
Security Review Report
NM-0395 Safe Anonymous Mail Module (SAMM)
Oxorio



NETHERMIND
SECURITY

(Mar 20, 2025)

Contents

1	Executive Summary	2
2	Audited Files	3
3	Summary of Issues	3
4	System Overview	4
5	Risk Rating Methodology	5
6	Issues	6
6.1	[Critical] Proof checking reverts for multiple proofs	6
6.2	[Low] Lack of specifying operation type	6
6.3	[Low] No check for minimum value of a secret	7
6.4	[Low] Domain is part of the Proof object but is not proved in the circuit	7
6.5	[Info] Hashing Part of the Public Key for 2048 Bits	7
6.6	[Info] Not all scenarios are taken into account for the From field	8
6.7	[Info] Redundant constraint in <code>lib.verify_samm_logic</code> function	9
6.8	[Info] Redundant constraints	10
6.9	[Info] <code>MAX_EMAIL_FIELD_LENGTH</code> is greater than <code>MAX_EMAIL_HEADER_LENGTH</code>	11
6.10	[Best Practice] Provide information about non-standard padding	11
7	Documentation Evaluation	12
8	Test Suite Evaluation	13
8.1	Compilation Output	13
8.1.1	Circuit Compilation Output	13
8.1.2	Smart Contract Compilation Output	14
8.2	Tests Output	17
8.2.1	Circuit Test Output	17
8.2.2	Smart Contract Output	17
9	About Nethermind	19

1 Executive Summary

This document outlines the security review conducted by [Nethermind Security](#) for Noir circuits and smart contracts of [Safe Anonymous Mail Module \(SAMM\)](#). The Safe Anonymous Mail Module (SAMM) is a specialized application integrated into Safe multisig wallets to enhance transaction anonymity and streamline management. Designed to work as a custom app within the Safe Wallet interface, SAMM ensures anonymous interaction for its participants using Noir Language. The module distinguishes user roles into Members and Safe Owners, providing role-specific interfaces and functionalities. Members access a web-based portal to generate transaction data and track statuses, while Safe Owners manage SAMM modules within the Safe Wallet (configuring settings and memberships). Safe Owners maintain administrative control but cannot approve transactions, ensuring privacy and security while streamlining multisig transaction management.

The code review covers both Noir circuits, with 487 lines of code, and Solidity smart contracts, comprising 479 lines of code. The Oxorio team has provided extensive resources to detail SAMM's functionality, including code comments, in-depth documentation, and collaborative meetings. Their documentation is particularly thorough, offering: i) clear and detailed diagrams that illustrate the entire flow at various levels of granularity; ii) precise definitions for all attributes used within the circuits and smart contracts; iii) comprehensive explanations of off-chain data storage methods; and iv) guidance on running tests, among other essential details.

The audit was performed using: (a) manual analysis of the codebase and (b) writing test cases. **Along this document, we report** 10 points of attention, where one is classified as *Critical*, two are classified as *Low*, and seven are classified as *Informational* or *Best Practice*. The issues are summarized in Fig. 1.

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

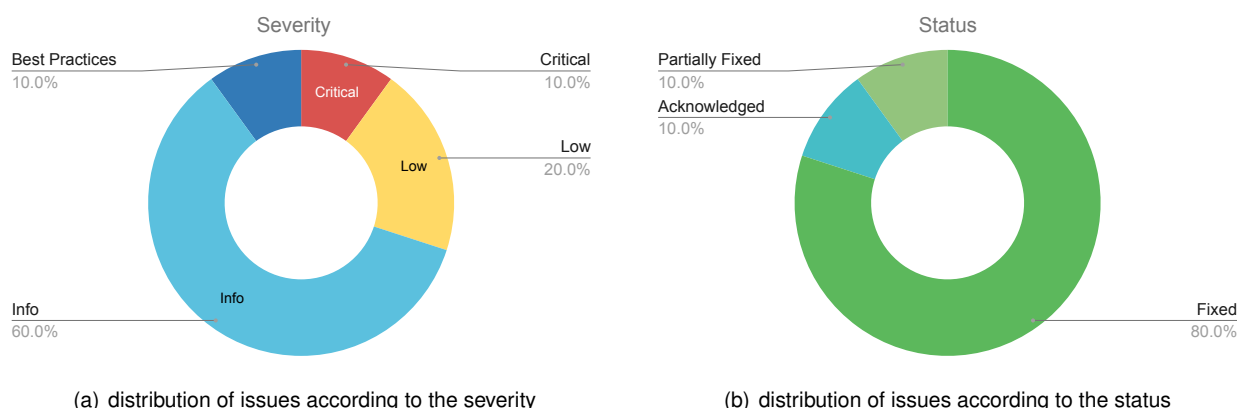


Fig 1: (a) Distribution of issues: Critical (1), High (0), Medium (0), Low (2), Undetermined (0), Informational (6), Best Practices (1). (b) Distribution of status: Fixed (8), Acknowledged (1), Mitigated (0), Unresolved (0), Partially Fixed (1)

Summary of the Audit

Audit Type	Security Review
Initial Report	Jan 8, 2025
Final Report	Mar 20, 2025
Methods	Manual Review, Tests
Repository	samm-circuits
Repository	samm-contracts
Commit Hash - Circuits	240b94a36effc0eadc8620766ab056d05e128483
Commit Hash - Smart contracts	a7df4a8d5a2e30f71201ab491cc145d9c00ab87f
Final Commit Hash - Circuits	6eb3eb90168df39edf43762f63b195b66afe7939
Final Commit Hash - Smart contracts	f57a64a69488725e5517b19b957c64ea5ff9b6d7
Documentation	Communication, Documentation, and NatSpec
Documentation Assessment	High
Test Suite Assessment	Medium

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	Samm.sol	246	155	63.0%	63	464
2	Safe/common/Singleton.sol	4	8	200.0%	1	13
3	Safe/interfaces/ISafe.sol	26	1	3.8%	3	30
4	Safe/interfaces/MinimalSafeModuleManager.sol	16	19	118.8%	6	41
5	libraries/PubSignalsConstructor.sol	40	22	55.0%	6	68
6	libraries/Enum.sol	7	5	71.4%	1	13
7	utils/DKIMRegistry.sol	36	15	41.7%	15	66
8	interfaces/ISAMM.sol	43	18	41.9%	6	67
9	interfaces/ISAMMEvents.sol	17	18	105.9%	1	36
10	interfaces/IDKIMRegistry.sol	4	1	25.0%	1	6
11	interfaces/ISAMMErrors.sol	20	18	90.0%	1	39
12	interfaces/ISAMMGetters.sol	16	18	112.5%	6	40
13	interfaces/IERC165.sol	4	10	250.0%	1	15
	Total	479	308	64.3%	105	892

	Circuit	LoC	Comments	Ratio	Blank	Total
1	builds/samm_2048/src/main.nr	121	9	7.4%	16	146
2	builds/samm_1024/src/main.nr	53	6	11.3%	9	68
3	lib/src/lib.nr	136	12	8.8%	22	170
4	lib/src/merkle_tree.nr	30	3	10.0%	3	36
5	lib/src/utils.nr	147	35	23.8%	22	204
	Total	487	65	13.3%	72	624

3 Summary of Issues

	Finding	Severity	Update
1	Proof checking reverts for multiple proofs	Critical	Fixed
2	Lack of specifying operation type	Low	Fixed
3	No check for minimum value of a secret	Low	Fixed
4	Domain is part of the Proof object but is not proved in the circuit	Info	Fixed
5	Hashing Part of the Public Key for 2048 Bits	Info	Fixed
6	Not all scenarios are taken into account for the From field	Info	Fixed
7	Redundant constraint in lib.verify_samm_logic function	Info	Fixed
8	Redundant constraints	Info	Partially Fixed
9	MAX_EMAIL_FIELD_LENGTH is greater than MAX_EMAIL_HEADER_LENGTH	Info	Fixed
10	Provide information about non-standard padding	Best Practices	Acknowledged

4 System Overview

The SAMM allows for anonymous voting via email. The emails are received by the Relayer. The email may contain the proposition of a transaction or can express voting for processing execution of the already proposed transaction. The Relayer processes the messages and stores all the votes. After the threshold is reached, the ZK proof is generated, and it is posted on-chain to a SAMM contract. After the proof is successfully verified, the transaction is executed using the Safe wallet. Below, we present the general flow.

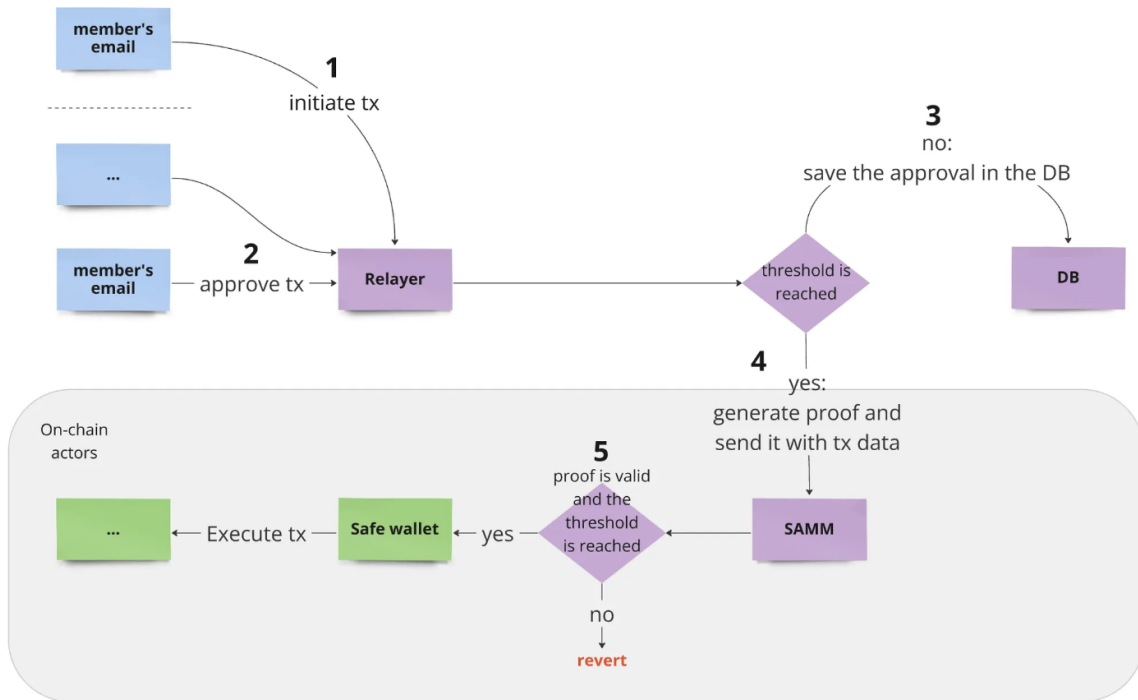


Fig. 2: The flow of the voting process is finalized with transaction execution. The grey box represents the on-chain part of the protocol. The diagram borrowed from the [SAMM documentation](#).

The security review focused mostly on the circuits written in the Noir language together with the SAMM contract for the context. The circuits are on the trusted Relayer. The circuit performs a number of actions to ensure correctness of the email. First, it checks the length of the inputs. Then, it ensures that the provided email address is in the Merkle Tree root, which ensures that the sender is actually in the allowed senders. The leaf that is checked against the tree is not just the email, but it's actually $\text{hash}(\text{email} || \text{secret})$. Next, the circuit calculates commitment, which is $\text{hash}(\text{leaf} || \text{message hash})$. The message hash is the unique identifier of the transaction created from the hash of transaction parameters: $\text{hash}(\text{recipient address} || \text{value of transferred tokens} || \text{calldataHash, operation type, nonce, deadline, address of the SAMM contract, block.chainid})$. At the end the circuit checks if the header is correctly signed by the sender.

5 Risk Rating Methodology

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

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

- a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;
- b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;
- c) **Low**: The issue is very complex and requires very specific conditions to be met.

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

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

- a) **High**: The issue can cause significant damage such as loss of funds or the protocol entering an unrecoverable state;
- b) **Medium**: The issue can cause moderate damage such as impacts that only affect a small group of users or only a particular part of the protocol;
- c) **Low**: The issue can cause little to no damage such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

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

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

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

- a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to formally pass to the client;
- b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;
- c) **Undetermined** findings are used when we cannot predict the impact or likelihood of the issue.

6 Issues

6.1 [Critical] Proof checking reverts for multiple proofs

File(s): SAMM.sol

Description: The function `_checkNProofs` validates the provided proofs against the generated `pubSignals`. Before proceeding with verification, it confirms the uniqueness of each proof's commit. A commit consists of the `hash(userEmail, msgHash)`, which ensures that each commit is distinct.

```

1  function _checkNProofs(Proof[] calldata proofs, bytes32[] memory pubSignals) private {
2      uint256 proofsLength = proofs.length;
3
4      for (uint256 i; i < proofsLength; i++) {
5          Proof memory currentProof = proofs[i];
6
7          // ...
8
9          for (uint256 j; j < i; j++) {
10             // @audit this condition will always be true and this function
11             // always reverts
12             if (proofs[i].commit == currentProof.commit) {
13                 revert SAMM__commitAlreadyUsed(i);
14             }
15         }
16
17         // ...
18         if (currentProof.is2048sig) {
19             result = VERIFIER2048.verify({proof: currentProof.proof, publicInputs: pubSignals});
20         } else {
21             result = VERIFIER1024.verify({proof: currentProof.proof, publicInputs: pubSignals});
22         }
23
24         // ...
25     }

```

However, the condition that compares `currentProof.commit` with all previous commits is always true when `proofs.length` is greater than 1, causing the function to revert.

Recommendation: change the index `i` by `j` inside the loop for the condition and the error message.

Status: Fixed

Update from the client:

6.2 [Low] Lack of specifying operation type

File(s): SAMM.sol

Description: The SAMM contract checks for whitelisted transactions by storing the function signature and target address in the `isTxAllowed` mapping. However, the operation type, which can be either `call` or `delegatecall` is not defined in the mapping. This potentially may result in incorrect operation by mistake or even malicious action.

Recommendation(s): Consider adding operation type to the `isTxAllowed` mapping.

Status: Fixed

Update from the client:

6.3 [Low] No check for minimum value of a secret

File(s): [lib.nr](#)

Description: The secret that is passed as a private input allows for hiding the user's email in the commitment, which allows for anonymous voting. However, if the secret is short in some scenarios, it would be possible to guess the secret. Consider the following scenario:

- user A knows user B's email address. This may happen because of leakage or in a setup where emails are known to the public since only the voting is anonymized;
- voting on some transactions happened, so there are commitments of participants published on-chain;
- user A wants to know if user B voted on this transaction;
- since the message hash is known to everyone, and the email address of user B is known to user A, user A may try to guess the secret by brute-force calculating the commitment hash(`msg_hash || hash(address || secret)`) and comparing it with commitment posted on-chain (max: 2^8);
- if secret is short, guessing the commitment is possible;

This may lead to de-anonymising the voting which has a high impact. However, it is based on assumption of known email address, which in most use cases would not happen.

Recommendation(s): Consider creating a minimum length of a secret to ensure that the secret is strong.

Status: Fixed

Update from the client:

6.4 [Low] Domain is part of the Proof object but is not proved in the circuit

File(s): [SAMM.sol](#), [lib.nr](#)

Description: The domain is checked on-chain in the SAMM contract to make sure that the hash of a public key was registered at the correct domain. However, the domain is not included in the circuit. This could potentially result in creating proof for an email with an incorrect/malicious domain. Later, during the on-chain call, the domain provided in Proof can be correct since anyone can call execution, which results in passing checks in the SAMM contract.

Recommendation(s): Consider including the domain in the circuit as a public input.

Status: Fixed

Update from the client:

6.5 [Info] Hashing Part of the Public Key for 2048 Bits

File: [samm_2048/src/main.nr](#)

Description: The proof includes the hash of the public key as a public output. This allows verifiers to confirm the correctness of the provided public key without requiring the full key. though the key is internally represented as 18 limbs (as defined by `KEY_LIMBS_2048`), only 16 of those limbs are hashed due to a limitation in Noir's Poseidon hashing function, which can process a maximum of 16 inputs at a time:

```
1 let mut pubkey_16 = [0; 16];
2 for i in 0..16 {
3     pubkey_16[i] = pubkey.modulus[i];
4 }
5 let pubkeyHash = std::hash::poseidon::bn254::hash_16(pubkey_16);
```

However, even if this method of hashing the key remains secure, there is no reason to lower the security of the scheme.

Recommendation: One better way to hash the key that makes no compromises would be chain hashing. This ensures that no information is omitted or treated less securely.

```
1 hash[hash(part_0,...,part_15), hash(part16,..., part17)]
```

It is also important to note that when the proof's output is used for verification, it is crucial to replicate the exact same hashing procedure.

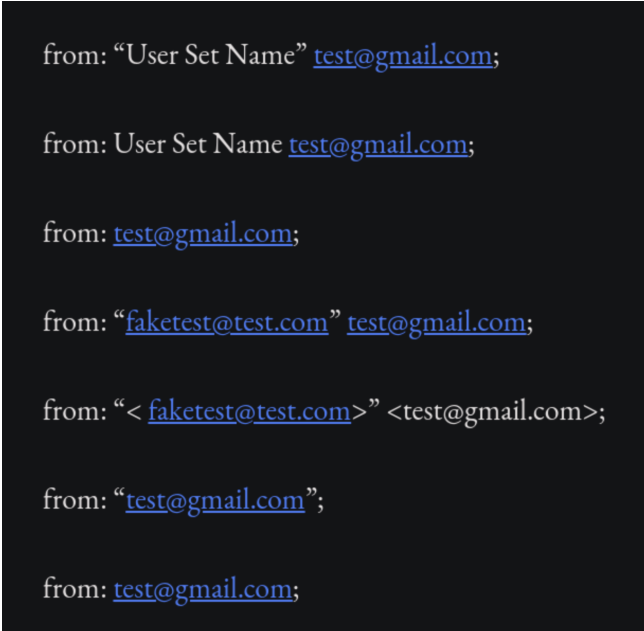
Status: Fixed

Update from the client:

6.6 [Info] Not all scenarios are taken into account for the From field

File(s): [utils.nr](#)

Description: The from field right now can cover the scenarios when it is like this Dry 914 dry.914@yandex.com or like this ad@oxor.io. However, looking at the [IMF RFC](#), it can have more ways to appear.



```
from: "User Set Name" test@gmail.com;  
  
from: User Set Name test@gmail.com;  
  
from: test@gmail.com;  
  
from: "<faketest@test.com>" test@gmail.com;  
  
from: "< faketest@test.com>" <test@gmail.com>;  
  
from: "test@gmail.com";  
  
from: test@gmail.com;
```

Fig. 3: This figure was extracted from [Why you need a regex for the from: field](#).

Recommendation(s): Either cover all scenarios or document the only ones covered.

Status: Fixed

Update from the client:

Update from Nethermind Security: The covered scenarios were set in the [documentation](#).

6.7 [Info] Redundant constraint in lib.verify_samm_logic function

File(s): lib.nr

Description: The function `verify_samm_logic` accepts several parameters, including `relayer` and `relayer_seq`. The length of `relayer` is constrained by the constant `MAX_EMAIL_ADDRESS_LENGTH`. The sequence `relayer_seq.length` represents the length of `relayer`.

```

1 pub fn verify_samm_logic(
2   header: BoundedVec<u8, MAX_EMAIL_HEADER_LENGTH>,
3   root: Field,
4   path_indices: [Field; LEVELS],
5   path_elements: [Field; LEVELS],
6   secret: Field,
7   padded_member: [u8; MAX_EMAIL_ADDRESS_LENGTH],
8   from_seq: Sequence,
9   member_seq: Sequence,
10  relayer: BoundedVec<u8, MAX_EMAIL_ADDRESS_LENGTH>,
11  to_seq: Sequence,
12  relayer_seq: Sequence,
13  msg_hash: [u8; MSG_HASH_LENGTH]
14 ) -> Field {
15   // ...

```

In the snippet below, the first constraint verifies whether `relayer_seq.length <= MAX_EMAIL_ADDRESS_LENGTH`, while the second ensures `relayer.len == relayer_seq.length`. However, the first constraint is redundant because the second guarantees that both `relayer` length and `relayer_seq.length` are equal, and `relayer` size is bounded by `MAX_EMAIL_ADDRESS_LENGTH`. The first redundant assertion introduces an unnecessarily constraint, increasing the circuit size unnecessarily.

```

1 // ...
2 // @audit This check is redundant since relayer.len == relayer_seq.length,
3 //       and relayer is constrained by MAX_EMAIL_ADDRESS_LENGTH.
4 assert(relayer_seq.length <= MAX_EMAIL_ADDRESS_LENGTH);
5 assert(relayer.len == relayer_seq.length);
6 // ...

```

Recommendation(s): Remove the constraint below:

```
-assert(relayer_seq.length <= MAX_EMAIL_ADDRESS_LENGTH);
```

Status: Fixed

Update from the client:

6.8 [Info] Redundant constraints

File(s): `utils.nr`, `lib.nr`

Description: The circuits contain some constraints that are potentially duplicates, which increases the circuit size. We list them below:

- The function `verify_samm_logic(...)` checks the length of the header input parameter, but the input is defined to be a `BoundedVec<u8, MAX_EMAIL_HEADER_LENGTH>` which already checks its length;
- The functions `check_to_field` and `check_from_field` check the `to` and `from` fields in the email header. As presented below, the `check_to_field` function calls a third-party function named `constrain_header_field_detect_last_angle_bracket`;

```

1 pub fn check_to_field(
2   header: BoundedVec<u8, MAX_EMAIL_HEADER_LENGTH>,
3   padded_to: [u8; MAX_EMAIL_ADDRESS_LENGTH],
4   to_seq: Sequence,
5   email_seq: Sequence
6 ) {
7   let to = comptime {
8     "to".as_bytes()
9   };
10
11   // check field is uninterrupted and matches the expected field name
12   let last_angle_bracket = constrain_header_field_detect_last_angle_bracket:<
13     MAX_EMAIL_HEADER_LENGTH, MAX_EMAIL_FIELD_LENGTH, TO_FIELD_LENGTH>(header, to_seq, to);
14   // ...

```

The function `constrain_header_field_detect_last_angle_bracket` is listed below:

```

1 pub fn constrain_header_field_detect_last_angle_bracket<let MAX_HEADER_LENGTH: u32, let MAX_HEADER_FIELD_LENGTH: u32,
2   ↳ let HEADER_FIELD_NAME_LENGTH: u32>{
3   header: BoundedVec<u8, MAX_HEADER_LENGTH>,
4   header_field_sequence: Sequence,
5   header_field_name: [u8; HEADER_FIELD_NAME_LENGTH],
6   ↳ u32 {
7   // check that the sequence is within bounds
8   assert(
9     header_field_sequence.index + header_field_sequence.length <= header.len(),
10    "Header field out of bounds",
11  );
12  // check the range of the sequence is within the header (so we can use get_unchecked)
13  let end_index = header_field_sequence.index + header_field_sequence.length;
14  assert(end_index <= header.len(), "Header field out of bounds of header");
15  // ...

```

As presented above, the checkings are redundant, and each check creates a constraint that increases the circuit size.

Recommendation: Clone the third-party function and remove one of the duplicated asserts.

Status: Partially fixed

Update from the client: Partially fixed - removed the redundant constraint from our code, but we do not want to clone the third-party function and modify it in this iteration.

6.9 [Info] MAX_EMAIL_FIELD_LENGTH is greater than MAX_EMAIL_HEADER_LENGTH

File(s): [lib.nr](#)

Description: The function `verify_samm_logic` ensures that `from_seq.length` and `to_seq.length` do not exceed `MAX_EMAIL_FIELD_LENGTH`. As shown in the code below, `MAX_EMAIL_HEADER_LENGTH` is set to 1024, while `MAX_EMAIL_FIELD_LENGTH` is 1029. This configuration allows the lengths of the "from" and "to" fields to exceed the header length.

```

1  global MAX_EMAIL_HEADER_LENGTH: u32 = 1024;
2  global MAX_EMAIL_ADDRESS_LENGTH: u32 = 124;
3  // @audit this field is greater than the header itself
4  global MAX_EMAIL_FIELD_LENGTH: u32 = 1029;
5
6  // ...
7
8  pub fn verify_samm_logic(
9      header: BoundedVec<u8, MAX_EMAIL_HEADER_LENGTH>,
10     root: Field,
11     path_indices: [Field; LEVELS],
12     path_elements: [Field; LEVELS],
13     secret: Field,
14     padded_member: [u8; MAX_EMAIL_ADDRESS_LENGTH],
15     from_seq: Sequence,
16     member_seq: Sequence,
17     relayer: BoundedVec<u8, MAX_EMAIL_ADDRESS_LENGTH>,
18     to_seq: Sequence,
19     relayer_seq: Sequence,
20     msg_hash: [u8; MSG_HASH_LENGTH]
21 ) -> Field {
22     assert(header.len <= MAX_EMAIL_HEADER_LENGTH);
23     // @audit Ensure the length of from_seq does not exceed MAX_EMAIL_FIELD_LENGTH that is greater than header length
24     assert(from_seq.length <= MAX_EMAIL_FIELD_LENGTH);
25     assert(member_seq.length <= MAX_EMAIL_ADDRESS_LENGTH);
26     // @audit MAX_EMAIL_FIELD_LENGTH is greater than header length
27     assert(to_seq.length <= MAX_EMAIL_FIELD_LENGTH);
28     // ...
29 }
```

Recommendation(s): Ensure that the lengths in the sequences `to` and `from` fields do not exceed the header length.

Status: Fixed

Update from the client:

6.10 [Best Practice] Provide information about non-standard padding

File(s): [lib.nr](#)

Description: The functions `member_to_leaf(...)` and `compute_commitment(...)` pad the provided input in a non-standard way since zeros are not appended before the data but in between the parts of the data. Below we provide the examples on `member_to_leaf(...)`:

```

1  // standard padding to create a leaf
2  hash(padding || member || secret)
3  // implemented padding to create a leaf
4  hash(0 || part1_member || 0 || part2_member || 0 || part3_member || 0 || part4_member || secret)
```

This should be clearly documented to the operators of the relayer that would create a merkle root, to be sure that this non-standard padding is used for root creation.

Recommendation(s): Consider clearly documenting how the data is padded.

Status: Acknowledged

Update from the client: Acknowledged - Will be added to the documentation for relayer operators (out of scope for this audit).

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. Protocols can use various types of software documentation for their circuits. Some of the most common types include:

Circuit Specifications: A documentation that outlines the purpose, functionality, and constraints of the circuit:

- i) High-level description - What the circuit is designed to achieve, e.g., proving membership in a Merkle tree, verifying a signature, etc. Description about inputs and outputs (public and private signals);
- ii) Mathematical model - Description of the mathematical computations and proofs being implemented. In addition, describe the cryptographic primitives that are used, e.g., Poseidon hash, EdDSA verification.
- iii) Expected behavior - Present a detailed explanation of how the circuit behaves under normal, edge, and adversarial conditions.

Documentation of the Input and Output: The documentation should describe details of the inputs and outputs of the circuit.

- i) Public inputs: List of public inputs and their expected values and ranges. Also, clarify constraints on the public inputs (e.g, non-zero).
- ii) Private inputs: List of private input signals and their roles in the circuit. The documentation should also include validation requirements for private inputs within the circuit.
- iii) Outputs: A description of the outputs, their intended use, their relationship with the inputs.

Performance Metrics: Describe details on the prover and verifier performance such as, time taken to generate proofs, memory usage during proof generation, time taken to verify proofs, etc.

Testing Documentation: Testing documentation is a document that provides information about how the circuits was tested. It includes details about the test cases that were used, coverage metrics showing which parts of the circuit have been tested, and any issues that were identified during testing.

Technical whitepaper: A technical whitepaper is a comprehensive document that describes the design and technical details of the protocol. It includes information about the purpose of the protocol, contracts, its architecture, its components, and how they interact with each other;

These types of documentation are essential for circuit development and maintenance. They help ensure that the circuit is properly designed, implemented, and tested. The documentation provides a reference for developers who need to modify the circuit in the future.

Remarks about Protocol documentation

The Oxorio team has provided comprehensive code comments detailing the inputs for circuits and smart contracts. Additionally, they have delivered high-quality documentation available in [SAMM Documentation](#) and [SAMM Milestone](#). This documentation includes: i) clear and detailed diagrams illustrating the entire flow at various levels of granularity to enhance understanding; ii) definitions for every attribute used within the circuits and smart contracts; iii) explanations of how data is stored off-chain; and iv) guidance on running tests, among other details.

8 Test Suite Evaluation

8.1 Compilation Output

8.1.1 Circuit Compilation Output

The compilation of the Noir circuits generated numerous warnings, mainly stemming from dependencies within the Noir-lang libraries and ZKEmail modules. To streamline the output for inclusion in the report, we have filtered the results to retain only the warnings directly related to the SAMM circuits.

```
% nargo compile
warning: DebugRandomEngine is private and not visible from the current module
  /nargo/github.com/noir-lang/noir_string_searchv0.2.0/src/lib.nr:3:43

3 pub use utils::{conditional_select, lt_f, DebugRandomEngine};
      ----- DebugRandomEngine is private

...

warning: StringBody is private and not visible from the current module
  /Downloads/oxorio-samm-noir/samm-circuits/lib/src/utils.nr:2:26

2 use dep::string_search::{StringBody, SubString, StringBody1024, SubString32, SubString64, SubString128};
      ----- StringBody is private

warning: SubString is private and not visible from the current module
  /Downloads/oxorio-samm-noir/samm-circuits/lib/src/utils.nr:2:38

2 use dep::string_search::{StringBody, SubString, StringBody1024, SubString32, SubString64, SubString128};
      ----- SubString is private

warning: StringBody1024 is private and not visible from the current module
  /Downloads/oxorio-samm-noir/samm-circuits/lib/src/utils.nr:2:49

2 use dep::string_search::{StringBody, SubString, StringBody1024, SubString32, SubString64, SubString128};
      ----- StringBody1024 is private

warning: SubString32 is private and not visible from the current module
  /Downloads/oxorio-samm-noir/samm-circuits/lib/src/utils.nr:2:65

2 use dep::string_search::{StringBody, SubString, StringBody1024, SubString32, SubString64, SubString128};
      ----- SubString32 is private

warning: SubString64 is private and not visible from the current module
  /Downloads/oxorio-samm-noir/samm-circuits/lib/src/utils.nr:2:78

2 use dep::string_search::{StringBody, SubString, StringBody1024, SubString32, SubString64, SubString128};
      ----- SubString64 is private

warning: SubString128 is private and not visible from the current module
  /Downloads/oxorio-samm-noir/samm-circuits/lib/src/utils.nr:2:91

2 use dep::string_search::{StringBody, SubString, StringBody1024, SubString32, SubString64, SubString128};
      ----- SubString128 is private

warning: constrain_header_field_detect_last_angle_bracket is private and not visible from the current module
  /Downloads/oxorio-samm-noir/samm-circuits/lib/src/utils.nr:3:44

3 use dep::zkemail::{headers::email_address::constrain_header_field_detect_last_angle_bracket, Sequence};
      -----
  ↳ constrain_header_field_detect_last_angle_bracket is private
```

```
warning: unused variable i
/Downloads/oxorio-samm-noir/samm-circuits/lib/src/utils.nr:247:9

247   for i in header_brackets.len() .. MAX_EMAIL_HEADER_LENGTH{
      - unused variable

warning: KEY_LIMBS_1024 is private and not visible from the current module
src/main.nr:2:5

2     KEY_LIMBS_1024, dkim:RSAPubkey, Sequence
      ----- KEY_LIMBS_1024 is private

warning: LEVELS is private and not visible from the current module
src/main.nr:5:5

5     LEVELS, MAX_EMAIL_HEADER_LENGTH, MAX_EMAIL_ADDRESS_LENGTH,
      ----- LEVELS is private

warning: MAX_EMAIL_HEADER_LENGTH is private and not visible from the current module
src/main.nr:5:13

5     LEVELS, MAX_EMAIL_HEADER_LENGTH, MAX_EMAIL_ADDRESS_LENGTH,
      ----- MAX_EMAIL_HEADER_LENGTH is private

warning: MAX_EMAIL_ADDRESS_LENGTH is private and not visible from the current module
src/main.nr:5:38

5     LEVELS, MAX_EMAIL_HEADER_LENGTH, MAX_EMAIL_ADDRESS_LENGTH,
      ----- MAX_EMAIL_ADDRESS_LENGTH is private

warning: MSG_HASH_LENGTH is private and not visible from the current module
src/main.nr:6:5

6     MSG_HASH_LENGTH, verify_samm_logic
      ----- MSG_HASH_LENGTH is private
```

8.1.2 Smart Contract Compilation Output

The compilation of the smart contracts generated the following output:

```
sam_contracts% forge build
[] Compiling...
[] Compiling 57 files with 0.8.23
[] Solc 0.8.23 finished in 2.14s
Compiler run successful with warnings:
Warning (5667): Unused function parameter. Remove or comment out the variable name to silence this warning.
--> src/Utils/Verifier1024.sol:432:93:
|
|
432 |     function generateEtaChallenge(Honk.Proof memory proof, bytes32[] calldata publicInputs, uint256
-> publicInputsSize)
|
|
Warning (5667): Unused function parameter. Remove or comment out the variable name to silence this warning.
--> src/Utils/Verifier1024.sol:1169:9:
|
|
1169 |         Transcript memory tp, // I think this is not needed
|
|
Warning (5667): Unused function parameter. Remove or comment out the variable name to silence this warning.
--> src/Utils/Verifier1024.sol:1226:9:
|
|
1226 |         Transcript memory tp, // I think this is not needed
|
|
Warning (5667): Unused function parameter. Remove or comment out the variable name to silence this warning.
--> src/Utils/Verifier2048.sol:432:93:
|
|
432 |     function generateEtaChallenge(Honk.Proof memory proof, bytes32[] calldata publicInputs, uint256
-> publicInputsSize)
|
|
Warning (5667): Unused function parameter. Remove or comment out the variable name to silence this warning.
--> src/Utils/Verifier2048.sol:1169:9:
|
|
1169 |         Transcript memory tp, // I think this is not needed
|
|
Warning (5667): Unused function parameter. Remove or comment out the variable name to silence this warning.
--> src/Utils/Verifier2048.sol:1226:9:
|
|
1226 |         Transcript memory tp, // I think this is not needed
|
|
Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier1024.sol:432:5:
|
|
432 |     function generateEtaChallenge(Honk.Proof memory proof, bytes32[] calldata publicInputs, uint256
-> publicInputsSize)
|
|
^ (Relevant source part starts here and spans across multiple lines).
Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier1024.sol:467:5:
|
|
467 |     function generateBetaAndGammaChalle ... Fr gamma, Fr nextPreviousChallenge)
|
|
^ (Relevant source part starts here and spans across multiple lines).
Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier1024.sol:489:5:
|
|
489 |     function generateAlphaChallenges(Fr ... y alphas, Fr nextPreviousChallenge)
|
|
^ (Relevant source part starts here and spans across multiple lines).
Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier1024.sol:517:5:
|
|
517 |     function generateGateChallenges(Fr ... allenges, Fr nextPreviousChallenge)
|
|
^ (Relevant source part starts here and spans across multiple lines).
Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier1024.sol:527:5:
|
|
527 |     function generateSumcheckChallenges ... allenges, Fr nextPreviousChallenge)
|
|
^ (Relevant source part starts here and spans across multiple lines).
```



```

Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier1024.sol:702:5:
|
702 |     function accumulateLogDerivativeLookupRelation(
|       ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier1024.sol:742:5:
|
742 |     function accumulateDeltaRangeRelation(
|       ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier1024.sol:814:5:
|
814 |     function
|       ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier1024.sol:1310:5:
|
1310 |     function loadVerificationKey() internal view returns (Honk.VerificationKey memory) {
|       ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier1024.sol:1317:5:
|
1317 |     function loadProof(bytes calldata proof) internal view returns (Honk.Proof memory) {
|       ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier1024.sol:1458:5:
|
1458 |     function checkSum(Fr[BATCHED_RELATION_PARTIAL_LENGTH] memory roundUnivariate, Fr roundTarget)
|       ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier2048.sol:432:5:
|
432 |     function generateEtaChallenge(Honk.Proof memory proof, bytes32[] calldata publicInputs, uint256
|       ^ (Relevant source part starts here and spans across multiple lines).
-> publicInputsSize)

Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier2048.sol:467:5:
|
467 |     function generateBetaAndGammaChalle ... Fr gamma, Fr nextPreviousChallenge)
|       ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier2048.sol:489:5:
|
489 |     function generateAlphaChallenges(Fr ... y alphas, Fr nextPreviousChallenge)
|       ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier2048.sol:517:5:
|
517 |     function generateGateChallenges(Fr ... allenges, Fr nextPreviousChallenge)
|       ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier2048.sol:527:5:
|
527 |     function generateSumcheckChallenges ... allenges, Fr nextPreviousChallenge)
|       ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier2048.sol:702:5:
|
702 |     function accumulateLogDerivativeLookupRelation(
|       ^ (Relevant source part starts here and spans across multiple lines).

```

```
Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier2048.sol:742:5:
|
742 |     function accumulateDeltaRangeRelation(
|         ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier2048.sol:814:5:
|
814 |     function
|         ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier2048.sol:1310:5:
|
1310 |     function loadVerificationKey() internal view returns (Honk.VerificationKey memory) {
|         ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier2048.sol:1317:5:
|
1317 |     function loadProof(bytes calldata proof) internal view returns (Honk.Proof memory) {
|         ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
--> src/Utils/Verifier2048.sol:1458:5:
|
1458 |     function checkSum(Fr[BATCHED_RELATION_PARTIAL_LENGTH] memory roundUnivariate, Fr roundTarget)
|         ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to view
--> src/SAMM.sol:430:5:
|
430 |     function _checkNProofs(Proof[] calldata proofs, bytes32[] memory pubSignals) private {
|         ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to view
--> test/unit/GuardSetup.t.sol:19:5:
|
19 |     function test_safeIsInitializedCorrectly() external {
|         ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to view
--> test/unit/SAMSetup.t.sol:21:5:
|
21 |     function test_rootIsInitializedCorrectly() external {
|         ^ (Relevant source part starts here and spans across multiple lines).
```

8.2 Tests Output

8.2.1 Circuit Test Output

The test outputs for sam_1024 and sam_2048 circuits are presented below. For clarity and brevity, warnings have been omitted, as they were already detailed in the compilation output.

```
sam_2048% nargo test
[samm_1024] Running 1 test function
[samm_1024] Testing test_sig_1024... ok
[samm_1024] 1 test passed
```

```
sam_2048% nargo test
[samm_2048] Running 1 test function
[samm_2048] Testing test_samm... ok
[samm_2048] 1 test passed
```

8.2.2 Smart Contract Output

The test output for the smart contracts are presented below.

```
samm-contracts % forge test
[] Compiling...
No files changed, compilation skipped

Ran 6 tests for test/unit/SAMSetup.t.sol:SAMExecuteTxTest
[PASS] test_impossibleToSetupMultiplyTimes() (gas: 19057)
[PASS] test_rootIsInitializedCorrectly() (gas: 15741)
[PASS] test_setupWithZeroRootWillRevert() (gas: 104532)
[PASS] test_setupWithZeroSafeWillRevert() (gas: 102444)
[PASS] test_setupWithZeroThresholdWillRevert() (gas: 104583)
[PASS] test_singletonSetupWillRevert() (gas: 14208)
Suite result: ok. 6 passed; 0 failed; 0 skipped; finished in 5.70s (1.71s CPU time)

Ran 3 tests for test/unit/GuardSetup.t.sol:GuardTest
[PASS] test_safeIsInitializedCorrectly() (gas: 18059)
[PASS] test_safeIsSetCorrectly() (gas: 96860)
[PASS] test_singletonSetupWillRevert() (gas: 10537)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 5.71s (3.63ms CPU time)

Ran 9 tests for test/unit/SAMExecuteTx.t.sol:SAMExecuteTxTest
[PASS] test_correctProofCanBeVerifiedAndTxExecuted() (gas: 1382396)
[PASS] test_correctProofCanBeVerifiedAndTxExecutedReturnData() (gas: 1383605)
[PASS] test_incorrectProofWillRevert() (gas: 865141)
[PASS] test_moduleWithoutSetupCantCallTheWallet() (gas: 207719)
[PASS] test_notAllowedTxWillRevert() (gas: 99531)
[PASS] test_notEnoughProofsWillRevert() (gas: 147347)
[PASS] test_sameProofCantBeUsedTwiceInSameTx() (gas: 1402456)
[PASS] test_setterWillRevertIfNotSafe() (gas: 40577)
[PASS] test_walletCanSetParamsDirectly() (gas: 125470)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 6.47s (4.03s CPU time)

Ran 3 test suites in 6.47s (17.87s CPU time): 18 tests passed, 0 failed, 0 skipped (18 total tests)
```

9 About Nethermind

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

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

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

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

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

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

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

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

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

Learn more about us at nethermind.io.

General Advisory to Clients

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

Disclaimer

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