Security Review Report
NM-0411-0446 WorldID Identity Manager upgrade

NETHERMIND
SECURITY

(February 05, 2025)

Contents

1   Executive Summary

This document outlines the security review conducted by Nethermind Security for Worldcoin's WorldID Identity Manager smart contracts. The audit focuses on the updates made to the WorldID Identity Manager Implementation contract, with the goal of processing compressed proofs.

The existing World ID manager will be upgraded to the new `WorldIDIdentityManagerImplV3` implementation. This update redefines the `verifyProof` function, enabling it to utilize the Semaphore Verifier's `verifyCompressedProof` function when a compressed proof is provided.

Moreover, the same updates have been implemented to the `WorldIDBridge` contract deployed on Worldchain, enabling it to use the compressed proof function.

The audited code comprises of 138 lines of code written in the Solidity language, and the audit was performed using (a) manual analysis of the codebase, (b) automated analysis tools, (c) simulation of the smart contract.

Along this document, we report one point of attention classified as `Best Practices`. The issues are summarized in Fig. 1.

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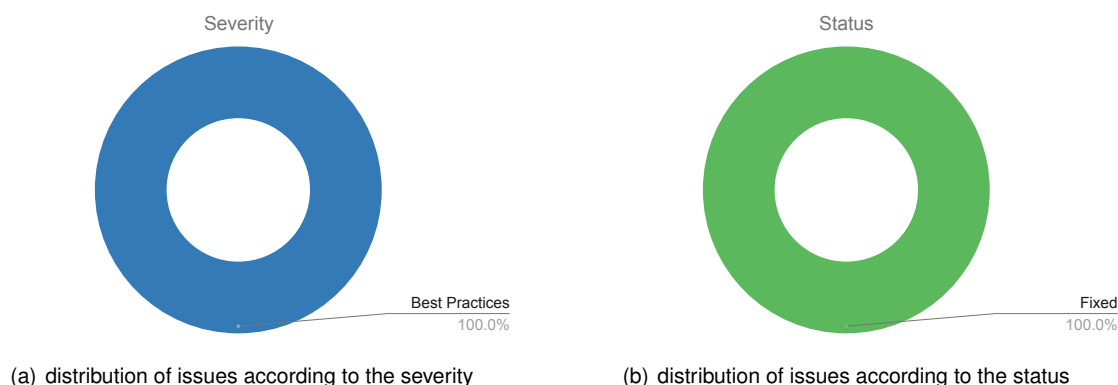

(a) distribution of issues according to the severity

(b) distribution of issues according to the status

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

Summary of the Audit

Audit Type	Security Review
Draft Report	February 05, 2025
Final Report	February 05, 2025
Methods	Manual Review, Automated analysis
Repository	world-id-contracts, world-id-state-bridge
Initial Commit Hash	**world-id-contracts:**
	bb48d8a5c57cbca3023be92d8aaaeaa6b4463e81
	world-id-state-bridge:
	5273e8d35a58286911eccbb778dba52e89eeb8aa
Final Commit Hash	**world-id-contracts:**
	64bdbb42609f47d2a4f721790c38cec50537dd7a
	world-id-state-bridge:
	5273e8d35a58286911eccbb778dba52e89eeb8aa
Documentation Assessment	High
Test Suite Assessment	High

2   Audited Files

2.1   world-id-contracts

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/WorldIDIdentityManagerImplV3.sol	31	68	219.4%	8	107
2	src/interfaces/ISemaphoreVerifier.sol	7	22	314.3%	2	31
	Total	**38**	**90**	**236.8%**	**10**	**138**

2.2   world-id-state-bridge

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/abstract/WorldIDBridge.sol	88	116	131.8%	44	248
	Total	**88**	**116**	**131.8%**	**44**	**248**

3   Summary of Issues

	Finding	Severity	Update
1	Comments/Natspec notes	Best Practices	Fixed

4   System Overview

The Worldcoin team introduced the `WorldIDIdentityManagerImplV3` contract to upgrade the existing `WorldIDIdentityManagerImplV2` implementation of the World ID identity manager. The primary goal of this upgrade is to enable the contract to process compressed ZK proofs. A proof is considered "compressed" when the last four uint values are zero, effectively reducing the proof size from 8 uints to 4 uints. Similar updates are implemented in the `WorldIDBridge` abstract contract.

The `SemaphoreVerifier` contract includes the `verifyCompressedProof` function that can be called directly to verify compressed proofs. As part of the upgrade, the `verifyProof` function has been modified to check whether the provided proof is compressed, and it will call the appropriate `SemaphoreVerifier` function based on the proof type.

```
function verifyProof(
        uint256 root,
        uint256 signalHash,
        uint256 nullifierHash,
        uint256 externalNullifierHash,
        uint256[8] calldata proof
) public view virtual override onlyProxy onlyInitialized
```
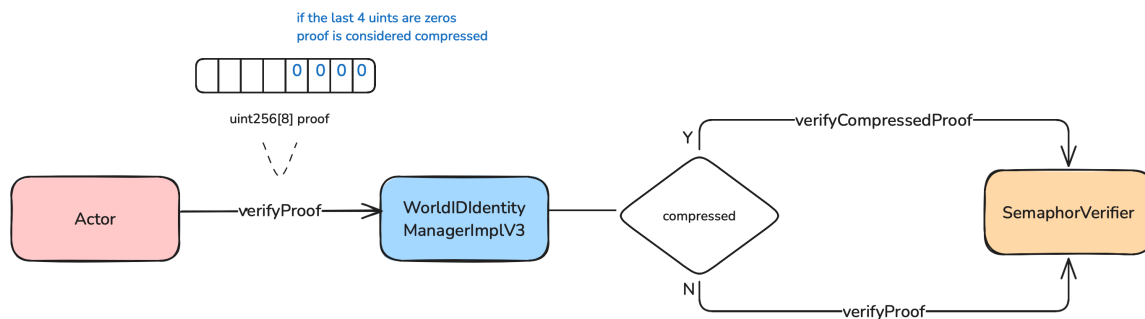

In addition, a new `verifyCompressedProof` function has been introduced, allowing to directly pass in a compressed proof for verification:

```
function verifyCompressedProof(
        uint256 root,
        uint256 signalHash,
        uint256 nullifierHash,
        uint256 externalNullifierHash,
        uint256[4] calldata compressedProof
) public view virtual onlyProxy onlyInitialized
```
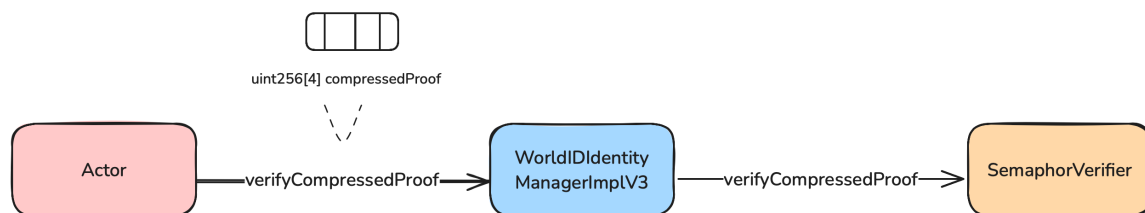

It's important to note that this upgrade does not require reinitializing the contract. As such, the contract's initialized state (`$._initialized`) will remain set to `2`. This poses no issues, as the change does not interfere with any other logic. Additionally, for a future `V4` upgrade, simply adding `reinitializer(4)` will be sufficient.

5 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 the likelihood and impact of an issue, the severity can be determined according to the table below.

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

To address issues that do not fit a High/Medium/Low severity, Nethermind 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.

6 Issues

6.1 [Best Practices] Comments/Natspec notes

File(s): `WorldIDIdentityManagerImplV3.sol`

Description: The following are observations about the comments and Natspec on the `WorldIDIdentityManagerImplV3` file:

```
WorldIDIdentityManagerImplV3
    - Comment on L21 stats that the contract inherits from `WorldIDIdentityManagerImplV1`.
      However, it actually inherits from `WorldIDIdentityManagerImplV2`.

WorldIDIdentityManagerImplV3.verifyProof
    - Comment on L61: "The of the Merkle tree" should be corrected to  "The root of the Merkle tree".

WorldIDIdentityManagerImplV3.verifyCompressedProof
    - Comment on L89: "The of the Merkle tree" should be corrected to "The root of the Merkle tree".
```

Recommendation(s): Consider addressing the comment and Natspec observations made above.

Status: Fixed

Update from the client: Fixed in commits: effe83411f6a15869b36b7446954db6aa4add8a5 and 64bdbb42609f47d2a4f721790c38cec50537dd7a

7 Documentation Evaluation

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

- Technical whitepaper: A technical whitepaper is a comprehensive document describing the smart contract's design and technical details. It includes information about the purpose of the contract, its architecture, its components, and how they interact with each other;

- User manual: A user manual is a document that provides information about how to use the smart contract. It includes step-by-step instructions on how to perform various tasks and explains the different features and functionalities of the contract;

- Code documentation: Code documentation is a document that provides details about the code of the smart contract. It includes information about the functions, variables, and classes used in the code, as well as explanations of how they work;

- API documentation: API documentation is a document that provides information about the API (Application Programming Interface) of the smart contract. It includes details about the methods, parameters, and responses that can be used to interact with the contract;

- Testing documentation: Testing documentation is a document that provides information about how the smart contract was tested. It includes details about the test cases that were used, the results of the tests, and any issues that were identified during testing;

- Audit documentation: Audit documentation includes reports, notes, and other materials related to the security audit of the smart contract. This type of documentation is critical in ensuring that the smart contract is secure and free from vulnerabilities.

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

Remarks about WorldID identity manager implementation upgrade documentation

The documentation for the World identity manager implementation upgrade is contained in the provided PR description in addition to the README file in the project's GitHub repository.

The PR description is thorough and informative. It includes a clear breakdown of the changes made, the impact on the existing codebase, and specific actions required for deployment.

The README file delivers an in-depth explanation of the WorldID Semaphore Contracts, clearly outlining the key components and how they interact. It also includes a section dedicated to explaining the development dependencies, as well as step-by-step instructions on how to build and test the code.

The added functions are thoroughly documented. Each function includes detailed comments that explain the logic behind its implementation and the significance of each parameter.

---

8 Slither

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

9 AuditAgent

All the relevant issues raised by the AuditAgent have been incorporated into this report. The AuditAgent is an AI-powered smart contract auditing tool that analyses code, detects vulnerabilities, and provides actionable fixes. It accelerates the security analysis process, complementing human expertise with advanced AI models to deliver efficient and comprehensive smart contract audits. Available at https://app.auditagent.nethermind.io.

10 Test Suite Evaluation

The test suite for the World Identity Manager contracts is comprehensive, covering both public and internal functions. For the audited changes, new tests have been added to upgrade the implementation to version 3 (v3), as well as to cover the flow of validating compressed proofs.

```
Ran 4 tests for src/test/identity-manager/WorldIDIdentityManagerCalculation.t.sol:WorldIDIdentityManagerCalculation
[PASS] testCalculateIdentityDeletionInputHashFromParametersOnKnownInput() (gas: 30011)
[PASS] testCalculateIdentityRegistrationInputHashFromParametersOnKnownInput() (gas: 32564)
[PASS] testCannotCalculateIdentityDeletionInputHashIfNotViaProxy() (gas: 15709)
[PASS] testCannotCalculateIdentityRegistrationInputHashIfNotViaProxy() (gas: 17907)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 17.97ms (2.54ms CPU time)


Ran 2 tests for src/test/identity-manager/WorldIDIdentityManagerConstruction.t.sol:WorldIDIdentityManagerConstruction
[PASS] testCanConstructIdentityManagerWithDelegate() (gas: 5222742)
[PASS] testCanConstructIdentityManagerWithNoDelegate() (gas: 97700)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 2.32ms (653.15µs CPU time)


Ran 16 tests for src/test/identity-manager/WorldIDIdentityManagerUninit.t.sol:WorldIDIdentityManagerUninit
[PASS] testShouldNotCallCalculateIdentityRegistrationInputHash() (gas: 2778279)
[PASS] testShouldNotCallDeleteIdentitiesWhileUninit() (gas: 2791947)
[PASS] testShouldNotCallGetDeleteIdentitiesVerifierLookupTableAddressWhileUninit() (gas: 2768320)
[PASS] testShouldNotCallGetRegisterIdentitiesVerifierLookupTableAddressWhileUninit() (gas: 2768363)
[PASS] testShouldNotCallGetRootHistoryExpiryWhileUninit() (gas: 2768341)
[PASS] testShouldNotCallGetSemaphoreVerifierAddressWhileUninit() (gas: 2768273)
[PASS] testShouldNotCallIdentityOperatorWhileUninit() (gas: 2768339)
[PASS] testShouldNotCallLatestRootWhileUninit() (gas: 2768317)
[PASS] testShouldNotCallQueryRootWhileUninit() (gas: 2768643)
[PASS] testShouldNotCallRegisterIdentitiesWhileUninit() (gas: 2796584)
[PASS] testShouldNotCallRequireValidRootWhileUninit() (gas: 2768578)
[PASS] testShouldNotCallSetDeleteIdentitiesVerifierLookupTableWhileUninit() (gas: 4020410)
[PASS] testShouldNotCallSetIdentityOperatorWhileUninit(address) (runs: 256, : 2768731, ~: 2768731)
[PASS] testShouldNotCallSetRegisterIdentitiesVerifierLookupTableWhileUninit() (gas: 4020432)
[PASS] testShouldNotCallSetRootHistoryExpiryWhileUninit() (gas: 2768503)
[PASS] testShouldNotCallSetSemaphoreVerifierWhileUninit() (gas: 3846052)
Suite result: ok. 16 passed; 0 failed; 0 skipped; finished in 93.71ms (78.31ms CPU time)


Ran 5 tests for
→ src/test/identity-manager/WorldIDIdentityManagerInitialization.t.sol:WorldIDIdentityManagerInitialization
[PASS] testCannotInitializeTheDelegate() (gas: 2207027)
[PASS] testCannotPassUnsupportedTreeDepth() (gas: 2787222)
[PASS] testInitialisation() (gas: 5228643)
[PASS] testInitialisation2() (gas: 3017428)
[PASS] testInitializationOnlyOnce() (gas: 106533)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 3.87ms (1.96ms CPU time)


Ran 4 tests for src/test/identity-manager/WorldIDIdentityManagerUpgrade.t.sol:WorldIDIdentityManagerUpdate
[PASS] testCanUpgradeImplementationWithCall() (gas: 2296969)
[PASS] testCanUpgradeImplementationWithoutCall() (gas: 2283269)
[PASS] testCannotUpgradeUnlessManager(address) (runs: 256, : 2308281, ~: 2308281)
[PASS] testCannotUpgradeWithoutProxy() (gas: 2282422)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 145.08ms (129.85ms CPU time)


Ran 11 tests for
→ src/test/identity-manager/WorldIDIdentityManagerOwnershipManagement.t.sol:WorldIDIdentityManagerOwnershipManagement
[PASS] testCanSetIdentityOperatorAsOwner(address) (runs: 256, : 60332, ~: 60332)
[PASS] testCannotAcceptOwnershipAsNonPendingOwner(address,address) (runs: 256, : 81561, ~: 81561)
[PASS] testCannotGetIdentityOperatorWithoutProxy() (gas: 8620)
[PASS] testCannotRenounceOwnershipAsOwner() (gas: 14243)
[PASS] testCannotRenounceOwnershipIfNotOwner(address) (runs: 256, : 48745, ~: 48745)
[PASS] testCannotSetIdentityOperatorAsNonOwner(address,address) (runs: 256, : 51547, ~: 51547)
[PASS] testCannotSetIdentityOperatorWithoutProxy(address) (runs: 256, : 8951, ~: 8951)
[PASS] testCannotTransferOwnerIfNotOwner(address,address) (runs: 256, : 49131, ~: 49131)
[PASS] testHasIdentityOperator() (gas: 26421)
[PASS] testHasOwner() (gas: 21976)
[PASS] testTransferOwner(address) (runs: 256, : 65411, ~: 65411)
Suite result: ok. 11 passed; 0 failed; 0 skipped; finished in 675.21ms (659.83ms CPU time)
```

```
Ran 21 tests for
→ src/test/identity-manager/WorldIDIdentityManagerGettersSetters.t.sol:WorldIDIdentityManagerGettersSetters
[PASS] testCanGetDeleteIdentitiesVerifierLookupTableAddress() (gas: 26423)
[PASS] testCanGetRegisterIdentitiesVerifierLookupTableAddress() (gas: 26520)
[PASS] testCanGetRootHistoryExpiry() (gas: 24394)
[PASS] testCanGetSemaphoreVerifierAddress() (gas: 15260)
[PASS] testCanSetDeleteIdentitiesVerifierLookupTable() (gas: 1292451)
[PASS] testCanSetRegisterIdentitiesVerifierLookupTable() (gas: 1292339)
[PASS] testCanSetRootHistoryExpiry(uint256) (runs: 256, : 37827, ~: 37827)
[PASS] testCanSetSemaphoreVerifier() (gas: 1118385)
[PASS] testCannotGetDeleteIdentitiesVerifierLookupTableAddressUnlessViaProxy() (gas: 8642)
[PASS] testCannotGetRegisterIdentitiesVerifierLookupTableAddressUnlessViaProxy() (gas: 8730)
[PASS] testCannotGetRootHistoryExpiryUnlessViaProxy() (gas: 8658)
[PASS] testCannotGetSemaphoreVerifierAddressUnlessViaProxy() (gas: 8642)
[PASS] testCannotSetDeleteIdentitiesVerifierLookupTableUnlessOwner(address) (runs: 256, : 1299148, ~: 1299148)
[PASS] testCannotSetDeleteIdentitiesVerifierLookupTableUnlessViaProxy() (gas: 1260825)
[PASS] testCannotSetRegisterIdentitiesVerifierLookupTableUnlessOwner(address) (runs: 256, : 1299070, ~: 1299070)
[PASS] testCannotSetRegisterIdentitiesVerifierLookupTableUnlessViaProxy() (gas: 1260694)
[PASS] testCannotSetRootHistoryExpiryToZero() (gas: 43660)
[PASS] testCannotSetRootHistoryExpiryUnlessOwner(address) (runs: 256, : 1125127, ~: 1125127)
[PASS] testCannotSetRootHistoryExpiryUnlessViaProxy(uint256) (runs: 256, : 9075, ~: 9075)
[PASS] testCannotSetSemaphoreVerifierAddressUnlessViaProxy() (gas: 1086805)
[PASS] testCannotSetSemaphoreVerifierUnlessOwner(address) (runs: 256, : 1125172, ~: 1125172)
Suite result: ok. 21 passed; 0 failed; 0 skipped; finished in 603.83ms (601.78ms CPU time)

Ran 9 tests for
→ src/test/identity-manager/WorldIDIdentityManagerIdentityDeletion.t.sol:WorldIDIdentityManagerIdentityDeletion
[PASS] testCannotDeleteIdentitiesAsNonIdentityOperator(address) (runs: 256, : 52768, ~: 52768)
[PASS] testCannotDeleteIdentitiesIfPostRootIncorrect(uint256) (runs: 256, : 7731978, ~: 7731978)
[PASS] testCannotDeleteIdentitiesWithIncorrectInputs(uint128[8],uint128,uint128) (runs: 256, : 1008629249, ~:
→ 1008629187)
[PASS] testCannotDeleteIdentitiesWithInvalidBatchSize(uint128[8],uint128,bytes,uint128) (runs: 256, : 6504101, ~:
→ 6504232)
[PASS] testCannotDeleteIdentitiesWithOutdatedRoot(uint256,uint256) (runs: 256, : 5286871, ~: 5287105)
[PASS] testCannotDelteIdentitiesIfNotViaProxy() (gas: 36680)
[PASS] testDeleteIdentitiesSelectsCorrectVerifier(uint128[8],uint128,uint128) (runs: 256, : 6813836, ~: 6813836)
[PASS] testDeleteIdentitiesWithCorrectInputs(uint128[8],uint128,bytes,uint128,address) (runs: 256, : 6539901, ~:
→ 6540188)
[PASS] testDeleteIdentitiesWithCorrectInputsFromKnown() (gas: 7791817)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 4.60s (6.08s CPU time)

Ran 4 tests for
→ src/test/identity-manager/WorldIDIdentityManagerSemaphoreVerification.t.sol:WorldIDIdentityManagerSemaphoreVerification
[PASS] testOptimizedProofVerificationWithCorrectInputs(uint256[8]) (runs: 256, : 6548491, ~: 6548491)
[PASS] testProofVerificationWithCompressedProof(uint8,uint256,uint256,uint256,uint256[4]) (runs: 256, : 8256047, ~:
→ 8256047)
[PASS] testProofVerificationWithCorrectInputs(uint8,uint256,uint256,uint256,uint256[8]) (runs: 256, : 5390271, ~:
→ 5390271)
[PASS] testProofVerificationWithInorrectProof(uint8,uint256,uint256,uint256,uint256[8]) (runs: 256, : 5390443, ~:
→ 5390443)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 4.71s (6.74s CPU time)

Ran 8 tests for src/test/identity-manager/WorldIDIdentityManagerDataQuery.t.sol:WorldIDIdentityManagerDataQuery
[PASS] testCanGetLatestRoot(uint256) (runs: 256, : 5249781, ~: 5250170)
[PASS] testCanGetTreeDepth(uint8) (runs: 256, : 5253233, ~: 5253233)
[PASS] testCannotGetLatestRootIfNotViaProxy() (gas: 8568)
[PASS] testCannotQueryRootIfNotViaProxy() (gas: 12024)
[PASS] testQueryCurrentRoot(uint128) (runs: 256, : 5268383, ~: 5268617)
[PASS] testQueryExpiredRoot(uint128[8],uint32,uint128,uint128,uint128[]) (runs: 256, : 6686452, ~: 6669088)
[PASS] testQueryInvalidRoot(uint256) (runs: 256, : 53034, ~: 53034)
[PASS] testQueryOlderRoot(uint128[8],uint32,uint128,uint128,uint128[]) (runs: 256, : 6682382, ~: 6677890)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 5.49s (3.11s CPU time)

Ran 12 tests for
→ src/test/identity-manager/WorldIDIdentityManagerIdentityRegistration.t.sol:WorldIDIdentityManagerIdentityRegistration
[PASS] testCannotRegisterIdentitiesAsNonIdentityOperator(address) (runs: 256, : 57334, ~: 57334)
[PASS] testCannotRegisterIdentitiesIfIdentitiesIncorrect(uint256) (runs: 256, : 7761383, ~: 7761383)
[PASS] testCannotRegisterIdentitiesIfNotViaProxy() (gas: 41017)
[PASS] testCannotRegisterIdentitiesIfPostRootIncorrect(uint256) (runs: 256, : 5536451, ~: 5536451)
[PASS] testCannotRegisterIdentitiesIfStartIndexIncorrect(uint32) (runs: 256, : 7737090, ~: 7737090)
```

```
[PASS] testCannotRegisterIdentitiesWithIncorrectInputs(uint128[8],uint32,uint128,uint128) (runs: 256, : 7739330, ~:
↪ 7739875)
[PASS] testCannotRegisterIdentitiesWithInvalidBatchSize(uint128[8],uint32,uint128,uint128,uint128[]) (runs: 256, :
↪ 6626277, ~: 6606605)
[PASS] testCannotRegisterIdentitiesWithOutdatedRoot(uint256,uint256) (runs: 256, : 5291683, ~: 5291839)
[PASS] testRegisterIdentitiesSelectsCorrectVerifier(uint128[8],uint32,uint128,uint128,uint128[]) (runs: 256, : 6961632,
↪ ~: 6948523)
[PASS] testRegisterIdentitiesWithCorrectInputs(uint128[8],uint32,uint128,uint128,uint128[],address) (runs: 256, :
↪ 6668309, ~: 6653440)
[PASS] testRegisterIdentitiesWithCorrectInputsFromKnown() (gas: 7796974)
[PASS] testRegisterIdentitiesWithRunsOfZeroes(uint8,uint8) (runs: 256, : 6576423, ~: 6580141)
Suite result: ok. 12 passed; 0 failed; 0 skipped; finished in 5.40s (19.09s CPU time)
```

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 procehttps://www.overleaf.com/project/65c0e737f41a29601bda5c48ss, 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.