and security parameter n, there exists a negligible function negl such that:

$$\Pr(\text{DLog}_{\mathcal{A},\mathcal{G}}(n) = 1) \le \text{negl}(n) \tag{2}$$

ⓒ. *Elliptic Curve Discrete Logarithm Problem (ECDLP):* in the above explanation we refer to the modular group, but in many cases the group we are referring to is an elliptic curve. The ECDLP is the building block for elliptic curve cryptography and pairing-based cryptography. It is defined as follows

Given an elliptic curve E over a finite field $\mathbb{F}_q$, where $q = p^n$ and p is a prime, the ECDLP is to find an integer a given points $P, Q \in E(\mathbb{F}_q)$ such that:

$$Q = aP \tag{3}$$

Ⓑ. **Number Field Sieves:** The exTNFS is a variant of the Special Number Field Sieve (SNFS) and the more general General Number Field Sieve (GNFS). Thus, we must briefly introduce them, for more information and an in-depth approach refer to [13].

NFS is an efficient way to calculate D.Logs for $\mathbb{F}_p$ and is generally done as follows:

1) Select two irreducible polynomials that define an algebraic number field that have a common root modulo p.
2) Search for numbers that factor into small primes (known as sieving).
3) Solve linear equations of each prime pair.
4) Get the D.Log by expressing them using previously computed logarithms.

6.1.2 Elliptic Curves, Pairings, and Zero-Knowledge Proofs

The issue's core is the usage of elliptic curve pairings that rely on the discrete logarithm problem for security. The security of pairing-based cryptosystems depends on the hardness of:

- The elliptic curve discrete logarithm problem
- The discrete logarithm problem over finite fields

Essentially, it is a mapping of two elements from two groups into a third group, expressed as:

$$e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T.$$

An elliptic curve pairing can be described as:

$$E_1/\mathbb{F}_p[r] \times E_2/\mathbb{F}_p[r] \to \mu_r \subset (\mathbb{F}_{p^n}).$$

Most SNARK cryptosystems rely on pairings to prove constraints. A detailed explanation of how this occurs is beyond the scope of this discussion, but we refer the reader to [2, 5].

6.2 exTNFS and the Insecurity of BN254

Zero-knowledge proofs work based on elliptic curve pairings, which are reduced to solving the Elliptic Curve Discrete Logarithm problem. However, the **exTNFS** enables discrete logs for prime fields (such as **BN254**) to be found much easier than previously thought, thus reducing their security.

Simple Explanation of exTNFS: A new variant of the Number Field Sieve (NFS) called the extended Tower Number Field Sieve (exTNFS) was introduced in [16], which significantly decreased the complexity of solving the Discrete Logarithm Problem (DLP) in finite fields. By selecting better polynomials in the polynomial selection step outlined in the previous section, solving for discrete logarithms in $\mathbb{F}_{p^k}$ becomes easier. Essentially, when the embedding degree k of an elliptic curve is of the form

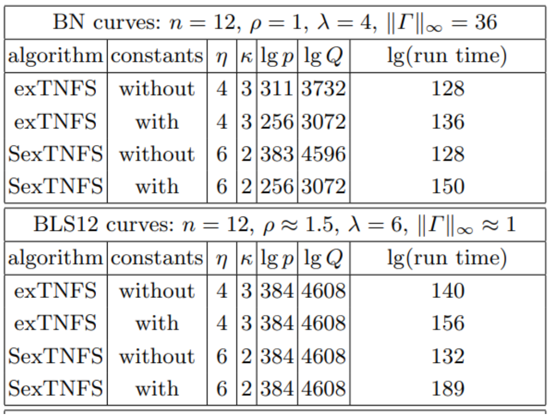
$$k = i \cdot j$$

where $\gcd(i, j) = 1$, the complexity of NFS is greatly reduced. We refer the reader to the original paper [16] for more details.

Implications on BN254: Since k often takes the form $2^n \cdot 3^m$ where $n, m > 1$, and since the embedding degree of a BN curve is always 12, i.e., divisible by 6, it is affected by the extended Tower Number Field Sieve (**exTNFS**).

The security level of BN254 has been debated, but general findings indicate that it has been reduced to approximately 96-110 bits. Specifically, [14] conservatively estimates the security of BN curves with a 256-bit prime to be 110 bits. Table 5 from the paper states that to achieve the desired 128-bit security level, the bit length should be 383 bits. In contrast, the bit length of the BLS12_381 curve would need to be 384 bits, which aligns with previous bit length recommendations.

A more recent study [4] states that the overall security level of BN curves has been reduced to 100 bits, considering optimizations that attackers can exploit in each stage of the NFS algorithm.



BN curves: $n = 12,\ \rho = 1,\ \lambda = 4,\ \|\Gamma\|_\infty = 36$						
algorithm	constants	η	κ	$\lg p$	$\lg Q$	lg(run time)
exTNFS	without	4	3	311	3732	128
exTNFS	with	4	3	256	3072	136
SexTNFS	without	6	2	383	4596	128
SexTNFS	with	6	2	256	3072	150
BLS12 curves: $n = 12,\ \rho \approx 1.5,\ \lambda = 6,\ \|\Gamma\|_\infty \approx 1$						
algorithm	constants	η	κ	$\lg p$	$\lg Q$	lg(run time)
exTNFS	without	4	3	384	4608	140
exTNFS	with	4	3	384	4608	156
SexTNFS	without	6	2	384	4608	132
SexTNFS	with	6	2	384	4608	189

Fig. 4. Table taken from [14]: Table 4 extract showing the prime field and embedded degree to achieve a 128-bit security level.

6.3 Privado.iD Context

Having understood that using the **BN254** curve as the elliptic curve of choice for the ZKP scheme has lower security guarantees than expected, we must discuss its effect on Privado and present our solution.

6.3.1 The Problem

To authenticate the user using `AuthV2Validator.sol` as well as in `CredAtomicV2/3Query.sol`, zk-SNARKs are used to verify credentials and responses. Thus, a significant portion of the overall security relies on these proofs. Specifically, the protocol depends on the user generating a SNARK proof using `Circom` and submitting it to the `Groth16` verifier. As explained above, the Groth16 proof scheme relies on elliptic curve pairings to verify the proof. Therefore, using the **BN254** curve (which Circom employs by default) to generate and verify proofs with `Groth16` reduces the overall security of the protocol to the security level of **BN254**.

6.3.2 The Solution

Usage of elliptic curves with security lower than 112 bits is not allowed for NIST compliance [15], and therefore, a more secure curve should be used. The most suitable candidate for this is `BLS12_381` [1], which is natively supported by Circom/SnarkJS [7].

Ⓐ. **BN254 Curve Advantages**

a. *Simplicity:* BN254 usage is straightforward with Circom due to its default support and extensive adoption, including an existing trusted setup ceremony [11].

b. *Gas efficiency:* Due to Ethereum's precompiles [8, 9], the usage of BN254 is significantly cheaper than `BLS12_381`. The gas cost of a BN254 elliptic curve pairing is given by:

$$80,000 \times k + 100,000$$

where k is the number of elliptic curve pairings to compute. It is worth noting that there is an Ethereum Improvement Proposal (EIP) to implement `BLS12_381` precompiles [10], which would improve gas efficiency.

Ⓑ. **BLS12_381 Advantages**

a. *NIST-Approved Security:* BLS curves are still considered secure under the safe-curves criteria and meet NIST security requirements. Thus, their adoption would ensure compliance with these standards.

b. *Native Circom Support:* Although not as straightforward to use as BN254, Circom does allow for the `BLS12_381` prime field [7].

We summarize the advantages and disadvantages of each curve in the table below.

Curve	Advantages
BN254	1. Gas efficiency 2. Simplicity
BLS12_381	1. NIST-approved security 2. Native Circom support

6.4 Conclusion

BLS12_381 provides stronger security than **BN254**. However, given Privado.iD's web3 context, gas efficiency is a key concern. While there are no immediate practical attacks, we recommend transitioning to **BLS12_381** before NIST's compliance deadline in 2030.

6.5 Client's response

The switch to **BLS12381** curve requires full support from proving and verifier generation libraries, which is not there yet. We added small changes to interfaces to simplify the onboarding of different proving systems and curves in the future.

7 Risk Rating Methodology

The risk rating methodology used by Nethermind Security 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 the finding is to be uncovered and exploited by an attacker. 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 an issue's likelihood and impact, 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 Security 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 pass to the client formally;

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 [High] Issuer may burn another issuer's funds

File(s): VCPayemnt.sol

Description: The issuerAddressBalance mapping registers the amount of ETH paid to the contract, which can later be withdrawn by the issuer. Each issuer can call the updateWithdrawAddress(...) function, which allows for changing the issuer's withdrawal address along with the balance registered in the issuerAddressBalance. This function, however, allows any issuer to burn another issuer's funds by providing the withdrawAddress as the victim address, which would override the previous balance:

```
function updateWithdrawAddress(
    uint256 issuerId,
    uint256 schemaHash,
    address withdrawAddress
) external ownerOrIssuer(issuerId, schemaHash) validAddress(withdrawAddress) {
    VCPaymentStorage storage $ = _getVCPaymentStorage();
    PaymentData storage payData = $.paymentData[keccak256(abi.encode(issuerId, schemaHash))];
    uint256 issuerBalance = $.issuerAddressBalance[payData.withdrawAddress];
 $.issuerAddressBalance[payData.withdrawAddress] = 0;
    // @audit: the balance is overriden instead of added
 $.issuerAddressBalance[withdrawAddress] = issuerBalance;

 payData.withdrawAddress = withdrawAddress;
    _setPaymentData(issuerId, schemaHash, payData);
}
```

Consider the following scenario:

- issuer A has registered 100 ETH in the issuerAddressBalance mapping;
- issuer B has registered 0 ETH in the issuerAddressBalance mapping;
- issuer B calls updateWithdrawAddress(...) with issuers A withdrawalAddress;
- issuer's A balance in issuerAddressBalance mapping is copied from issuer B balance;
- issuer A has a balance of 0 ETH, and the ETH is not recoverable;

Recommendation(s): Consider adding the balance to the new address instead of copying.

Status: Fixed

Update from the client: Fixed by adding the balance and restricting the method to onlyOwner

Update from Nethermind Security: Fixed at commit: 42afd61c.

8.2 [Medium] Request creation may be front-run

File(s): Verifier.sol

Description: The requests are created with `setRequests(...)` function. The request ID is created from the hash of request parameters. This allows the malicious actor to front-run such transactions with requests containing the same parameters. This may lead to the following scenarios:

- scenario 1: front-running batch of requests with a single request creation to fail the batch call:

- the request creator calls `setRequests(...)` with requests [`request_1`, `request_2`] - the malicious actor front-runs the transaction with a call to `setRequests(...)` containing a single request with the same parameters leading to the creation of `request_1` - the call of the request creator with a batch of requests would fail since the `request_1` is already created

- scenario 2: front-running the V2 request creation to provide incorrect data:;

- the request creator calls `setRequests(...)` with `request_1` and parameters for V2 validator - malicious actor front-runs with modified `request_1` with the same parameter but with different metadata, validator (V3 or LMQ), and creator - this leads to an invalid request since the validator does not match the parameters and can't be proven. Additionally, the request with the same parameters can't be created again since V2 doesn't have any variable parameters (in contrast to V3, for which `nullifierSessionID` can be used to create a new request ID with the same parameters)

Scenario 1 temporary DOS, since the request creator can just create the rest of the requests from a failed call. Scenario 2 is particularly dangerous since the request has V2 parameters, but an incorrect validator blocks the set of parameters from being used again, which prevents the request creator from recreating the request. The likelihood of this attack depends on how often creators would choose V2 instead of V3. Additionally, the owner of the `UniversalVerifier` can update the request to the correct validator with the `updateRequest(...)`, therefore the request would not be blocked permanently.

Recommendation(s): Consider adding the `msg.sender` to the creation of request ID, so it's based on request parameters and the sender address. This approach would prevent possible front-running attacks from malicious actors.

Status: Fixed

Update from the client: Now requestId is expected to be calculated as a hash of request params and msg.sender. Contract owner can provide any address instead of msg.sender.

8.3 [Medium] The withdrawToAllIssuers(...) may be blocked

File(s): VCPayment.sol

Description: The function `withdrawToAllIssuers(...)` allows the owner to distribute the payment to all the issuers. The function iterates over every payment data, and if any funds are available, it tries to send Ether to the provided withdrawal address. The funds are transferred with the `call(...)` function, and the returned value is checked:

```
function _withdraw(uint256 amount, address to) internal {
 ...
 (bool sent, ) = to.call{value: amount}("");
    if (!sent) {
        revert WithdrawError("Failed to withdraw");
 }
}
```

This approach is not robust since if the recipient doesn't accept the ETH transfer, the function will revert. This would block the usage of `withdrawToAllIssuers(...)`. Not receiving an ETH transfer may happen if the receiver is the contract that does not implement the `receive(...)` function accidentally or on purpose.

Recommendation(s): Consider handling the `sent` variable in a way that would not block the whole function in a case where one address does not accept ETH.

Status: Fixed

Update from the client: Fixed by removing the withdrawToAllIssuers as not mandatory and potentially dangerous functionality

Update from Nethermind Security: Fixed at commit: 42afd61c.

8.4 [Medium] Verifier can call nonactive AuthValidators

File(s): Verifier.sol

Description: The UniversalVerifier and EmbeddedVerifier contracts extend the Verifier contract and override the submitResponse function. These contracts then invoke super.submitResponse to submit an array of responses and update the proof status.

The Verifier.submitResponse receives the authMethod that is used to authenticate the user and get userID. The AuthMethodData is loaded from storage for a given authMethod.

```
function submitResponse(
    AuthResponse memory authResponse,
    Response[] memory responses,
    bytes memory crossChainProofs
) public virtual {
    // ...
    AuthMethodData storage authMethodData = $._authMethods[authResponse.authMethod];
   // ...
```

As shown below, the struct includes the validator, params, and a boolean flag indicating whether the authentication method is active.

```
struct AuthMethodData {
    IAuthValidator validator;
    bytes params;
    bool isActive;
}
```

An authentication method can be enabled or disabled. However, as seen in the code snippet below of the submitResponse function, once authMethodData is retrieved, the authMethodData.validator.verify function is called without verifying whether the authentication method is still active.

```
    // @audit authMethodData.isActive is never checked
    IAuthValidator.AuthResponseField[] memory authResponseFields;
 (userIDFromAuthResponse, authResponseFields) = authMethodData.validator.verify(
 sender,
 authResponse.proof,
 authMethodData.params
 );
   //...
```

Recommendation(s): Ensure only enabled authentication methods are called.

Status: Fixed

Update from the client: Now the contract reverts if the auth method is disabled

8.5 [Low] Everyone can disable and enable an auth type in UniversalVerifier

File(s): `UniversalVerifier.sol`

Description: The `Verifier` abstract contract provides public functions to set, disable, and enable an auth method. The `UniversalVerifier` extends `Verifier` and correctly, overrides `setAuthMethod` where only the owner can add an auth type. However, the functions `disableAuthMethod` and `enableAuthMethod` are not `virtual` in `Verifier` contract, consequently, they are not overridden in `UniversalVerifier` restricting the access to the owner.

```
function disableAuthMethod(
    string calldata authMethod
) public checkAuthMethodExistence(authMethod, true) {
    VerifierStorage storage s = _getVerifierStorage();
 s._authMethods[authMethod].isActive = false;
}

function enableAuthMethod(
    string calldata authMethod
) public checkAuthMethodExistence(authMethod, true) {
    VerifierStorage storage s = _getVerifierStorage();
 s._authMethods[authMethod].isActive = true;
}
```

This allows the attacker to potentially disable the valid authentication method. Using those authentication methods would still be possible by calling `enableAuthMethod(...)` while submitting the response in the same transaction. Note that the described issue is currently not a problem since the protocol does not use the `isActive` variable, which is described in "[Medium] Verifier can call non-active AuthValidators".

Recommendation(s): Ensure only the owner can disable and enable authentication types.

Status: Fixed

Update from the client: Now, only the contract owner can enable and disable

8.6 [Low] Lack of check for V3 query in request group

File(s): Verifier.sol

Description: The request group may be consisted of CredentialAtomicQueryV3 and LinkedMultiQuery requests. The validity of the group is ensured by the CredentialAtomicQueryV3 response. However, it is possible to create a group request without CredentialAtomicQueryV3. Additionally, responses for such groups may be provided since the `submitResponse(...)` would not validate `userID` if it is not in the response fields:

```
function _checkUserIDMatch(
    uint256 userIDFromAuthResponse,
    IRequestValidator.ResponseField[] memory signals
) internal pure {
    for (uint256 j = 0; j < signals.length; j++) {
        if (
            keccak256(abi.encodePacked(signals[j].name)) ==
            keccak256(abi.encodePacked("userID"))
 ) {
            if (userIDFromAuthResponse != signals[j].value) {
                revert UserIDMismatch(userIDFromAuthResponse, signals[j].value);
 }
 }
 }
}
```

In effect, a validated group that consists only of LinkedMultiQuery requests can exist. Creation of such a group is not likely to happen since the request validator is responsible for providing correct requests, but due to a mistake, such a group could potentially be created and later validated.

Recommendation(s): Consider ensuring that each group contains at least one CredentialAtomicQueryV3 request during group creation.

Status: Fixed

Update from the client: Fixed in a way that at least one request in a group should have `userID` in its public inputs. The same rule applies to each standalone request. The check is done via the internal function:

```
function _isUserIDInputInRequest(
IVerifier.Request memory request
) internal view returns (bool) {
bool userIDInRequests = false;

try request.validator.inputIndexOf(USERID_KEY) {
userIDInRequests = true;
// solhint-disable-next-line no-empty-blocks
} catch {}

return userIDInRequests;
}
```

8.7 [Low] Responses are valid for disabled requests

File(s): *

Description: The existing requests may be disabled by calling `disableRequest(...)` or by removing the whitelisted validator from the `_validatorWhitelist` mapping. Disabled requests are no longer active and can't be fulfilled with proof. However, the previously provided proofs for disabled requests are still considered valid. This may cause issues since the user may provide proof that it is a response to a currently inactive request, but the receiver of that proof may not know about the fact that the request is disabled or the validator for that request is removed.

Recommendation(s): Consider accounting for the request state and its validator when checking proof validity in functions `areMultiRequestProofsVerified(...)` and `isRequestProofVerified(...)`.

Status: Acknowledged

Update from the client: Although it may be confusing that disabled requests still return their verified status, it was a deliberate decision. The disabling should influence only new responses submitted.

8.8 [Low] The `groupID` is not bound to the group contents

File(s): `Verifier.sol`

Description: The `groupID` is currently a random number assigned to requests that are part of the group. However, since the `groupID` is not bound to the requests in the group, there are possible scenarios that the attacker may perform:

- the malicious actor can front-run the call to `setRequests(...)` made by request creator and use their `groupID` with a different set of requests, which would cause the request creator's transaction to fail. This attack is temporal since the request creator can generate a new `groupID`, but the attacker could (potentially) continue the DOS attack. Note, however, that there is no direct incentive for attackers apart from blocking targeted address from creating a group;
- the malicious actor may front-run the call to `setRequests(...)` made by the request creator with the same requests but create a group with only a subgroup or extend a group with additional requests. This could potentially lead to misleading the request creators and users. However, since the creator's transaction would fail, and the requests in the group may be checked, it is unlikely that such a group would be treated as valid by the creator;

Recommendation(s): Consider binding the `groupID` to the requests in the group. This may be achieved by creating `groupID` as a hash of requests' IDs. During group creation, the `groupID` could be compared against the hash of requests' IDs.

Status: Fixed

Update from the client: Fixed as suggested. GroupId should coincide with the hash of concatenated request IDs.

8.9 [Low] The creation of MultiRequest may be front-run

File(s): `Verifier.sol`

Description: The MultiRequest contains an arbitrary ID. However, the creation of a MultiRequest may be front-run by the malicious actor who uses the same exact ID, blocking the original creator from creating the MultiRequest. Note that the attacker has no incentive other than just to block the targeted address from creating a MultiRequest.

Recommendation(s): Consider restricting the MultiRequest ID to a hash of the sender address and additional data.

Status: Fixed

Update from the client: The multiRequestId is now a hase of its requestIds, groupIds, and the sender.

8.10 [Low] `setPaymentValue(...)` can reset totalValue

File(s): `VCPayment.sol`

Description: The function `setPaymentValue(...)` does not check if the payment data was already assigned in the `paymentData` mapping under the pair (issuerId, schemaHash):

```
function setPaymentValue(
    uint256 issuerId,
    uint256 schemaHash,
    uint256 value,
    uint256 ownerPercentage,
    address withdrawAddress
) public onlyOwner validPercentValue(ownerPercentage) validAddress(withdrawAddress) {
    VCPaymentStorage storage $ = _getVCPaymentStorage();
    // @audit paymentData[keccak256(abi.encode(issuerId, schemaHash))] can be replaced
    //        and reset totalValue
    PaymentData memory newPaymentData = PaymentData(
 issuerId,
 schemaHash,
 value,
 ownerPercentage,
 withdrawAddress,
        0
 );
    // @audit paymentDataId can be duplicated in paymentDataIds
 $.paymentDataIds.push(keccak256(abi.encode(issuerId, schemaHash)));
    _setPaymentData(issuerId, schemaHash, newPaymentData);
}
```

Setting the payment data again to the same pair of (issuerId, schemaHash) would result in resetting the `totalValue` and adding unnecessary data in the `paymentDataIds` array.

Recommendation(s): Consider checking if the payment data was already added in the `setPaymentValue(...)`.

Status: Fixed

Update from the client: Fixed via checking that withdrawAddress is not zero, which is never the case as it's impossible to create PaymentData with zero address.

Update from Nethermind Security: Fixed at commit: `42afd61c`.

8.11 [Low] payERC20Permit is vulnerable to DOS via front-running

File(s): `MCPayment.sol`

Description: The `payERC20Permit` function utilizes `permitSignature` to facilitate ERC20 token payments using EIP-2612 permits, reducing gas costs. As shown below, the function calls `token.permit`, allowing the caller to grant `MCPayment` permission to transfer tokens on their behalf.

```
function payERC20Permit(
 bytes memory permitSignature,
 Iden3PaymentRailsERC20RequestV1 memory paymentData,
 bytes memory signature
) external {
 address signer = _recoverERC20PaymentSignature(paymentData, signature);
 ERC20Permit token = ERC20Permit(paymentData.tokenAddress);
 if (permitSignature.length != 65) {
 revert ECDSAInvalidSignatureLength();
 }
 // ...

 assembly {
 r := mload(add(permitSignature, 0x20))
 s := mload(add(permitSignature, 0x40))
 v := byte(0, mload(add(permitSignature, 0x60)))
 }
 //@audit If permitSignature is front-run, then this function reverts
 //       once the nonce in it was already consumed.
 token.permit(
 msg.sender,
 address(this),
 paymentData.amount,
 paymentData.expirationDate,
 v,
 r,
 s
 );
 _transferERC20(paymentData, signer);
}
```

However, a malicious actor can execute a `permit` before the original transaction is processed. This front-running attack would consume the nonce in `permitSignature`, causing the original transaction to fail due to an invalid nonce.

In this case, the allowance is granted, but the function `payERC20Permit` still reverts. As a result, the signer must generate a new signature for the caller to successfully invoke `payERC20Permit` again.

Recommendation(s): Ensure that an allowance for the `paymentData.amount` exists if the call to `permit` fails, for example:

```
try token.permit(
 msg.sender,
 address(this),
 paymentData.amount,
 paymentData.expirationDate,
 v,
 r,
 s
 ) {
 // the permit function was executed successfully, continue
} catch {
 // Check the allowance to confirm if the permit was already executed
 uint256 currentAllowance = token.allowance(msg.sender, address(this));
 if(currentAllowance < paymentData.amount) {
 revert("Permit failed and the allowance is not sufficient");
 }
 // Otherwise, proceed as if permit was successful
}
```

Status: Fixed

Update from the client:

Update from Nethermind Security: Fixed at commit bc56dc3b.

8.12 [Info] Checking the validity of the response group

File(s): `Verifier.sol`

Description: Currently, there is no functionality to explicitly check the validity of responses for the chosen group. To check if the responses to a group of requests are valid there are two ways:

- getting all the requests from a group with `getGroupedRequests(...)`, then checking for each response the status with `getRequestProofStatus(...)` and fetching `linkID` from response fields with `getResponseFields(...)` and making sure all the `linkID` in the group are equal;
- calling `getMultiRequestProofsStatus(...)` or `areMultiRequestProofsVerified(...)` for the MultiRequest that contains a desired group. Those functions use `_checkLinkedResponseFields(...)`, which ensures that the responses in a group have the same correct `linkID`;

Recommendation(s): Consider implementing a function that allows checking the validity of responses in a group that would check if the elements of a group have a common `link`. Alternatively, consider checking if the response's `linkID` matches `linkIDs` of other responses in a group during the `submitResponse(...)`. This would ensure that each response is correctly added to the group during the submission time.

Status: Acknowledged

Update from the client: At the moment there is no business need to check a group status in a separate function. We can skip it for now because it can be added later. An alternative way of checking for equality of the linkIDs at submit response time was discussed and rejected as it would mean full group submission in a single tx, which may increase risks of hitting the block gas limit of 30M gas for some groups.

8.13 [Info] LinkID may be shared between groups

File(s): `*`

Description: The `linkID` bounds responses within a group. It is created from `hash(issuerClaimHash, linkNonce)`. However, the user may generate the same `linkID` for many groups since `linkNonce` may be provided the same for many responses.

Recommendation(s): Consider ensuring that the `linkID` is exclusive to one group only.

Status: Acknowledged

Update from the client: This does not introduce a vulnerability since the proof with linkID can't be reused for another credential.

8.14 [Info] LinkedMultiQuery proof may have `linkID=0`

File(s): `LinkedMultiQueryValidator.sol`

Description: The proof for LinkedMultiQuery may contain `linkID=0`, a value reserved only for poofs outside the group.

Recommendation(s): Consider checking if the `linkID` is not zero in the `LinkedMultiQueryValidator`.

Status: Fixed

Update from the client: The vulnerability is not even an info level but a more serious one. If an attacker deliberately use linkNonce == 0 then linkID will be equal to constant value 0 rather than hash-based. So this is the way to use any (!) credential for LMK in the proof for grouped requests but not the one that corresponds to the V3 request.

8.15 [Info] Low Security from BN254 Curve

File(s): `/groth16-verifiers`

Description: The BN254 curve provides only around a 100-bit level of security due to the number of theoretic attacks possible. Specifically, the Tower Number Field Sieve (TNFS) [BGK15] and its derivatives, which is the most efficient way to solve Discrete Logarithms. This reduces the intended security of BN254 of 128-bit security level to between 96-bit and 110-bit security [KB15][PSS]. Additionally, NIST requires a bit security level of 112-bits [Barker20].

Recommendation(s): We recommend switching to a more secure elliptic curve, such as BLS12-381, which Circom/SnarkJS provides natively.

Status: Acknowledged

Update from the client: The switch to BLS12-381 curve requires full support from prooving and verifier generation libraries which is not there yet. We added small changes to interfaces to simplify onboarding of different prooving systems and curves in the future.

8.16 [Info] Redundant storage write

File(s): VCPayment.sol

Description: The functions updateOwnerPercentage(...), updateWithdrawAddress(...), updateValueToPay(...) and pay(...) modify the payData in the mapping paymentData, but later call _setPaymentData(...) which additionally writes the payData to the paymentData mapping.

```
    function updateOwnerPercentage(
        uint256 issuerId,
        uint256 schemaHash,
        uint256 ownerPercentage
) public onlyOwner validPercentValue(ownerPercentage) {
        VCPaymentStorage storage $ = _getVCPaymentStorage();
        PaymentData storage payData = $.paymentData[keccak256(abi.encode(issuerId, schemaHash))];
payData.ownerPercentage = ownerPercentage;
        // @audit redundant update for withdrawAddress
        _setPaymentData(issuerId, schemaHash, payData);
}
```

This increases the cost of writing data for storage.

Recommendation(s): To save gas costs, consider avoiding unnecessary writes to storage.

Status: Fixed

Update from the client: Fixed by removing _setPaymentData internal function and writing only necessary fields in all the methods.

Update from Nethermind Security: Fixed at commit: 42afd61c.

8.17 [Info] The getRequestProofStatus(...) does not account for group validity

File(s): Verifier.sol

Description: The getRequestProofStatus(...) returns the status of a proof. However, the data returned by this function is insufficient to ensure validity in the case of LMQ. For the LMQ, the linkID should be checked against the group to ensure that a V3 proof is also verified. The caller of the function getRequestProofStatus(...) may consider proof LMQ valid even if the V3 proof in a group has not been verified yet.

Recommendation(s): Consider handling LMQ casein getRequestProofStatus(...) to correctly return the proof status.

Status: Acknowledged

Update from the client: Although the comment is correct, this is a known issue, and we don't see it as a problem as it is the responsibility of a proof status consumer to point to correct requests and groups. At the same, as far as partial group submission is concerned, this function may be a good API to track data of such submissions in a request-isolated manner.

8.18 [Info] The withdrawToAllIssuers(...) is not optimal

File(s): VCPayment.sol

Description: The function withdrawToAllIssuers(...) always iterates over the array containing all the created payment IDs. This is not optimal since it requires always iterating over all payment data, even if the issuer is inactive. Such an approach makes executing the function more costly with an increased number of issuers.

Recommendation(s): Consider allowing the removal of inactive issuers.

Status: Fixed

Update from the client: Fixed by removing the withdrawToAll method as not mandatory functionality

Update from Nethermind Security: Fixed at commit: 42afd61c.

8.19 [Info] The cross-chain proofs may be provided to the same chain

File(s): State.sol

Description: The cross-chain proof may be submitted in the submitResponse(...). It is handled by State.processCrossChainProofs(...) and stored in either _rootToGistRootReplacedAt or _idToStateReplacedAt mapping. However, the provided cross-chain proof may be obtained from the same chain on which the State contract exists. This would result in storing it in _rootToGistRootReplacedAt or _idToStateReplacedAt mapping, but could not be accessed by getGistRootReplacedAt(...) or getStateReplacedAt(...) since the provided id is supported on such chain.

Recommendation(s): Consider not storing the cross-chain proof in _rootToGistRootReplacedAt and _idToStateReplacedAt if the id is supported.

Status: Acknowledged

Update from the client: Although the cross-chain from the same chain can be stored in the `_rootToGistRootReplacedAt` and `_idToStateReplacedAt`, it will not take effect on the verification as `getStateReplacedAt` and `getGistRootReplacedAt` methods of state contract will not refer to that mapping if identity idType is supported, which effectively mean that an identity should "natively" reside in the State contract.

8.20 [Info] Unused skipClaimRevocationCheck

File(s): `CredentialAtomicQueryV3Validator.sol`

Description: The `skipClaimRevocationCheck` is a parameter of the `CredentialAtomicQueryV3` request. However, the CredentialAtomicQueryV3Validator does not use that.

Recommendation(s): Consider removing unused parameter from the `CredentialAtomicQueryV3` request.

Status: Acknowledged

Update from the client: Although that is true that the parameter is not used in the validator, it serves for informational purposes.

8.21 [Info] Using CREATE2 on ZKsync Era

File(s): `*`

Description: The Privado ID aims to keep uniformity of contract addresses between many EVM-compatible networks. To achieve this, the `CREATE2` opcode is used to control the parameters from which the address is created. This is possible since many EVM chains use the same formula to derive the address. However, the ZKsync Era, an EVM-compatible chain, uses a distinct address derivation method compared to Ethereum. This means the same bytecode deployed on Ethereum and ZKsync will have different addresses. More information can be found under this link: ZKsync Era Address Derivation. If the Privado ID system plans to deploy on ZKsync Era, the address uniformity may not be achievable on this network and could cause downstream issues.

Recommendation(s): Consider creating a strategy for handling the ZKsync Era addresses differences. It may involve documenting this special case to users and developers.

Status: Acknowledged

Update from the client: If we deploy to the ZKSync era, the relevant documentation will be prepared

8.22 [Info] `MCPayment` can't be used with USDT

File(s): `MCPayment.sol`

Description: The `_transferERC20(...)` allows to transfer the tokens only if the `token.transferFrom(...)` return `true`. However, some tokens don't follow exactly `ERC20` standard and don't return `true` on transfer. In particular, the `USDT` is a widely used token that doesn't return `true`. In the current implementation, the `USDT` could not be used as payment.

Recommendation(s): Consider using `SafeERC20.trySafeTransfer(...)` library to safely handle transfers that both return and don't return boolean. The `trySafeTransfer(...)` would return `true` on successful calls in both cases, which can be easily integrated with the current version of `MCPayemnt`.

Status: Fixed

Update from the client: `SafeERC20.trySafeTransfer(...)` is not yet included in the latest OpenZeppelin version, so `SafeERC20.safeTransferFrom(...)` was used instead for the time being until `trySafeTransfer(...)` is available.

8.23 [Best Practice] Apply reentrancy protection

File(s): `VCPayment.sol`

Description: The functions that can be accessed by the issuers, i.e.: `updateWithdrawAddress(...)` and `updateValueToPay(...)` can be reentered when the issuer address receives ETH during withdrawal. While it does not pose direct security risk, it is best practice to protect from the reentrancy.

Recommendation(s): Consider restricting those functions against reentrancy.

Status: Fixed

Update from the client: nonReentrant OpenZeppelin modifier applied to updateWithdrawAddress, updateValueTopay, and pay methods.

Update from Nethermind Security: Fixed at commit: `42afd61c`.

8.24 [Best Practice] Disable the initializer of the implementation

File(s): `VCPayment.sol`

Description: The initializer is not disabled during the construction time of the implementation contract. This may result in any address gaining the owner role of the implementation contract. While this does not create a direct danger to the proxy, the malicious address may use an implementation contract to interact in unwanted ways, like calling blacklisted addresses which may cause reputation or legal problems.

Recommendation(s): Consider disabling the constructor's initializer. This may be done by using `_disableInitializers(...)` in the constructor of the implementation contract.

Status: Fixed

Update from the client: Fixed

Update from Nethermind Security: Fixed at commit: `42afd61c`.

8.25 [Best Practice] Lack of `_disableInitializers(...)`

File(s): `*`

Description: Current impelemntation contracts that use `_disableInitializers(...)` are State and VCPayemnt. However, all the listed contracts do not use `_disableInitializers(...)`:

- MCPayment.sol;
- Auth2Validator.sol;
- EthIdentityValidator.sol;
- CredentialAtomicQueryMTPV2Validator.sol;
- CredentialAtomicQuerySigV2Validator.sol;
- CredentialAtomicQueryV3Validator.sol;
- LinkedMultiQueryValidator.sol;
- UniversalVerifier.sol;

Recommendation(s): Consider implementing the `_disableInitializers(...)` in the

Status: Fixed

Update from the client: `_disableInitializers(...)` are added to all upgradeable contracts

9 Documentation Evaluation

Software documentation refers to the written or visual information that describes the functionality, architecture, design, and implementation of software. 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 to that the smart contract is secure and free from vulnerabilities.

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

Remarks about the Privado.iD documentation

The **Privado.iD** protocol documentation was made available through docs.privado.id with a high-level overview along with examples. In addition, the team provided audit-specific documentation, detailing the core contracts and outlining the verification flow step by step. Furthermore, the **Privado.iD** team thoroughly addressed all questions and concerns raised by the **Nethermind Security** team, providing valuable insights and a comprehensive understanding of the project's technical aspects.

10 Test Suite Evaluation

10.1 Compilation Output

```
% npx hardhat compile
Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing
↪  "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for
↪  non-open-source code. Please see https://spdx.org for more information.
--> contracts/test-helpers/ERC20PermitToken.sol
Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing
↪  "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for
↪  non-open-source code. Please see https://spdx.org for more information.
--> contracts/test-helpers/ERC20Token.sol

Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
   --> contracts/identitytreestore/IdentityTreeStore.sol:137:9:
    |
137 |         uint256 id,
    |         ^^^^^^^^^^

Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
   --> contracts/validators/auth/AuthV2Validator.sol:104:9:
    |
104 |         address sender,
    |         ^^^^^^^^^^^^^^


Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
   --> contracts/validators/auth/AuthV2Validator.sol:107:9:
    |
107 |         bytes calldata params
    |         ^^^^^^^^^^^^^^^^^^^^^

Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
  --> contracts/validators/auth/EthIdentityValidator.sol:77:9:
   |
77 |         bytes calldata params
   |         ^^^^^^^^^^^^^^^^^^^^^

Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
   --> contracts/validators/request/LinkedMultiQueryValidator.sol:102:9:
    |
102 |         address sender,
    |         ^^^^^^^^^^^^^^

Warning: Contract code size is 30612 bytes and exceeds 24576 bytes (a limit introduced in Spurious Dragon). This
↪  contract may not be deployable on Mainnet. Consider enabling the optimizer (with a low "runs" value!), turning off
↪  revert strings, or using libraries.
 --> contracts/verifiers/RequestDisableable.sol:9:1:
  |
9 | contract RequestDisableable is Verifier {
  | ^ (Relevant source part starts here and spans across multiple lines).

Warning: Contract code size is 30549 bytes and exceeds 24576 bytes (a limit introduced in Spurious Dragon). This
↪  contract may not be deployable on Mainnet. Consider enabling the optimizer (with a low "runs" value!), turning off
↪  revert strings, or using libraries.
  --> contracts/verifiers/ValidatorWhitelist.sol:12:1:
   |
12 | contract ValidatorWhitelist is Verifier {
   | ^ (Relevant source part starts here and spans across multiple lines).

Generating typings for: 119 artifacts in dir: typechain-types for target: ethers-v6
Successfully generated 292 typings!
Compiled 109 Solidity files successfully (evm target: paris).
```

10.2 Tests Output

```
npx hardhat test
Downloading compiler 0.8.27
Downloading compiler 0.8.27
Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing
 ↪  "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for
 ↪  non-open-source code. Please see https://spdx.org for more information.
--> contracts/test-helpers/ERC20PermitToken.sol

Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing
 ↪  "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for
 ↪  non-open-source code. Please see https://spdx.org for more information.
--> contracts/test-helpers/ERC20Token.sol

Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
   --> contracts/identitytreestore/IdentityTreeStore.sol:125:9:
    |
125 |         uint256 id,
    |         ^^^^^^^^^^

Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
  --> contracts/validators/AuthV2Validator.sol:80:9:
   |
80 |         uint256[] memory inputs,
   |         ^^^^^^^^^^^^^^^^^^^^^^^

Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
  --> contracts/validators/AuthV2Validator.sol:82:9:
   |
82 |         uint256[2] memory a,
   |         ^^^^^^^^^^^^^^^^^^^

Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
  --> contracts/validators/AuthV2Validator.sol:84:9:
   |
84 |         uint256[2][2] memory b,
   |         ^^^^^^^^^^^^^^^^^^^^^^

Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
  --> contracts/validators/AuthV2Validator.sol:86:9:
   |
86 |         uint256[2] memory c,
   |         ^^^^^^^^^^^^^^^^^^^

Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
  --> contracts/validators/AuthV2Validator.sol:88:9:
   |
88 |         bytes calldata data,
   |         ^^^^^^^^^^^^^^^^^^^

Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
  --> contracts/validators/AuthV2Validator.sol:90:9:
   |
90 |         address sender
   |         ^^^^^^^^^^^^^^

Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
   --> contracts/validators/AuthV2Validator.sol:106:9:
    |
106 |         bytes calldata data,
    |         ^^^^^^^^^^^^^^^^^^^

Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
   --> contracts/verifiers/UniversalVerifier.sol:185:9:
```

```
185 |         uint256 startIndex,
    |         ^^^^^^^^^^^^^^^^^^


Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
   --> contracts/verifiers/UniversalVerifier.sol:186:9:
    |
186 |         uint256 length
    |         ^^^^^^^^^^^^^^


Warning: Function state mutability can be restricted to pure
  --> contracts/validators/AuthV2Validator.sol:78:5:
   |
78 |     function verify(
   |     ^ (Relevant source part starts here and spans across multiple lines).


Warning: Function state mutability can be restricted to pure
   --> contracts/verifiers/UniversalVerifier.sol:184:5:
    |
184 |     function getZKPRequests(
    |     ^ (Relevant source part starts here and spans across multiple lines).


Compiled 108 Solidity files successfully (evm target: paris).


GenesisUtilsWrapper deployed to: 0x5FbDB2315678afecb367f032d93F642f64180aa3
PrimitiveUtilsWrapper deployed to: 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512
  IdentityTreeStore
 Poseidon1Element deployed to: 0x9fE46736679d2D9a65F0992F2272dE9f3c7fa6e0
 Poseidon2Element deployed to: 0xCf7Ed3AccA5a467e9e704C703E8D87F634fB0Fc9
 Poseidon3Element deployed to: 0xDc64a140Aa3E981100a9becA4E685f962f0cF6C9
 SmtLib deployed to:  0x5FC8d32690cc91D4c39d9d3abcBD16989F875707
 Groth16VerifierStub contract deployed to address 0x0165878A594ca255338adfa4d48449f69242Eb8F from
 ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
 StateLib deployed to:  0xa513E6E4b8f2a923D98304ec87F64353C4D5C853
 StateCrossChainLib deployed to:  0x2279B7A0a67DB372996a5FaB50D91eAA73d2eBe6
 CrossChainProofValidator deployed to: 0x8A791620dd6260079BF849Dc5567aDC3F2FdC318
Warning: Potentially unsafe deployment of contracts/state/State.sol:State

    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.

 State contract deployed to address 0xB7f8BC63BbcaD18155201308C8f3540b07f84F5e from
 ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
 Poseidon2Element deployed to: 0x0DCd1Bf9A1b36cE34237eEaFef220932846BCD82
 Poseidon3Element deployed to: 0x9A676e781A523b5d0C0e43731313A708CB607508
Warning: Potentially unsafe deployment of contracts/identitytreestore/IdentityTreeStore.sol:IdentityTreeStore

    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.


IdentityTreeStore deployed to: 0x959922bE3CAee4b8Cd9a407cc3ac1C251C2007B1
     Should return the revocation status single leaf (84ms)
     Should revert on invalid roots length
    Should return the revocation status many leafs
       left key path
       right key path (40ms)

  State Cross Chain
 CrossChainProofValidator deployed to: 0x3Aa5ebB10DC797CAC828524e59A333d0A371443c
     Should process the messages without replacedAtTimestamp
     Should process the messages with replacedAtTimestamp
     Oracle timestamp should not be in the past
     Oracle replacedAtTimestamp or oracle timestamp cannot be in the future
     Should fail to verify a message which was tampered with
     Should fail to verify a message which signature is invalid

  Disable Proxy Contract test
```

```
      Should disable and enable proxy contract (52ms)

  generate ID from genesis state and idType
      testVector: 0
      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
      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

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

  test calculate id from ETH address
      calcOnchainIdFromAddress

  test calculate id type from id
      Iden3 Polygon Amoy

  Claim builder tests
ClaimBuilderWrapper deployed to: 0x7a2088a1bFc9d81c55368AE168C2C02570cB814F
      validate buildClaim
      validate buildClaim errors (42ms)
25
      validate buildClaim from file

  Next tests reproduce identity life cycle
 Poseidon3Element deployed to: 0x09635F643e140090A9A8Dcd712eD6285858ceBef
 Poseidon4Element deployed to: 0xc5a5C42992dECbae36851359345FE25997F5C42d
 Poseidon1Element deployed to: 0x67d269191c92Caf3cD7723F116c85e6E9bf55933
 Poseidon2Element deployed to: 0xE6E340D132b5f46d1e472DebcD681B2aBc16e57E
 Poseidon3Element deployed to: 0xc3e53F4d16Ae77Db1c982e75a937B9f60FE63690
 SmtLib deployed to:  0x84eA74d481Ee0A5332c457a4d796187F6Ba67fEB
 Groth16VerifierStateTransition contract deployed to address 0x9E545E3C0baAB3E08CdfD552C960A1050f373042 from
  ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
 StateLib deployed to:  0xa82fF9aFd8f496c3d6ac40E2a0F282E47488CFc9
 StateCrossChainLib deployed to:  0x1613beB3B2C4f22Ee086B2b38C1476A3cE7f78E8
 CrossChainProofValidator deployed to: 0x851356ae760d987E095750cCeb3bC6014560891C
Warning: Potentially unsafe deployment of contracts/state/State.sol:State

    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.

 State contract deployed to address 0x95401dc811bb5740090279Ba06cfA8fcF6113778 from
  ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
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.

GenesisUtilsWrapper deployed to: 0x0E801D84Fa97b50751Dbf25036d067dCf18858bF
    create identity
       deploy state and identity
       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
       getClaimProofWithStateInfo should return non-existence proof
       getRevocationProofWithStateInfo should return non-existence proof
       getRootProofWithStateInfo should return non-existence proof
    add claim
       we should not have proof about claim existing but not published
       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
    make transition
       we should have proof about claim existing if published
       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
       claim proof must exist after publishing and StateInfo should be latest
    revoke state
       revoked index should not exists in Revocation tree if not published
       transit of revocation tree shouldn't update root of roots tree
       Root of Roots and Claims Root should be changed
    make transition after revocation
       revoked index should exists in Revocation tree if published
       state should be updated
       revocation proof must exist after publishing and StateInfo should be latest

  Claims tree proofs
 Poseidon3Element deployed to: 0x809d550fca64d94Bd9F66E60752A544199cfAC3D
 Poseidon4Element deployed to: 0x4c5859f0F772848b2D91F1D83E2Fe57935348029
 Poseidon1Element deployed to: 0x1291Be112d480055DaFd8a610b7d1e203891C274
 Poseidon2Element deployed to: 0x5f3f1dBD7B74C6B46e8c44f98792A1dAf8d69154
 Poseidon3Element deployed to: 0xb7278A61aa25c888815aFC32Ad3cC52fF24fE575
 SmtLib deployed to:  0xCD8a1C3ba11CF5ECfa6267617243239504a98d90
 Groth16VerifierStateTransition contract deployed to address 0x82e01223d51Eb87e16A03E24687EDF0F294da6f1 from
 →  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
 StateLib deployed to:  0x2bdCC0de6bE1f7D2ee689a0342D76F52E8EFABa3
 StateCrossChainLib deployed to:  0x7969c5eD335650692Bc04293B07F5BF2e7A673C0
 CrossChainProofValidator deployed to: 0x7bc06c482DEAd17c0e297aFbC32f6e63d3846650
Warning: Potentially unsafe deployment of contracts/state/State.sol:State

    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.

 State contract deployed to address 0xFD471836031dc5108809D173A067e8486B9047A3 from
  →  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
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
     Get proof for claim by root

  Revocation tree proofs
 Poseidon3Element deployed to: 0x1fA02b2d6A771842690194Cf62D91bdd92BfE28d
 Poseidon4Element deployed to: 0xdbC43Ba45381e02825b14322cDdd15eC4B3164E6
 Poseidon1Element deployed to: 0x04C89607413713Ec9775E14b954286519d836FEf
 Poseidon2Element deployed to: 0x4C4a2f8c81640e47606d3fd77B353E87Ba015584
 Poseidon3Element deployed to: 0x21dF544947ba3E8b3c32561399E88B52Dc8b2823
 SmtLib deployed to:  0x2E2Ed0Cfd3AD2f1d34481277b3204d807Ca2F8c2
  Groth16VerifierStateTransition contract deployed to address 0xD8a5a9b31c3C0232E196d518E89Fd8bF83AcAd43 from
  →  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
 StateLib deployed to:  0xDC11f7E700A4c898AE5CAddB1082cFfa76512aDD
 StateCrossChainLib deployed to:  0x51A1ceB83B83F1985a81C295d1fF28Afef186E02
 CrossChainProofValidator deployed to: 0x36b58F5C1969B7b6591D752ea6F5486D069010AB
Warning: Potentially unsafe deployment of contracts/state/State.sol:State

    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.

 State contract deployed to address 0x0355B7B8cb128fA5692729Ab3AAa199C1753f726 from
  ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
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
     Get proof for revocation by root

  Root of roots tree proofs
 Poseidon3Element deployed to: 0x2B0d36FACD61B71CC05ab8F3D2355ec3631C0dd5
 Poseidon4Element deployed to: 0xfbC22278A96299D91d41C453234d97b4F5Eb9B2d
 Poseidon1Element deployed to: 0x46b142DD1E924FAb83eCc3c08e4D46E82f005e0E
 Poseidon2Element deployed to: 0xC9a43158891282A2B1475592D5719c001986Aaec
 Poseidon3Element deployed to: 0x1c85638e118b37167e9298c2268758e058DdfDA0
 SmtLib deployed to:  0x367761085BF3C12e5DA2Df99AC6E1a824612b8fb
 Groth16VerifierStateTransition contract deployed to address 0x4C2F7092C2aE51D986bEFEe378e50BD4dB99C901 from
  ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
 StateLib deployed to:  0x7A9Ec1d04904907De0ED7b6839CcdD59c3716AC9
 StateCrossChainLib deployed to:  0x49fd2BE640DB2910c2fAb69bB8531Ab6E76127ff
 CrossChainProofValidator deployed to: 0x4631BCAbD6dF18D94796344963cB60d44a4136b6
Warning: Potentially unsafe deployment of contracts/state/State.sol:State

    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.

 State contract deployed to address 0xA4899D35897033b927acFCf422bc745916139776 from
  ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
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
    Check historical Claim tree root in claims tree root
       Check that Root of Roots tree not contains latest claims tree root
       Check that current Root of roots contains latest claims tree root

  Compare historical roots with latest roots from tree
 Poseidon3Element deployed to: 0xc0F115A19107322cFBf1cDBC7ea011C19EbDB4F8
 Poseidon4Element deployed to: 0xc96304e3c037f81dA488ed9dEa1D8F2a48278a75
 Poseidon1Element deployed to: 0x34B40BA116d5Dec75548a9e9A8f15411461E8c70
 Poseidon2Element deployed to: 0xD0141E899a65C95a556fE2B27e5982A6DE7fDD7A
 Poseidon3Element deployed to: 0x07882Ae1ecB7429a84f1D53048d35c4bB2056877
 SmtLib deployed to:  0x22753E4264FDDc6181dc7cce468904A80a363E44
 Groth16VerifierStateTransition contract deployed to address 0xA7c59f010700930003b33aB25a7a0679C860f29c from
  ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
 StateLib deployed to:  0xfaAddC93baf78e89DCf37bA67943E1bE8F37Bb8c
 StateCrossChainLib deployed to:  0x276C216D241856199A83bf27b2286659e5b877D3
 CrossChainProofValidator deployed to: 0x3347B4d90ebe72BeFb30444C9966B2B990aE9FcB
Warning: Potentially unsafe deployment of contracts/state/State.sol:State

    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.

 State contract deployed to address 0x5bf5b11053e734690269C6B9D438F8C9d48F528A from
  ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
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
 Poseidon3Element deployed to: 0x5fc748f1FEb28d7b76fa1c6B07D8ba2d5535177c
 Poseidon4Element deployed to: 0xB82008565FdC7e44609fA118A4a681E92581e680
 Poseidon1Element deployed to: 0x2a810409872AfC346F9B5b26571Fd6eC42EA4849
 Poseidon2Element deployed to: 0xb9bEECD1A582768711dE1EE7B0A1d582D9d72a6C
 Poseidon3Element deployed to: 0x8A93d247134d91e0de6f96547cB0204e5BE8e5D8
 SmtLib deployed to:  0x40918Ba7f132E0aCba2CE4de4c4baF9BD2D7D849
 Groth16VerifierStateTransition contract deployed to address 0xF32D39ff9f6Aa7a7A64d7a4F00a54826Ef791a55 from
  ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
 StateLib deployed to:  0xd6e1afe5cA8D00A2EFC01B89997abE2De47fdfAf
 StateCrossChainLib deployed to:  0x99dBE4AEa58E518C50a1c04aE9b48C9F6354612f
 CrossChainProofValidator deployed to: 0x6F6f570F45833E249e27022648a26F4076F48f78
Warning: Potentially unsafe deployment of contracts/state/State.sol:State

    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.

 State contract deployed to address 0xB0f05d25e41FbC2b52013099ED9616f1206Ae21B from
  ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
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
 Poseidon3Element deployed to: 0x6C2d83262fF84cBaDb3e416D527403135D757892
 Poseidon4Element deployed to: 0xFD6F7A6a5c21A3f503EBaE7a473639974379c351
 Poseidon1Element deployed to: 0xa6e99A4ED7498b3cdDCBB61a6A607a4925Faa1B7
 Poseidon2Element deployed to: 0x5302E909d1e93e30F05B5D6Eea766363D14F9892
 Poseidon3Element deployed to: 0x0ed64d01D0B4B655E410EF1441dD677B695639E7
 SmtLib deployed to:  0x4bf010f1b9beDA5450a8dD702ED602A104ff65EE
 Groth16VerifierStateTransition contract deployed to address 0x40a42Baf86Fc821f972Ad2aC878729063CeEF403 from
  ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
 StateLib deployed to:  0x96F3Ce39Ad2BfDCf92C0F6E2C2CAbF83874660Fc
 StateCrossChainLib deployed to:  0x986aaa537b8cc170761FDAC6aC4fc7F9d8a20A8C
 CrossChainProofValidator deployed to: 0xde2Bd2ffEA002b8E84ADeA96e5976aF664115E2c
Warning: Potentially unsafe deployment of contracts/state/State.sol:State

    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.

 State contract deployed to address 0x870526b7973b56163a6997bB7C886F5E4EA53638 from
  ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
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

  MC Payment Contract
     Check signature verification:
     Check payment:
     Update owner percentage:
     Owner withdraw not owner account:
     Invalid signature
     Invalid payment value
     Pay twice for the same nonce
     Expired payment
```

```
ERC-20 Payment Gas: 141188
     ERC-20 payment:
     ERC-20 payment - invalid signature
     ERC-20 payment - paymend already done
     ERC-20 payment - call erc20Payment without approval
EIP-2612 Payment Gas: 194375
     ERC-20 Permit (EIP-2612) payment:
     ERC-20 Permit (EIP-2612) payment - invalid permit signature length:
     ERC-20 Permit (EIP-2612) payment - invalid signature:
     ERC-20 Permit (EIP-2612) payment - payment already done:
     Check payment - different signer and recipient:
     ERC-20 payment - different signer and recipient:
     ERC-20 Permit (EIP-2612) payment - different signer and recipient:

  VC Payment Contract
     Payment and issuer withdraw:
     Withdraw to all issuers and owner:
     Update withdrawAddress
     updateValueToPay
     getPaymentData work only for issuer or owner
     updateOwnerPercentage work only for owner
     updateValueToPay work only for issuer or owner
     updateWithdrawAddress work only for issuer or owner
     test rounded division for percents

  poseidon
 Poseidon1Element deployed to: 0xddE78e6202518FF4936b5302cC2891ec180E8bFf
 Poseidon2Element deployed to: 0xB06c856C8eaBd1d8321b687E188204C1018BC4E5
 Poseidon3Element deployed to: 0xaB7B4c595d3cE8C85e16DA86630f2fc223B05057
 Poseidon4Element deployed to: 0xAD523115cd35a8d4E60B3C0953E0E0ac10418309
 Poseidon5Element deployed to: 0x045857BDEAE7C1c7252d611eB24eB55564198b4C
 Poseidon6Element deployed to: 0x2b5A4e5493d4a54E717057B127cf0C000C876f9B
SpongePoseidon deployed to: 0x413b1AfCa96a3df5A686d8BFBF93d30688a7f7D9
PoseidonFacade deployed to: 0x02df3a3F960393F5B349E40A599FEda91a7cc1A7
     check poseidon hash function with inputs [1, 2]
     check poseidon hash function with inputs [1, 2, 3]
     check sponge poseidon hash function with inputs (1526ms)

  uint conversions
     convert address to uint in little endian and back
     convert address to challenge (uint in little endian)
     convert uintLE to address
     invalid challenge (uint256 LE address) must produce error
     convert address to uint256
     convert uint256 to addr

  ReverseHashWrapper
 Poseidon2Element deployed to: 0x821f3361D454cc98b7555221A06Be563a7E2E0A6
 Poseidon3Element deployed to: 0x1780bCf4103D3F501463AD3414c7f4b654bb7aFd
     Should save and get preimages

  Merkle tree proofs of SMT
    SMT existence proof
      keys 4 (100), 2 (010)
 Poseidon2Element deployed to: 0x8F4ec854Dd12F1fe79500a1f53D0cbB30f9b6134
 Poseidon3Element deployed to: 0xC66AB83418C20A65C3f8e83B3d11c8C3a6097b6F
 SmtLib deployed to:  0xeF31027350Be2c7439C1b0BE022d49421488b72C
        add 1 leaf and generate the proof for it
        add 2 leaves (depth = 2) and generate the proof of the second one
        add 2 leaves (depth = 2) update 2nd one and generate the proof of the first one
        add 2 leaves (depth = 2) update the 2nd leaf and generate the proof of the second one
        add 2 leaves (depth = 2) update the 2nd leaf and generate the proof of the first one for the previous root state
        add 2 leaves (depth = 2) update the 2nd leaf and generate the proof of the second one for the previous root state
      keys 3 (011), 7 (111)
        add 1 leaf and generate the proof for it
        add 2 leaves (depth = 2) and generate the proof of the second one
        add 2 leaves (depth = 2) update 2nd one and generate the proof of the first one (45ms)
        add 2 leaves (depth = 2) update the 2nd leaf and generate the proof of the second one (38ms)
        add 2 leaves (depth = 2) update the 2nd leaf and generate the proof of the first one for the previous root state
        add 2 leaves (depth = 2) update the 2nd leaf and generate the proof of the second one for the previous root state
      big keys and values
        add 10 big keys and values and generate the proof of the last one (48ms)
    SMT non existence proof
      keys 4 (100), 2 (010)
        add 1 leaf and generate a proof on non-existing leaf
        add 2 leaves (depth = 2) and generate proof on non-existing leaf WITH aux node
        add 2 leaves (depth = 2) and generate proof on non-existing leaf WITHOUT aux node
        add 2 leaves (depth = 2), update the 2nd leaf and generate proof of non-existing leaf WITH aux node (which
        ↪  existed before update)
        add 2 leaves (depth = 2), update the 2nd leaf and generate proof of non-existing leaf WITHOUT aux node
        add 2 leaves (depth = 2), add 3rd leaf and generate proof of non-existence for the 3rd leaf in the previous
        ↪  root state
      keys 3 (011), 7 (111)
        add 1 leaf and generate a proof on non-existing leaf
        add 2 leaves (depth = 2) and generate proof on non-existing leaf WITH aux node
        add 2 leaves (depth = 2) and generate proof on non-existing leaf WITHOUT aux node
        add 2 leaves (depth = 2), update the 2nd leaf and generate proof of non-existing leaf WITH aux node (which
        ↪  existed before update)
        add 2 leaves (depth = 2), update the 2nd leaf and generate proof of non-existing leaf WITHOUT aux node
        add 2 leaves (depth = 2), add 3rd leaf and generate proof of non-existence for the 3rd leaf in the previous
        ↪  root state
      big keys and values
        add 10 leaves and generate a proof on non-existing WITH aux node (56ms)
        add 10 leaves and generate a proof on non-existing WITHOUT aux node (46ms)
      empty tree
        generate proof for some key
        generate proof for some key and zero historical root
    SMT add leaf edge cases
       Positive: add two leaves with maximum depth (less significant bits SET) (84ms)
       Positive: add two leaves with maximum depth (less significant bits NOT SET) (111ms)
       Positive: add two leaves with maximum depth (less significant bits are both SET and NOT SET) (54ms)
       Negative: add two leaves with maximum depth + 1 (less significant bits SET)
       Negative: add two leaves with maximum depth + 1 (less significant bits NOT SET)
       Negative: add two leaves with maximum depth + 1 (less significant bits are both SET and NOT SET

  Root history requests
 Poseidon2Element deployed to: 0x63fea6E447F120B8Faf85B53cdaD8348e645D80E
 Poseidon3Element deployed to: 0xdFdE6B33f13de2CA1A75A6F7169f50541B14f75b
 SmtLib deployed to:  0xaC9fCBA56E42d5960f813B9D0387F3D3bC003338
     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
 Poseidon2Element deployed to: 0xd9140951d8aE6E5F625a02F5908535e16e3af964
 Poseidon3Element deployed to: 0x56D13Eb21a625EdA8438F55DF2C31dC3632034f5
 SmtLib deployed to:  0xE8addD62feD354203d079926a8e563BC1A7FE81e
     comprehensive check (47ms)
     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
     should return correct list and length just after init

  Binary search in SMT root history
    Empty history
 Poseidon2Element deployed to: 0x071586BA1b380B00B793Cc336fe01106B0BFbE6D
 Poseidon3Element deployed to: 0xe70f935c32dA4dB13e7876795f1e175465e6458e
 SmtLib deployed to:  0x3C15538ED063e688c8DF3d571Cb7a0062d2fB18D
       Should return zero root for some search
    One root in the root history
       Should return the first root when equal
       Should return zero when search for less than the first
       Should return the last root when search for greater than the last
    Two roots in the root history
       Should return the first root when search for equal
       Should return the second root when search for equal
       Should return zero when search for less than the first
       Should return the last root when search for greater than the last
    Three roots in the root history
       Should return the first root when equal
       Should return the second root when equal
       Should return the third root when equal
       Should return zero root when search for less than the first
       Should return the last root when search for greater than the last
    Four roots in the root history
       Should return the first root when equal
       Should return the fourth root when equal
       Should return zero when search for less than the first
       Should return the last root when search for greater than the last
    Search in between the values
       Should return the first root when search in between the first and second
       Should return the fourth root when search in between the fourth and the fifth
    Search in array with duplicated values
       Should return the last root among two equal values when search for the value
       Should return the last root among three equal values when search for the value
    Search in array with duplicated values and in between values
       Should search in between the third (1st, 2nd, 3rd equal) and fourth values and return the third
       Should search in between the fifth (4th, 5th equal) and sixth values and return the fifth

  Binary search in SMT proofs
    Zero root proofs
 Poseidon2Element deployed to: 0x7B4f352Cd40114f12e82fC675b5BA8C7582FC513
 Poseidon3Element deployed to: 0xcE0066b1008237625dDDBE4a751827de037E53D2
 SmtLib deployed to:  0x82EdA215Fa92B45a3a76837C65Ab862b6C7564a8
       Should return zero proof for some search
       Should return zero proof for some search back in time
    Non-zero root proofs
       Should return zero proof for some search current time
       Should return zero proof for some search current block

  Edge cases with exceptions
 Poseidon2Element deployed to: 0x8fC8CFB7f7362E44E472c690A6e025B80E406458
 Poseidon3Element deployed to: 0xC7143d5bA86553C06f5730c8dC9f8187a621A8D4
 SmtLib deployed to:  0x359570B3a0437805D0a71457D61AD26a28cAC9A2
     getRootInfo() should throw when root does not exist
     getProofByRoot() should throw when root does not exist

  maxDepth setting tests
 Poseidon2Element deployed to: 0xD5724171C2b7f0AA717a324626050BD05767e2C6
 Poseidon3Element deployed to: 0x70eE76691Bdd9696552AF8d4fd634b3cF79DD529
 SmtLib deployed to:  0x8B190573374637f144AC8D37375d97fd84cBD3a0
     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 groth16 verifier
 Poseidon1Element deployed to: 0x8D81A3DCd17030cD5F23Ac7370e4Efb10D2b3cA4
 Poseidon2Element deployed to: 0xcC4c41415fc68B2fBf70102742A83cDe435e0Ca7
 Poseidon3Element deployed to: 0xa722bdA6968F50778B973Ae2701e90200C564B49
 SmtLib deployed to:  0xc7cDb7A2E5dDa1B7A0E792Fe1ef08ED20A6F56D4
 Groth16VerifierStateTransition contract deployed to address 0x967AB65ef14c58bD4DcfFeaAA1ADb40a022140E5 from
 ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
 StateLib deployed to:  0xe1708FA6bb2844D5384613ef0846F9Bc1e8eC55E
 StateCrossChainLib deployed to:  0x0aec7c174554AF8aEc3680BB58431F6618311510
 CrossChainProofValidator deployed to: 0x8e264821AFa98DD104eEcfcfa7FD9f8D8B320adA
Warning: Potentially unsafe deployment of contracts/state/State.sol:State

    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.

 State contract deployed to address 0x6A59CC73e334b018C9922793d96Df84B538E6fD5 from
 ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
     Zero-knowledge proof of state transition is not valid
     Initial state publishing
     Subsequent state update
  State transition negative cases
 Poseidon1Element deployed to: 0x71a0b8A2245A9770A4D887cE1E4eCc6C1d4FF28c
 Poseidon2Element deployed to: 0xb185E9f6531BA9877741022C92CE858cDCc5760E
 Poseidon3Element deployed to: 0xAe120F0df055428E45b264E7794A18c54a2a3fAF
 SmtLib deployed to:  0x193521C8934bCF3473453AF4321911E7A89E0E12
 Groth16VerifierStub contract deployed to address 0x9Fcca440F19c62CDF7f973eB6DDF218B15d4C71D from
 ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
 StateLib deployed to:  0x01E21d7B8c39dc4C764c19b308Bd8b14B1ba139E
 StateCrossChainLib deployed to:  0x3C1Cb427D20F15563aDa8C249E71db76d7183B6c
 CrossChainProofValidator deployed to: 0x1343248Cbd4e291C6979e70a138f4c774e902561
Warning: Potentially unsafe deployment of contracts/state/State.sol:State

    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.

 State contract deployed to address 0x547382C0D1b23f707918D3c83A77317B71Aa8470 from
 ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cfffb92266
     Old state does not match the latest state
     Old state is genesis but identity already exists
     Old state is not genesis but identity does not yet exist
     ID should not be zero
     New state should not be zero
     Should allow only one unique state per identity

  StateInfo history
 Poseidon1Element deployed to: 0xeAd789bd8Ce8b9E94F5D0FCa99F8787c7e758817
 Poseidon2Element deployed to: 0x95775fD3Afb1F4072794CA4ddA27F2444BCf8Ac3
 Poseidon3Element deployed to: 0xd9fEc8238711935D6c8d79Bef2B9546ef23FC046
 SmtLib deployed to:  0xd3FFD73C53F139cEBB80b6A524bE280955b3f4db
 Groth16VerifierStub contract deployed to address 0x512F7469BcC83089497506b5df64c6E246B39925 from
 ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cfffb92266
 StateLib deployed to:  0x9fD16eA9E31233279975D99D5e8Fc91dd214c7Da
 StateCrossChainLib deployed to:  0xCBBe2A5c3A22BE749D5DDF24e9534f98951983e2
 CrossChainProofValidator deployed to: 0x987e855776C03A4682639eEb14e65b3089EE6310
Warning: Potentially unsafe deployment of contracts/state/State.sol:State

    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.

 State contract deployed to address 0xE8F7d98bE6722d42F29b50500B0E318EF2be4fc8 from
 ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cfffb92266
     should return state history

  GIST proofs
 Poseidon1Element deployed to: 0x75c68e69775fA3E9DD38eA32E554f6BF259C1135
 Poseidon2Element deployed to: 0x572316aC11CB4bc5daf6BDae68f43EA3CCE3aE0e
 Poseidon3Element deployed to: 0x975Ab64F4901Af5f0C96636deA0b9de3419D0c2F
 SmtLib deployed to:  0x4593ed9CbE6003e687e5e77368534bb04b162503
 Groth16VerifierStub contract deployed to address 0xCd7c00Ac6dc51e8dCc773971Ac9221cC582F3b1b from
 ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cfffb92266
 StateLib deployed to:  0x8ac87219a0F5639BC01b470F87BA2b26356CB2B9
 StateCrossChainLib deployed to:  0x94fFA1C7330845646CE9128450F8e6c3B5e44F86
 CrossChainProofValidator deployed to: 0xCa1D199b6F53Af7387ac543Af8e8a34455BBe5E0
Warning: Potentially unsafe deployment of contracts/state/State.sol:State

    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.

 State contract deployed to address 0xf5c4a909455C00B99A90d93b48736F3196DB5621 from
 ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cfffb92266
     Should be correct historical proof by root and the latest root
     Should be correct historical proof by time
     Should be correct historical proof by block

  GIST root history
 Poseidon1Element deployed to: 0x742489F22807ebB4C36ca6cD95c3e1C044B7B6c8
 Poseidon2Element deployed to: 0x1D8D70AD07C8E7E442AD78E4AC0A16f958Eba7F0
 Poseidon3Element deployed to: 0xA9e6Bfa2BF53dE88FEb19761D9b2eE2e821bF1Bf
 SmtLib deployed to:  0x1E3b98102e19D3a164d239BdD190913C2F02E756
 Groth16VerifierStub contract deployed to address 0x3fdc08D815cc4ED3B7F69Ee246716f2C8bCD6b07 from
 ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cfffb92266
 StateLib deployed to:  0x286B8DecD5ED79c962b2d8F4346CD97FF0E2C352
 StateCrossChainLib deployed to:  0xb868Cc77A95a65F42611724AF05Aa2d3B6Ec05F2
 CrossChainProofValidator deployed to: 0x70E5370b8981Abc6e14C91F4AcE823954EFC8eA3
Warning: Potentially unsafe deployment of contracts/state/State.sol:State
    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.

 State contract deployed to address 0x9338CA7d556248055f5751d85cDA7aD6eF254433 from
 ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cfffb92266
     Should search by block and by time return same root
     Should have correct GIST root transitions info

  Set Verifier
 Poseidon1Element deployed to: 0x0c626FC4A447b01554518550e30600136864640B
 Poseidon2Element deployed to: 0xA21DDc1f17dF41589BC6A5209292AED2dF61Cc94
 Poseidon3Element deployed to: 0x2A590C461Db46bca129E8dBe5C3998A8fF402e76
 SmtLib deployed to:  0x158d291D8b47F056751cfF47d1eEcd19FDF9B6f8
 Groth16VerifierStateTransition contract deployed to address 0x2F54D1563963fC04770E85AF819c89Dc807f6a06 from
 ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
 StateLib deployed to:  0xF342E904702b1D021F03f519D6D9614916b03f37
 StateCrossChainLib deployed to:  0x9849832a1d8274aaeDb1112ad9686413461e7101
 CrossChainProofValidator deployed to: 0xa4E00CB342B36eC9fDc4B50b3d527c3643D4C49e
Warning: Potentially unsafe deployment of contracts/state/State.sol:State

    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.

 State contract deployed to address 0x325c8Df4CFb5B068675AFF8f62aA668D1dEc3C4B from
 ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
     Should set groth16 verifier
     Should not set groth16 verifier if not owner
     Should allow groth16 verifier zero address to block any state transition

  Check replacedAt timestamp expirations
 Poseidon1Element deployed to: 0x627b9A657eac8c3463AD17009a424dFE3FDbd0b1
 Poseidon2Element deployed to: 0xa62835D1A6bf5f521C4e2746E1F51c923b8f3483
 Poseidon3Element deployed to: 0x8E45C0936fa1a65bDaD3222bEFeC6a03C83372cE
 SmtLib deployed to:  0xBEe6FFc1E8627F51CcDF0b4399a1e1abc5165f15
 Groth16VerifierStateTransition contract deployed to address 0xC32609C91d6B6b51D48f2611308FEf121B02041f from
 ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
 StateLib deployed to:  0x262e2b50219620226C5fB5956432A88fffd94Ba7
 StateCrossChainLib deployed to:  0x10e38eE9dd4C549b61400Fc19347D00eD3edAfC4
 CrossChainProofValidator deployed to: 0xd753c12650c280383Ce873Cc3a898F6f53973d16
Warning: Potentially unsafe deployment of contracts/state/State.sol:State

    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.

 State contract deployed to address 0xd30bF3219A0416602bE8D482E0396eF332b0494E from
 ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
     Should throw for non-existent state and GIST roots (511ms)

  Get State old Contract and migrate to latest version
    - Check migration

  Negative tests
 StateLib deployed to:  0x06b3244b086cecC40F1e5A826f736Ded68068a0F
     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

  StateInfo history
 StateLib deployed to:  0x74Df809b1dfC099E8cdBc98f6a8D1F5c2C3f66f8
     should return state history
     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
 StateLib deployed to:  0x1D13fF25b10C9a6741DFdce229073bed652197c7
     comprehensive check
     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
 Poseidon1Element deployed to: 0x76cec9299B6Fa418dc71416FF353737AB7933A7D
 Poseidon2Element deployed to: 0xA3307BF348ACC4bEDdd67CCA2f7F0c4349d347Db
 Poseidon3Element deployed to: 0x313F922BE1649cEc058EC0f076664500c78bdc0b
[ 'deploying with BASIC strategy...' ]
 SmtLib deployed to:  0xc0Bb1650A8eA5dDF81998f17B5319afD656f4c11
[ '======== State: deploy started ========' ]
[ 'found defaultIdType 0x0112 for chainId 31337' ]
[ 'deploying Groth16VerifierStateTransition...' ]
 Groth16VerifierStateTransition contract deployed to address 0x5322471a7E37Ac2B8902cFcba84d266b37D811A0 from
  ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
[ 'deploying StateLib...' ]
 StateLib deployed to:  0x90c84237fDdf091b1E63f369AF122EB46000bc70
[ 'deploying StateCrossChainLib...' ]
 StateCrossChainLib deployed to:  0x3D63c50AD04DD5aE394CAB562b7691DD5de7CF6f
[ 'deploying CrossChainProofValidator...' ]
 CrossChainProofValidator deployed to: 0x103A3b128991781EE2c8db0454cA99d67b257923
[ 'deploying State...' ]
[ 'deploying with BASIC strategy...' ]
Warning: Potentially unsafe deployment of contracts/state/State.sol:State

    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.

 State contract deployed to address 0xB9d9e972100a1dD01cd441774b45b5821e136043 from
  ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
[ 'Added id type 0x0100' ]
[ '======== State: deploy completed ========' ]
 Groth16VerifierMTPWrapper Wrapper deployed to: 0x2538a10b7fFb1B78c890c870FC152b10be121f04
[ 'deploying with BASIC strategy...' ]
 CredentialAtomicQueryMTPV2Validator deployed to: 0xdB05A386810c809aD5a77422eb189D36c7f24402
     Validate Genesis User State. Issuer Claim IdenState is in Chain. Revocation State is in Chain (39ms)
     Validation of proof failed (68ms)
     User state is not genesis but latest (55ms)
     The non-revocation issuer state is not expired (71ms)
     The non-revocation issuer state is expired (146ms)
     GIST root expired, Issuer revocation state is not expired (118ms)
     The generated proof is expired (93ms)
     Validate Genesis User State. Issuer Claim IdenState is in Chain. Revocation State is in Chain (75ms)
     check inputIndexOf

  Atomic Sig Validator
 Poseidon1Element deployed to: 0xbf2ad38fd09F37f50f723E35dd84EEa1C282c5C9
 Poseidon2Element deployed to: 0xF66CfDf074D2FFD6A4037be3A669Ed04380Aef2B
 Poseidon3Element deployed to: 0xFC4EE541377F3b6641c23CBE82F6f04388290421
[ 'deploying with BASIC strategy...' ]
 SmtLib deployed to:  0x20d7B364E8Ed1F4260b5B90C41c2deC3C1F6D367
[ '======== State: deploy started ========' ]
[ 'found defaultIdType 0x0112 for chainId 31337' ]
[ 'deploying Groth16VerifierStateTransition...' ]
 Groth16VerifierStateTransition contract deployed to address 0xf5C3953Ae4639806fcbCC3196f71dd81B0da4348 from
  ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
[ 'deploying StateLib...' ]
 StateLib deployed to:  0x90b97E83e22AFa2e6A96b3549A0E495D5Bae61aF
[ 'deploying StateCrossChainLib...' ]
 StateCrossChainLib deployed to:  0xdb54fa574a3e8c6aC784e1a5cdB575A737622CFf
[ 'deploying CrossChainProofValidator...' ]
 CrossChainProofValidator deployed to: 0xDDa0648FA8c9cD593416EC37089C2a2E6060B45c
[ 'deploying State...' ]
[ 'deploying with BASIC strategy...' ]
Warning: Potentially unsafe deployment of contracts/state/State.sol:State

    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.

 State contract deployed to address 0xc6B407503dE64956Ad3cF5Ab112cA4f56AA13517 from
  ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
[ 'Added id type 0x0100' ]
[ '======== State: deploy completed ========' ]
 Groth16VerifierSigWrapper Wrapper deployed to: 0x6A47346e722937B60Df7a1149168c0E76DD6520f
[ 'deploying with BASIC strategy...' ]
 CredentialAtomicQuerySigV2Validator deployed to: 0x2BB8B93F585B43b06F3d523bf30C203d3B6d4BD4
     Validate Genesis User State. Issuer Claim IdenState is in Chain. Revocation State is in Chain (45ms)
     Validation of proof failed (69ms)
     User state is not genesis but latest (56ms)
```

```
      The non-revocation issuer state is not expired (68ms)
      The non-revocation issuer state is expired (171ms)
      GIST root expired, Issuer revocation state is not expired (135ms)
      The generated proof is expired (96ms)
      Validate Genesis User State. Issuer Claim IdenState is in Chain. Revocation State is in Chain (69ms)
      check inputIndexOf

   Atomic V3 Validator
  Poseidon1Element deployed to: 0xB7ca895F81F20e05A5eb11B05Cbaab3DAe5e23cd
  Poseidon2Element deployed to: 0xd0EC100F1252a53322051a95CF05c32f0C174354
  Poseidon3Element deployed to: 0x2d13826359803522cCe7a4Cfa2c1b582303DD0B4
[ 'deploying with BASIC strategy...' ]
  SmtLib deployed to:  0xCa57C1d3c2c35E667745448Fef8407dd25487ff8
[ '======== State: deploy started ========' ]
[ 'found defaultIdType 0x0112 for chainId 31337' ]
[ 'deploying Groth16VerifierStateTransition...' ]
  Groth16VerifierStateTransition contract deployed to address 0xc3023a2c9f7B92d1dd19F488AF6Ee107a78Df9DB from
  ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
[ 'deploying StateLib...' ]
  StateLib deployed to:  0x124dDf9BdD2DdaD012ef1D5bBd77c00F05C610DA
[ 'deploying StateCrossChainLib...' ]
  StateCrossChainLib deployed to:  0xe044814c9eD1e6442Af956a817c161192cBaE98F
[ 'deploying CrossChainProofValidator...' ]
  CrossChainProofValidator deployed to: 0xaB837301d12cDc4b97f1E910FC56C9179894d9cf
[ 'deploying State...' ]
[ 'deploying with BASIC strategy...' ]
Warning: Potentially unsafe deployment of contracts/state/State.sol:State

    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.

  State contract deployed to address 0x0F527785e39B22911946feDf580d87a4E00465f0 from
  ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
[ 'Added id type 0x0212' ]
[ '======== State: deploy completed ========' ]
  Groth16VerifierV3Wrapper Wrapper deployed to: 0x9C85258d9A00C01d00ded98065ea3840dF06f09c
[ 'deploying with BASIC strategy...' ]
  CredentialAtomicQueryV3Validator deployed to: 0x398E4948e373Db819606A459456176D31C3B1F91
      Validate Genesis User State. Issuer Claim IdenState is in published onchain. Revocation State is published onchain.
      ↪  BJJ Proof (62ms)
      Validation of Sig proof failed (91ms)
      User state is not genesis but latest (68ms)
      The non-revocation issuer state is latest (81ms)
      The non-revocation issuer state is expired (154ms)
      GIST root expired, Issuer revocation state is not expired (156ms)
      The generated proof is expired (110ms)
      Validate Genesis User State. Issuer Claim IdenState is in Chain. Revocation State is in Chain (94ms)
      Valid BJJ genesis proof with isBJJAuthEnabled=0 (UserID correspond to the sender) (55ms)
      Validate Genesis User State. Issuer Claim IdenState is in Chain. Revocation State is in Chain. MTP Proof. (44ms)
      Validation of MTP proof failed (77ms)
      User state is not genesis but latest. MTP Proof. (58ms)
      The non-revocation issuer state is not expired. MTP Proof. (78ms)
      The non-revocation issuer state is expired. MTP Proof. (149ms)
      GIST root expired, Issuer revocation state is not expired. MTP Proof. (154ms)
      The generated proof is expired. MTP Proof. (112ms)
      Validate Genesis User State. Issuer Claim IdenState is in Chain. Revocation State is in Chain. MTP Proof. (81ms)
      Valid MTP genesis proof with isBJJAuthEnabled=0 (UserID correspond to the sender) (43ms)
      Valid BJJ genesis proof with isBJJAuthEnabled=0 (UserID does NOT correspond to the sender) (119ms)
      Valid MTP genesis proof with isBJJAuthEnabled=0 (UserID does NOT correspond to the sender) (114ms)
      Validate First User State, Issuer Genesis. BJJ Proof (46ms)
      Validate First User State, Issuer Genesis. BJJ Proof (Challenge should match the sender) (119ms)
      Valid BJJ genesis proof with isBJJAuthEnabled=0 (Invalid Link ID pub signal) (79ms)
      Valid BJJ genesis proof with isBJJAuthEnabled=0 (Proof type should match the requested one in query) (79ms)
      Valid BJJ genesis proof with isBJJAuthEnabled=0 (Invalid nullify pub signal) (79ms)
      Valid BJJ genesis proof with isBJJAuthEnabled=0 (Query hash does not match the requested one) (113ms)

   Embedded ZKP Verifier
  Poseidon1Element deployed to: 0x01cf58e264d7578D4C67022c58A24CbC4C4a304E
  Poseidon2Element deployed to: 0xd038A2EE73b64F30d65802Ad188F27921656f28F
  Poseidon3Element deployed to: 0x666432Ccb747B2220875cE185f487Ed53677faC9
[ 'deploying with BASIC strategy...' ]
  SmtLib deployed to:  0xeC1BB74f5799811c0c1Bff94Ef76Fb40abccbE4a
[ '======== State: deploy started ========' ]
 [ 'found defaultIdType 0x0112 for chainId 31337' ]
 [ 'deploying Groth16VerifierStateTransition...' ]
  Groth16VerifierStateTransition contract deployed to address 0xF6a8aD553b265405526030c2102fda2bDcdDC177 from
   ↪ 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
 [ 'deploying StateLib...' ]
  StateLib deployed to:  0x09120eAED8e4cD86D85a616680151DAA653880F2
 [ 'deploying StateCrossChainLib...' ]
  StateCrossChainLib deployed to:  0x3E661784267F128e5f706De17Fac1Fc1c9d56f30
 [ 'deploying CrossChainProofValidator...' ]
  CrossChainProofValidator deployed to: 0x6732128F9cc0c4344b2d4DC6285BCd516b7E59E6
 [ 'deploying State...' ]
 [ 'deploying with BASIC strategy...' ]
 Warning: Potentially unsafe deployment of contracts/state/State.sol:State

     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.

  State contract deployed to address 0xAe9Ed85dE2670e3112590a2BB17b7283ddF44d9c from
   ↪ 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
 [ 'Added id type 0x0112' ]
 [ '======== State: deploy completed ========' ]
  VerifierLib deployed to:  0x75b0B516B47A27b1819D21B26203Abf314d42CCE
 Warning: Potentially unsafe deployment of
  ↪  contracts/test-helpers/EmbeddedZKPVerifierWrapper.sol:EmbeddedZKPVerifierWrapper

     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.

 EmbeddedZKPVerifierWrapper deployed to: 0xD94A92749C0bb33c4e4bA7980c6dAD0e3eFfb720
 Validator stub deployed to: 0xDf951d2061b12922BFbF22cb17B17f3b39183570
      test submit response
      test submit response v2
      test getZKPRequest and request id exists

   Universal Verifier Linked proofs
  Poseidon1Element deployed to: 0x74ef2B06A1D2035C33244A4a263FF00B84504865
  Poseidon2Element deployed to: 0xF5b81Fe0B6F378f9E6A3fb6A6cD1921FCeA11799
  Poseidon3Element deployed to: 0x67baFF31318638F497f4c4894Cd73918563942c8
 [ 'deploying with BASIC strategy...' ]
  SmtLib deployed to:  0x6533158b042775e2FdFeF3cA1a782EFDbB8EB9b1
 [ '======== State: deploy started ========' ]
 [ 'found defaultIdType 0x0112 for chainId 31337' ]
 [ 'deploying Groth16VerifierStateTransition...' ]
  Groth16VerifierStateTransition contract deployed to address 0x73C68f1f41e4890D06Ba3e71b9E9DfA555f1fb46 from
   ↪ 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
 [ 'deploying StateLib...' ]
  StateLib deployed to:  0xD2D5e508C82EFc205cAFA4Ad969a4395Babce026
 [ 'deploying StateCrossChainLib...' ]
  StateCrossChainLib deployed to:  0x2b639Cc84e1Ad3aA92D4Ee7d2755A6ABEf300D72
 [ 'deploying CrossChainProofValidator...' ]
  CrossChainProofValidator deployed to: 0xF85895D097B2C25946BB95C4d11E2F3c035F8f0C
 [ 'deploying State...' ]
 [ 'deploying with BASIC strategy...' ]
 Warning: Potentially unsafe deployment of contracts/state/State.sol:State

     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.

  State contract deployed to address 0x7bdd3b028C4796eF0EAf07d11394d0d9d8c24139 from
   ↪ 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
 [ 'Added id type 0x0112' ]
 [ '======== State: deploy completed ========' ]
  VerifierLib deployed to:  0x47c05BCCA7d57c87083EB4e586007530eE4539e9
 [ 'deploying with BASIC strategy...' ]
 Warning: Potentially unsafe deployment of contracts/verifiers/UniversalVerifier.sol:UniversalVerifier
     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.

  UniversalVerifier deployed to: 0x773330693cb7d5D233348E25809770A32483A940
  Groth16VerifierV3Wrapper Wrapper deployed to: 0x52173b6ac069619c206b9A0e75609fC92860AB2A
 [ 'deploying with BASIC strategy...' ]
  CredentialAtomicQueryV3Validator deployed to: 0x40A633EeF249F21D95C8803b7144f19AAfeEF7ae
      should linked proof validation pass
      should linked proof validation fail
   Universal Verifier submitZKPResponseV2 SigV2 validators
  CrossChainProofValidator deployed to: 0x6f2E42BB4176e9A7352a8bF8886255Be9F3D2d13
  Poseidon1Element deployed to: 0xA3f7BF5b0fa93176c260BBa57ceE85525De2BaF4
  Poseidon2Element deployed to: 0x25A1DF485cFBb93117f12fc673D87D1cddEb845a
  Poseidon3Element deployed to: 0xD855cE0C298537ad5b5b96060Cf90e663696bbf6
 [ 'deploying with BASIC strategy...' ]
  SmtLib deployed to:  0xF45B1CdbA9AACE2e9bbE80bf376CE816bb7E73FB
 [ '======== State: deploy started ========' ]
 [ 'found defaultIdType 0x0112 for chainId 31337' ]
 [ 'deploying Groth16VerifierStateTransition...' ]
  Groth16VerifierStateTransition contract deployed to address 0x22b1c5C2C9251622f7eFb76E356104E5aF0e996A from
   ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
 [ 'deploying StateLib...' ]
  StateLib deployed to:  0x5A569Ad19272Afa97103fD4DbadF33B2FcbaA175
 [ 'deploying StateCrossChainLib...' ]
  StateCrossChainLib deployed to:  0x696358bBb1a743052E0E87BeD78AAd9d18f0e1F4
 [ 'deploying CrossChainProofValidator...' ]
  CrossChainProofValidator deployed to: 0x7036124464A2d2447516309169322c8498ac51e3
 [ 'deploying State...' ]
 [ 'deploying with BASIC strategy...' ]
 Warning: Potentially unsafe deployment of contracts/state/State.sol:State

     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.

  State contract deployed to address 0x5c932424AcBfab036969b3B9D94bA9eCbae7565D from
   ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
 [ 'Added id type 0x01A1' ]
 [ 'Added id type 0x0102' ]
 [ '======== State: deploy completed ========' ]
  VerifierLib deployed to:  0xE7FF84Df24A9a252B6E8A5BB093aC52B1d8bEEdf
 [ 'deploying with BASIC strategy...' ]
 Warning: Potentially unsafe deployment of contracts/verifiers/UniversalVerifier.sol:UniversalVerifier

     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.

  UniversalVerifier deployed to: 0x04F339eC4D75Cf2833069e6e61b60eF56461CD7C
 Validator stub deployed to: 0x3de00f44ce68FC56DB0e0E33aD4015C6e78eCB39
      Test submit response V2 (38ms)
      Test submit response V2 with disable/enable functionality (61ms)
      Test submit response V2 check whitelisted functionality (60ms)

   Universal Verifier events
  Poseidon1Element deployed to: 0xa195ACcEB1945163160CD5703Ed43E4f78176a54
  Poseidon2Element deployed to: 0x6212cb549De37c25071cF506aB7E115D140D9e42
  Poseidon3Element deployed to: 0x6F9679BdF5F180a139d01c598839a5df4860431b
 [ 'deploying with BASIC strategy...' ]
  SmtLib deployed to:  0xf4AE7E15B1012edceD8103510eeB560a9343AFd3
 [ '======== State: deploy started ========' ]
 [ 'found defaultIdType 0x0112 for chainId 31337' ]
 [ 'deploying Groth16VerifierStateTransition...' ]
  Groth16VerifierStateTransition contract deployed to address 0x0bF7dE8d71820840063D4B8653Fd3F0618986faF from
   ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
 [ 'deploying StateLib...' ]
  StateLib deployed to:  0xc981ec845488b8479539e6B22dc808Fb824dB00a
 [ 'deploying StateCrossChainLib...' ]
  StateCrossChainLib deployed to:  0x5E5713a0d915701F464DEbb66015adD62B2e6AE9
 [ 'deploying CrossChainProofValidator...' ]
  CrossChainProofValidator deployed to: 0x97fd63D049089cd70D9D139ccf9338c81372DE68
 [ 'deploying State...' ]
 [ 'deploying with BASIC strategy...' ]
 Warning: Potentially unsafe deployment of contracts/state/State.sol:State

     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.

  State contract deployed to address 0x43cA9bAe8dF108684E5EAaA720C25e1b32B0A075 from
   ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
 [ 'Added id type 0x0112' ]
 [ '======== State: deploy completed ========' ]
  VerifierLib deployed to:  0x59C4e2c6a6dC27c259D6d067a039c831e1ff4947
 [ 'deploying with BASIC strategy...' ]
```

```
Warning: Potentially unsafe deployment of contracts/verifiers/UniversalVerifier.sol:UniversalVerifier

    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.

 UniversalVerifier deployed to: 0x687bB6c57915aa2529EfC7D2a26668855e022fAE
 Groth16VerifierSigWrapper Wrapper deployed to: 0x49149a233de6E4cD6835971506F47EE5862289c1
[ 'deploying with BASIC strategy...' ]
 CredentialAtomicQuerySigV2Validator deployed to: 0xAe2563b4315469bF6bdD41A6ea26157dE57Ed94e
     Check ZKPRequestSet event
 Poseidon1Element deployed to: 0x4CF4dd3f71B67a7622ac250f8b10d266Dc5aEbcE
 Poseidon2Element deployed to: 0x2498e8059929e18e2a2cED4e32ef145fa2F4a744
 Poseidon3Element deployed to: 0x447786d977Ea11Ad0600E193b2d07A06EfB53e5F
[ 'deploying with BASIC strategy...' ]
 SmtLib deployed to:  0x6DcBc91229d812910b54dF91b5c2b592572CD6B0
[ '======== State: deploy started ========' ]
[ 'found defaultIdType 0x0112 for chainId 31337' ]
[ 'deploying Groth16VerifierStateTransition...' ]
 Groth16VerifierStateTransition contract deployed to address 0x245e77E56b1514D77910c9303e4b44dDb44B788c from
  ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
[ 'deploying StateLib...' ]
 StateLib deployed to:  0xE2b5bDE7e80f89975f7229d78aD9259b2723d11F
[ 'deploying StateCrossChainLib...' ]
 StateCrossChainLib deployed to:  0xC6c5Ab5039373b0CBa7d0116d9ba7fb9831C3f42
[ 'deploying CrossChainProofValidator...' ]
 CrossChainProofValidator deployed to: 0x4ea0Be853219be8C9cE27200Bdeee36881612FF2
[ 'deploying State...' ]
[ 'deploying with BASIC strategy...' ]
Warning: Potentially unsafe deployment of contracts/state/State.sol:State

    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.

 State contract deployed to address 0x9155497EAE31D432C0b13dBCc0615a37f55a2c87 from
  ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
[ 'Added id type 0x0112' ]
[ '======== State: deploy completed ========' ]
 VerifierLib deployed to:  0xc1EeD9232A0A44c2463ACB83698c162966FBc78d
[ 'deploying with BASIC strategy...' ]
Warning: Potentially unsafe deployment of contracts/verifiers/UniversalVerifier.sol:UniversalVerifier

    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.

 UniversalVerifier deployed to: 0xfaE849108F2A63Abe3BaB17E21Be077d07e7a9A2
 Groth16VerifierSigWrapper Wrapper deployed to: 0x12456Fa31e57F91B70629c1196337074c966492a
[ 'deploying with BASIC strategy...' ]
 CredentialAtomicQuerySigV2Validator deployed to: 0xce830DA8667097BB491A70da268b76a081211814
     Check ZKPRequestUpdate event

  Universal Verifier MTP & SIG validators
 Poseidon1Element deployed to: 0x05bB67cB592C1753425192bF8f34b95ca8649f09
 Poseidon2Element deployed to: 0xa85EffB2658CFd81e0B1AaD4f2364CdBCd89F3a1
 Poseidon3Element deployed to: 0x8aAC5570d54306Bb395bf2385ad327b7b706016b
[ 'deploying with BASIC strategy...' ]
 SmtLib deployed to:  0x64f5219563e28EeBAAd91Ca8D31fa3b36621FD4f
[ '======== State: deploy started ========' ]
[ 'found defaultIdType 0x0112 for chainId 31337' ]
[ 'deploying Groth16VerifierStateTransition...' ]
 Groth16VerifierStateTransition contract deployed to address 0x1757a98c1333B9dc8D408b194B2279b5AFDF70Cc from
  ↪  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
[ 'deploying StateLib...' ]
 StateLib deployed to:  0x6484EB0792c646A4827638Fc1B6F20461418eB00
[ 'deploying StateCrossChainLib...' ]
 StateCrossChainLib deployed to:  0xf201fFeA8447AB3d43c98Da3349e0749813C9009
[ 'deploying CrossChainProofValidator...' ]
 CrossChainProofValidator deployed to: 0xA75E74a5109Ed8221070142D15cEBfFe9642F489
[ 'deploying State...' ]
[ 'deploying with BASIC strategy...' ]
Warning: Potentially unsafe deployment of contracts/state/State.sol:State

    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.
 State contract deployed to address 0x840748F7Fd3EA956E5f4c88001da5CC1ABCBc038 from
  →  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
[ 'Added id type 0x0112' ]
[ '======== State: deploy completed ========' ]
 VerifierLib deployed to:  0x04f1A5b9BD82a5020C49975ceAd160E98d8B77Af
[ 'deploying with BASIC strategy...' ]
Warning: Potentially unsafe deployment of contracts/verifiers/UniversalVerifier.sol:UniversalVerifier

    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.

 UniversalVerifier deployed to: 0xbFD3c8A956AFB7a9754C951D03C9aDdA7EC5d638
Validator stub deployed to: 0x38F6F2caE52217101D7CA2a5eC040014b4164E6C
     Test get state address
     Test add, get ZKPRequest, requestIdExists, getZKPRequestsCount (44ms)
     Test add, get ZKPRequest, requestIdExists, getZKPRequestsCount with multiple set
     Test submit response
     Check access control
     Check disable/enable functionality (38ms)
 Groth16VerifierMTPWrapper Wrapper deployed to: 0x837a41023CF81234f89F956C94D676918b4791c1
[ 'deploying with BASIC strategy...' ]
 CredentialAtomicQueryMTPV2Validator deployed to: 0x04d7478fDF318C3C22cECE62Da9D78ff94807D77
     Check whitelisted validators (67ms)
     Check updateZKPRequest
     updateZKPRequest - not existed request

  Universal Verifier V3 validator
 Poseidon1Element deployed to: 0x04d7478fDF318C3C22cECE62Da9D78ff94807D77
 Poseidon2Element deployed to: 0xd9abC93F81394Bd161a1b24B03518e0a570bDEAd
 Poseidon3Element deployed to: 0xcB0f2a13098f8e841e6Adfa5B17Ec00508b27665
[ 'deploying with BASIC strategy...' ]
 SmtLib deployed to:  0x37D31345F164Ab170B19bc35225Abc98Ce30b46A
[ '======== State: deploy started ========' ]
[ 'found defaultIdType 0x0112 for chainId 31337' ]
[ 'deploying Groth16VerifierStateTransition...' ]
 Groth16VerifierStateTransition contract deployed to address 0x6345e50859b0Ce82D8A495ba9894C6C81de385F3 from
  →  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
[ 'deploying StateLib...' ]
 StateLib deployed to:  0x88D1aF96098a928eE278f162c1a84f339652f95b
[ 'deploying StateCrossChainLib...' ]
 StateCrossChainLib deployed to:  0x7Ce73F8f636C6bD3357A0A8a59e0ab6462C955B0
[ 'deploying CrossChainProofValidator...' ]
 CrossChainProofValidator deployed to: 0x87c470437282174b3f8368c7CF1Ac03bcAe57954
[ 'deploying State...' ]
[ 'deploying with BASIC strategy...' ]
Warning: Potentially unsafe deployment of contracts/state/State.sol:State

    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.

 State contract deployed to address 0x96E303b6D807c0824E83f954784e2d6f3614f167 from
  →  0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
[ 'Added id type 0x0212' ]
[ '======== State: deploy completed ========' ]
 VerifierLib deployed to:  0xd3b893cd083f07Fe371c1a87393576e7B01C52C6
 Groth16VerifierV3Wrapper deployed to: 0x3BFbbf82657577668144921b96aAb72BC170646C
[ 'deploying with BASIC strategy...' ]
 CredentialAtomicQueryV3Validator deployed to: 0x930b218f3e63eE452c13561057a8d5E61367d5b7
[ 'deploying with BASIC strategy...' ]
Warning: Potentially unsafe deployment of contracts/verifiers/UniversalVerifier.sol:UniversalVerifier

    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.

 UniversalVerifier deployed to: 0x38c76A767d45Fc390160449948aF80569E2C4217
     Test submit response (49ms)
     Test submit response V2
     Test submit response fails with UserID does not correspond to the sender (68ms)
     Test submit response fails with Issuer is not on the Allowed Issuers list (50ms)
     Test submit response fails with Invalid Link ID pub signal (51ms)
     Test submit response fails with Proof type should match the requested one in query (47ms)
     Test submit response fails with Invalid nullify pub signal (46ms)
    Test submit response fails with Query hash does not match the requested one (79ms)
    Test submit response fails with Generated proof is outdated (114ms)


  316 passing (29s)
  1 pending
```

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

A Appendix: Deployment Scripts

The security review of the **Privado ID** protocol was extended beyond the smart contracts to deployment scripts. **Nethermind Security** team reviewed the files that are related to the deployment of the smart contracts in scope.

	File
1	helpers/constants.ts
2	helpers/DeployHelper.ts
3	helpers/helperUtils.ts
4	ignition/modules/authV2Validator.ts
5	ignition/modules/crate2AddressAnchor.ts
6	ignition/modules/credentialAtomicQueryMTPV2Validator.ts
7	ignition/modules/credentialAtomicQuerySigV2Validator.ts
8	ignition/modules/credentialAtomicQueryV3Validator.ts
9	ignition/modules/libraries.ts
10	ignition/modules/linkedMultiQuery.ts
11	ignition/modules/mcPayment.ts
12	ignition/modules/state.ts
13	ignition/modules/universalVerifier.ts
14	ignition/modules/vcPayment.ts
15	scripts/deploy/deployCreate2AddressAnchor.ts
16	scripts/deploy/deployLibraries.ts
17	scripts/deploy/deployMCPayment.ts
18	scripts/deploy/deployPoseidon.ts
19	scripts/deploy/deployState.ts
20	scripts/deploy/deployUniversalVerifier.ts
21	scripts/deploy/deployVCPayment.ts
22	scripts/deploy/deployValidators.ts

A.1 Using CREATE2 on ZKsync Era

File(s): .

Description: The Privado ID aims to keep uniformity of contract addresses between many EVM-compatible networks. To achieve this, the `CREATE2` opcode is used, which allows for controlling the parameters from which the address is created. This is possible since many EVM chains use the same formula to derive the address. However, the ZKsync Era, which is an EVM-compatible chain, uses a distinct address derivation method compared to Ethereum. This means that the same bytecode deployed on Ethereum and ZKsync will have different addresses. More information can be found under this link: ZKsync Era Address Derivation. If the Privado ID system plans to deploy on ZKsync Era, the address uniformity may not be achievable on this network, and could cause downstream issues.

Recommendation(s): Consider creating a strategy for handling the ZKsync Era addresses differences. It may involve clearly documenting this special case to users and developers.

Status: Acknowledged

Update from the client: If we deploy to the ZKSync era, the relevant documentation will be prepared

A.2 Check of unified addresses

File(s): *

Description: The deployed contracts are checked against the address defined in `constants.ts` only in the case of `Create2AddressAnchor`, but not in other scripts. Consider always checking if the deployed contracts match the predicted addresses during the deployment.

A.3 Default strategy

File(s): *

Description: The default strategy in `deployCreate2AddressAnchor.ts` is `create2`, while in all other scripts, it's `base`. Consider unifying the default strategy to avoid potential issues during tests or deployment.

A.4 Incorrect spelling

File(s): `crate2AddressAnchor.ts`

Description: The `crate2AddressAnchor.ts` should be `crate2AddressAnchor.ts`.

Status: Fixed

A.5 Lack of `verify:verify` script

File(s): `package.json`

Description: The `verifyContract(...)` function calls `run("verify:verify", ...)` to verify the proxy, proxy admin, and implementation contracts. However, the `verify:verify` script is not defined.

A.6 Lack of deployment script for CrossChainValidator

File(s): *

Description: The CrossChainValidator contract is not deployed, the scripts do not contain module for it and deployment script.

Recommendation(s): Consider implementing deployment scripts for the CrossChainValidator.

Status: Won't fix

Update from the client: The CrossChainValidator is supposed to be deployed in scope of State contract deployment

A.7 Not all deployed contracts are verified

File(s): *

Description: The `verifyContract(...)` is not called for every deployed contracts. Consider adding verification to all deployed contracts.

A.8 Order of the deployment is not ensured

File(s): *

Description: The deployment scripts should be run in a defined order to ensure the correct deployment. In particular, the Create2AddressAnchor (dummy implementation) should be deployed first since, without it, the deployment of other proxies would fail.

Recommendation(s): Consider ensuring the correct order of the deployment.

A.9 Removing unnecessary plugins

File(s): *

Description: The current deployment scripts use many hardhat plugins: `ethers`, `network`, `ignition`, and `upgrades`. The plugin `upgrades` is a community plugin developed by the OpenZeppelin. This plugin is used to deploy a proxy or perform the upgrade. However, those actions may be performed by the `ignition` plugin (docs about the upgrade: link). Consider performing proxy deployments and upgrades with the `ignition` instead of `upgrades`. Using fewer plugins removes complexity and makes script maintenance easier.

A.10 Script for deployment of UniversalVerifier fails

File(s): `deployUniversalVerifier.ts`

Description: The `deployUniversalVerifier.ts` fails at the `tmpContractDeployments.remove()` due to incorrect path.

Status: Fixed

Update from the client: tmpContractDeployments creation looks redundant. All operations relevant to tmpContractDeployments are removed.

A.11 The `create2` strategy is never tested

File(s): `tests/`

Description: Currently there is no option to run the tests with the "create2" strategy and are always run with "basic" strategy. This leads to never testing the "create2" strategy which is the desired way of deployment.

Recommendation(s): Consider testing with the "create2" strategy.

A.12 Tests are failing with `create2` strategy

File(s): `tests/`

Description: The current tests are all being run with contracts deployed in the "basic" strategy. If the contracts are deployed with the "create2" strategy, the tests fail. An example of the `test/validators/v3/index.ts` test modification is below:

```
describe("Atomic V3 Validator", function () {
  let state: any, v3Validator;

  async function deployContractsFixture() {
    const deployHelper = await DeployHelper.initialize(null, true);

    const { state: stateContract } = await deployHelper.deployStateWithLibraries(["0x0212"]);
    const contracts = await deployHelper.deployValidatorContractsWithVerifiers(
      "v3",
      await stateContract.getAddress(),
      "create2" //@audit: modified version
    );
    const validator = contracts.validator;

    return {
      stateContract,
      validator,
    };
  }
```

If we run tests with the applied modification, the following error appears:

```
332 passing (18s)
  1 failing

  1) Atomic V3 Validator
       "before each" hook for "Validate Genesis User State. Issuer Claim IdenState is in published onchain. Revocation
        ↪  State is published onchain. BJJ Proof":
     Error: VM Exception while processing transaction: reverted with custom error
      ↪  'OwnableUnauthorizedAccount("0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266")'
```

The above error message is thrown during the upgrading of the V3 validator proxy contract in `deployValidatorContractsWithVerifiers(...)`:

```
const validatorAddress = await validator.getAddress();
await upgrades.forceImport(validatorAddress, Create2AddressAnchorFactory);
validator = await upgrades.upgradeProxy(validatorAddress, ValidatorFactory, {
unsafeAllow: ["external-library-linking"],
redeployImplementation: "always",
call: {
  fn: "initialize",
  args:
    validatorType != "lmk"
      ? [
          stateContractAddress,
          await groth16VerifierWrapper.getAddress(),
          await owner.getAddress(),
        ]
      : [await groth16VerifierWrapper.getAddress(), await owner.getAddress()],
},
});
```

It most likely comes from the fact that the proxy was deployed by one address earlier in the tests, but the upgrade is being done by another address, making it an incorrect caller. Another error throws when the tests are run with the selected test, e.g., only for `Atomic V3 Validator`:

```
  0 passing (2s)
  1 failing

  1) Atomic V3 Validator
       "before each" hook for "Validate Genesis User State. Issuer Claim IdenState is in published onchain. Revocation
       ↪  State is published onchain. BJJ Proof":
     HardhatPluginError: The deployment wasn't successful, there were failures:

Failures:

  * CredentialAtomicQueryV3ValidatorProxyModule#TransparentUpgradeableProxy
  1: Simulating the transaction failed with error:
      Reverted with custom error FailedContractCreation("0xba5Ed099633D3B313e4D5F7bdc1305d3c28ba5Ed")
```

This error is thrown during the attempt of deployment of the validator proxy:

```
if (!create2AlreadyDeployed) {
    // Deploying Validator contract to predictable address but with dummy implementation
    validator = (
      await ignition.deploy(validatorModule, {
        strategy: deployStrategy,
      })
    ).proxy;
    await validator.waitForDeployment();
}
```

This is because of the attempt to deploy the validator proxy, but the dummy implementation has never been deployed before in this test scenario.

Recommendation(s): Consider modifying tests and functions in `DeployHelper` to pass tests of deployment with the "create2" strategy.

A.13 The Auth2Validator is not being deployed

File(s): *

Description: The Auth2Validator module is defined in the `ingition/modules/auth2Validator.ts` as `AuthV2ValidatorProxyModule`. However, it is never deployed in any deployment script in `scripts/deploy`. Moreover, there are tests at `test/validators/authv2/index.ts` which deploy the Auth2Validator contract and perform tests, but with the current scripts, this contract would not be deployed.

Recommendation(s): Add deployment of Auth2Validator to the deployment scripts.

A.14 The EthIdentityValidator is not being deployed

File(s): *

Description: The EthIdentityValidator is not in the deployment scripts. It does not have a defined module and does not exist in any deployment scripts or tests. Additionally, the function `deployValidatorContracts(...)` that is responsible for deploying validators does not contain EthIdentityValidator in switch statements for `validatorContractName` and `validatorModule`. It also is not taken into account in the upgrade part:

```
validator = await upgrades.upgradeProxy(validatorAddress, ValidatorFactory, {
        unsafeAllow: ["external-library-linking"],
        redeployImplementation: "always",
        call: {
          fn: "initialize",
          // @audit: EthIdentityValidator is not taken into account here
          // it takes state contract and owner addresses in initialize(...)
          args:
            validatorType != "lmk"
              ? [
                  stateContractAddress,
                  await groth16VerifierWrapper.getAddress(),
                  await owner.getAddress(),
                ]
              : [await groth16VerifierWrapper.getAddress(), await owner.getAddress()],
        },
      });
```

Recommendation(s): Consider adding the EthIdentityValidator to the deployment, maintenance, upgrade scripts, and tests.

A.15 The `deployCrossChainVerifierWithRequests` script does not whitelist LinkedMultiQueryValidator and auth validators

File(s): `deployCrossChainVerifierWithRequests.ts`

Description: The `deployCrossChainVerifierWithRequests` script does not deploy and whitelist the LinkedMultiQueryValidator contract. This leads to deploying the UniversalVerifier without the LMQ validator. Additionally, the authentication methods are not deployed and added to the UniversalVerifier as well.

Recommendation(s): Consider whitelisting every validator and authentication method in the `deployCrossChainVerifierWithRequests.ts`.

A.16 The deployCrossChainVerifierWithRequests script allows only basic strategy

File(s): `deployCrossChainVerifierWithRequests.ts`

Description: The `deployCrossChainVerifierWithRequests.ts` deployment script does not allow to pass the strategy from the config, in effect always deploying the contracts with `create` instead of `create2`. It is possible that those contracts were deployed with `deployValidators.ts` but the option to use `create2` should probably be possible.

Recommendation(s): Consider adding option to deploy contracts with `create2`.

A.17 The logic in `DeployHelpers.deployState(...)` may be incorrect for Polygon networks

File(s): `DeployHelpers.ts`

Description: If the deployment script for State will be used on Polygon Amoy tesnet and Polygon PoS mainnet, then the following logic in `DeployHelpers.deployState(...)` would be incorrect:

```
if (deployStrategy === "create2") {
  state = await getUnifiedContract(contractsInfo.STATE.name);

  if (state) {
    let version;
    try {
      version = await state.VERSION();
    } catch (e) {
      create2AlreadyDeployed = true;
      Logger.warning(
        `Create2AnchorAddress implementation already deployed to TransparentUpgradeableProxy of
        ↪  ${contractsInfo.STATE.name}.`,
      );
    }
```

The contract address would be fetched from `constants.ts` and check if, under this address, there is a deployed contract. However, the address from `constants.ts` does not match the address of the State contract deployed at the Polygon testnet and mainnet. This would deploy another State contract on those networks. In case the currently deployed State contracts on Polygon networks should stay at the same addresses, the script would work incorrectly since it would deploy a new State contract at the unified address. The script works correctly if the State contracts at Polygon networks should be deployed at unified addresses. Note that, most likely, this script would have to be run on Polygon since the CrossChainValidator is deployed in `deployState(...)`.

References

[1] Benjaminion. *BLS12-381*. HackMD. 2023. URL: https://hackmd.io/@benjaminion/bls12-381.

[2] Daniel J. Bernstein and Tanja Lange. *Cryptography Engineering*. ToC Cryptobook, 2015. URL: https://toc.cryptobook.us/book.pdf.

[3] Daniel J. Bernstein and Tanja Lange. *SafeCurves: Choosing Safe Curves for Elliptic-Curve Cryptography*. 2014. URL: https://safecurves.cr.yp.to/.

[4] J. Bos et al. "Selecting Elliptic Curves for Cryptography: An Efficiency and Security Analysis". In: *Journal of Cryptology* 32.3 (2019), pp. 468–494. URL: https://link.springer.com/article/10.1007/s00145-018-9280-5.

[5] Vitalik Buterin. *Quadratic Arithmetic Programs: From Zero to Hero*. Medium. 2016. URL: https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649.

[6] IETF CFRG. *Pairing-Friendly Curves for Cryptography*. 2022. URL: https://www.ietf.org/archive/id/draft-irtf-cfrg-pairing-friendly-curves-08.html#section-4.

[7] Circom Documentation. *Flags and Options Related to the R1CS Optimization*. 2023. URL: https://docs.circom.io/getting-started/compilation-options/#flags-and-options-related-to-the-r1cs-optimization.

[8] Ethereum Foundation. *EIP-196: Precompiled Contracts for Optimal Ate Pairing Check on the Curve*. 2017. URL: https://eips.ethereum.org/EIPS/eip-196.

[9] Ethereum Foundation. *EIP-197: Precompiled Contracts for Optimal Ate Pairing on the Curve*. 2017. URL: https://eips.ethereum.org/EIPS/eip-197.

[10] Ethereum Foundation. *EIP-2537: BLS12-381 Curve Operations*. 2020. URL: https://eips.ethereum.org/EIPS/eip-2537.

[11] Iden3. *snarkjs README*. URL: https://github.com/iden3/snarkjs?tab=readme-ov-file#7-prepare-phase-2.

[12] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC, 2007. URL: https://ia800907.us.archive.org/1/items/strategic_intelligence_network/communication/encryption/introduction_to_modern_cryptography.pdf.

[13] Arjen K. Lenstra et al. "The Number Field Sieve". In: *ResearchGate* (1993). URL: https://www.researchgate.net/publication/221590545_The_Number_Field_Sieve.

[14] A. Menezes, T. Oliveira, and F. Rodríguez-Henríquez. "Reducing the Security Level of Pairing-Based Cryptography". In: *IACR Cryptology ePrint Archive* (2016). URL: https://eprint.iacr.org/2016/1102.pdf.

[15] National Institute of Standards and Technology (NIST). *Recommendation for Key Management – Part 1: General (Rev. 5)*. 2020. URL: https://csrc.nist.gov/pubs/sp/800/57/pt1/r5/final.

[16] F. M. Vercauteren. "A Subexponential Algorithm for Discrete Logarithms over Finite Fields of Small Characteristic". In: *Advances in Cryptology – EUROCRYPT 2016*. Springer, 2016. URL: https://link.springer.com/chapter/10.1007/978-3-662-53018-4_20.