Security Review Report
NM-0393 Data Ownership Protocol (DOP)
(DOP Labs Circuits)

NETHERMIND
SECURITY

(Jan 14, 2025)

Contents

1 Executive Summary **2**

2 Audited Files **3**

3 Summary of Issues **3**

4 System Overview **4**
4.1 Private and public transactions . . . 4
4.2 Selective disclosure . . . 5

5 Risk Rating Methodology **6**

6 Issues **7**
6.1 [Critical] Burning users' funds with incorrect nullifiers . . . 7
6.2 [Critical] Token which was not deposited can be transferred . . . 7
6.3 [High] The feature user-controlled selective disclosure fails when there is no nullifier . . . 8
6.4 [Medium] The `nullifiers` are not checked during the selective-transparency . . . 9
6.5 [Medium] The strict equality check in selective transparency may lead to data leak . . . 9
6.6 [Info] Centralization Risk . . . 9
6.7 [Info] Not all public inputs are verified in the signature check . . . 10
6.8 [Best Practices] `nOutputs` is not needed in SelectiveTransparency . . . 10

7 Documentation Evaluation **11**

8 Test Suite Evaluation **12**
8.1 Compilation Output . . . 12
8.2 Tests Output . . . 25

9 About Nethermind **26**

1 Executive Summary

This document outlines the security review conducted by Nethermind Security for Circom circuits of Data Ownership Protocol (DOP). *DOP* implements a non-custodial wallet that allows for public and private transfers of `ERC20` and `ERC721` tokens. Moreover, the wallet allows to selectively share specific account information and transaction details while keeping other data private.

The reviewed Circom circuits consist of 212 lines of code. The `DOP` team has shared information on the functionality of audited circuits through code comments, whitepaper, smart contracts, and meetings, but a detailed system specification is missing. This specification is crucial for identifying design flaws and ensuring secure implementation. The review of the Circom circuits was conducted in the context of Solidity smart contracts at the commit 3bb9905. Those contracts are out of scope, but were necessary to understand how the circuits are used and identify some of the issues. **The audit was performed using**: (a) manual analysis of the codebase, and (b) writing test cases. **Along this document, we report 7 points of attention summarized in Fig. 1**.

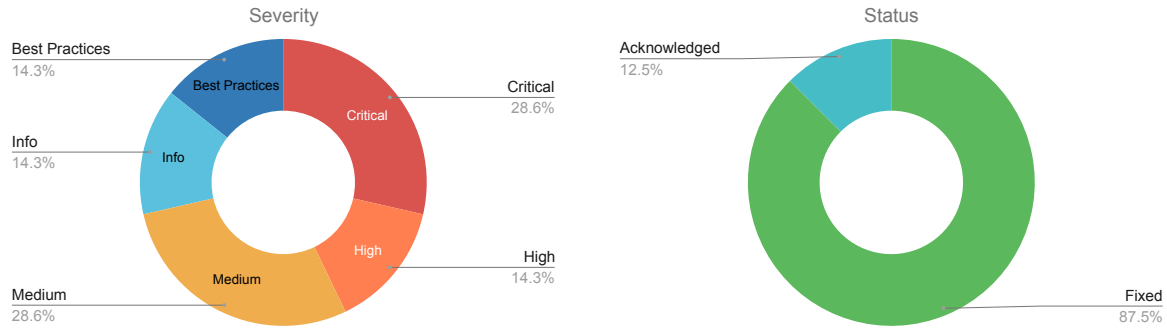
This document is organized as follows. Section 2 presents the files in the scope of this audit. Section 3 summarizes the issues. Section 4 presents the system overview. Section 5 discusses the risk rating methodology adopted for this audit. Section 6 details the issues. Section 7 discusses the documentation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

Disclaimer

During the analysis of the circuits implemented in Circom, we identified issues in the Solidity code of the associated smart contracts at the commit 3bb9905. Although some of these issues are reported in this document for the client's awareness, **the audit scope is strictly limited to the Circom circuits, and the Solidity code has not been audited**. Therefore, Nethermind Security cannot be held responsible for any issues arising from the Solidity code, as it falls outside the defined scope of this engagement.

Tests remarks

During the security review of circuit `SelectiveTransparency`, it was identified that the client did not implement specific tests to validate the behavior and integrity of this circuit. Conversely, `Transfer` circuit includes only a single test scenario, whereas it should have been tested in multiple scenarios. Furthermore, no scripts or documentation were provided to execute tests related to all circuits in the scope, compromising the ability to verify the correct functionality of the designed features in an automated and standardized way. We recommend that the DOP team develop tests for the circuit `SelectiveTransparency` and provide scripts to facilitate the execution and replication of these tests to ensure reliability and security.



(a) distribution of issues according to the severity

(b) distribution of issues according to the status

Fig 1: (a) Distribution of issues: Critical (2), **High** (1), **Medium** (2), **Low** (0), **Undetermined** (0), **Informational** (1), **Best Practices** (1). **(b) Distribution of status: Fixed** (7), **Acknowledged** (1), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	Dec 20, 2024
Final Report	Jan 14, 2025
Methods	Manual Review, Tests
Repository	protocol-circuits
Commit Hash - transfer.circom	8b48e1d480fe1d789f26947ef985f43acfd39ef0
Commit Hash - merkle-proof-verifier.circom	8b48e1d480fe1d789f26947ef985f43acfd39ef0
Commit Hash - nullifier-check.circom	8b48e1d480fe1d789f26947ef985f43acfd39ef0
Commit Hash - selective-transparency.circom	552f7ac0cc04d6a9c52c26e3067c94604952c240
Final Commit Hash	68d18a3459410784ea9fe38f306084300f0dd671
Documentation	Communication, white paper, and NatSpec
Documentation Assessment	Medium
Test Suite Assessment	Low

2 Audited Files

	Circuit	LoC	Comments	Ratio	Blank	Total
1	src/library/nullifier-check.circom	16	0	0.0%	4	20
2	src/library/transfer.circom	91	24	26.4%	20	135
3	src/library/merkle-proof-verifier.circom	32	1	3.1%	7	40
4	src/library/selective-transparency .circom	73	19	26.0%	23	115
	Total	**212**	**44**	**20.8%**	**54**	**310**

3 Summary of Issues

	Finding	Severity	Update
1	Burning users' funds with incorrect nullifiers	Critical	Fixed
2	Token which was not deposited can be transferred	Critical	Fixed
3	The feature user-controlled selective disclosure fails when there is no nullifier	High	Fixed
4	The `nullifiers` are not checked during the selective-transparency	Medium	Fixed
5	The strict equality check in selective transparency may lead to data leak	Medium	Fixed
6	Centralization Risk	Info	Acknowledged
7	Not all public inputs are verified in the signature check	Info	Fixed
8	`nOutputs` is not needed in SelectiveTransparency	Best Practices	Fixed

4 System Overview

The DOP protocol is a private fund management solution that allows users to send private transactions, viewable only by the recipient. The DOP wallet also supports public transactions and the disclosure of private data through selective disclosure actions. The wallet infrastructure consists of two parts: common on-chain contracts and off-chain infrastructure, which includes circuits. Users can participate in the protocol through `DOPSmartWallet` by depositing their funds into an on-chain public vault and requesting "encoding" of their funds. Once encoded, funds are transferred from the vault to the DOPSmartWallet and mixed with other users' funds. The deposited token and amount data are added as a commitment to a Merkle tree, and the user can then perform private transactions. A user's balance is represented similarly to a UTXO (Unspent Transaction Output) model, with the deposited or transferred token amounts added as a `leaf = hash(npk || token || amount)` to a Merkle tree, where npk is the user identification. As the Merkle tree is add-only, nullifiers are used to mark leaves that are already used during fund transfers.

4.1 Private and public transactions

The user must first generate the proof to initiate a transaction, providing the correct Merkle root (from the on-chain contract), commitments that prove the user holds funds, and nullifiers that verify the user hasn't spent those funds yet. The proof generation happens on the user's infrastructure (mobile device/laptop/PC), and all the keys are held and controlled only by the user. The circuit verifies that the provided leaves (`hash(hash(hash(publicKey || nullifyingKey) || randomIn) || token || valueIn))` are in the Merkle tree with the corresponding `merkleRoot` and check if the provided `nullifiers` correspond to provided leaves. It also ensures that the total value of the tokens in the Merkle tree (`valueIn`) equals the total value of transfers (`valueOut`). This check is crucial since it ensures that the user has enough funds for the transfer. The operations of the circuit are presented in the **Fig. 2**. Note that the data is different depending on the token type. The `ERC20` tokens are represented as `token: actual address of a token, value: amount of tokens`, whereas for ERC721 `token: hash(token type || token address || token ID), value: always 1`. After the proof is generated, it is supplied on-chain to `DOPSmartWallet`. The contract validates transaction parameters and marks provided nullifiers as used. If the transaction is private, there are no actual token transfers, but only the Merkle root is updated to a new one that contains the provided commitments that represent the recipient's right to tokens (`note = hash(recipient_npk || token || amount)`). If the transaction is public, the Merkle root is not updated, and the tokens are transferred out of the contract to a specified address.

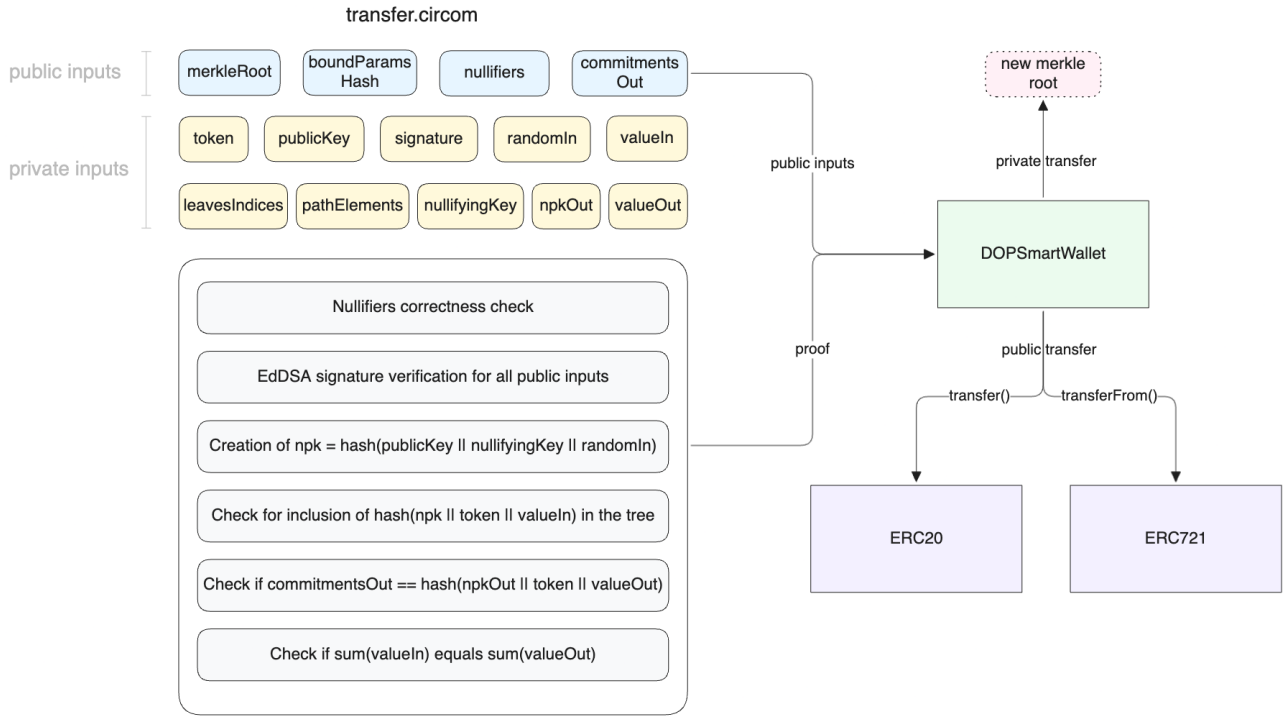


Fig. 2: The flow of transfer action. The proof generated by the user is supplied on-chain for performing private transfers (updating Merkle tree root) or public transfers (sending tokens out of the wallet).

4.2 Selective disclosure

Users can disclose the chosen part of their data. For example, users who hold 10 USDC, 50 USDC, and 30 WETH can prove and publish that they hold 10 USDC. The circuit flow is similar to the one in the `transfer.circom` circuit. The circuit ensures that provided token values corresponding to the user's `npk` exist in the tree with corresponding nullifiers and checks if the sum of those tokens equals the `valueCheck`, which is the value being disclosed. In the case of `ERC721` tokens, the user holds a specific token ID, and the proved value in `valueCheck` is always equal to `1`. The on-chain contract validates if the proof with corresponding public inputs is correct.

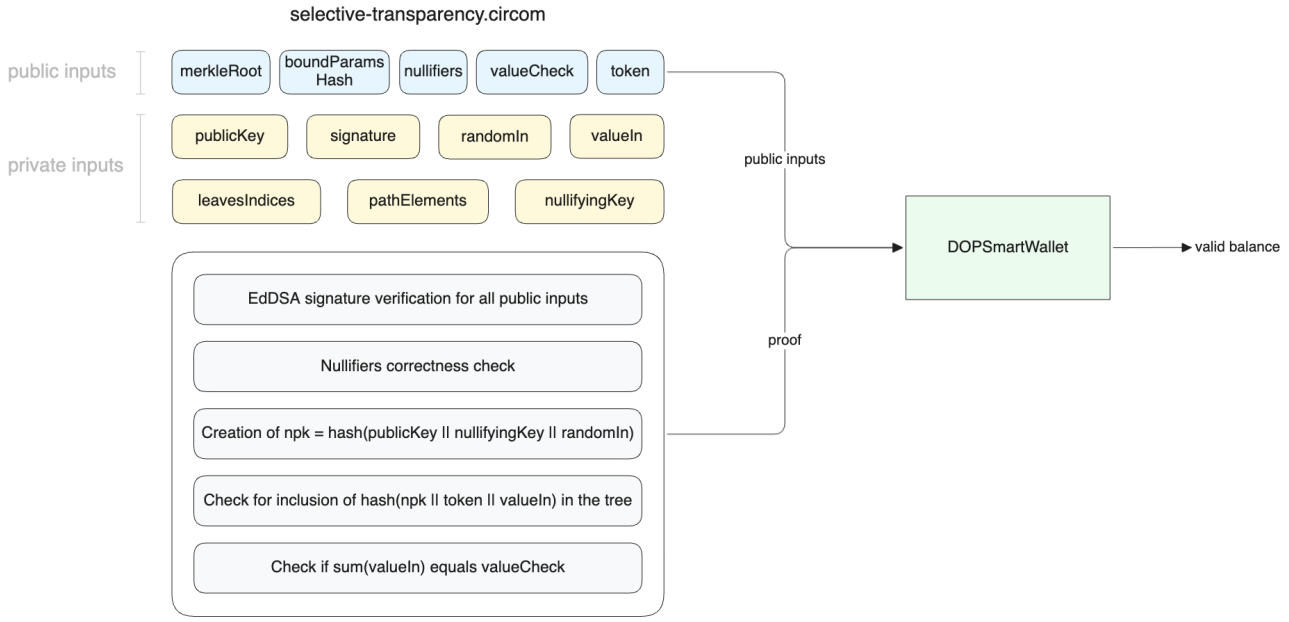


Fig. 3: The flow of selective disclosure action. The proof generated by the user is supplied on-chain for check of correctness of published data.

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] Burning users' funds with incorrect nullifiers

File(s): transfer.circom

Description: The transfer circuit, when receiving zero in the `valueIn` parameter, disables checks for nullifiers and the Merkle tree inclusion of leaves. However, this allows for creating a proof of transaction with nullifiers of another user. This transaction can be executed on-chain with the correct proof, and malicious nullifiers could be used to nullify another user's leaves, leading to the burning of the user's funds. The knowledge of nullifiers would not be difficult, as users post their nullifiers during selective disclosure.

Recommendation(s): Consider removing the `isDummy` mechanism. Alternatively, do not allow passing zero values on deposit/transfer and execute checks of nullifiers and Merkle tree inclusion for any value, including zero.

Status: Fixed.

Update from the client: The `isDummy` mechanism is removed. Commit: 68d18a3.

6.2 [Critical] Token which was not deposited can be transferred

File(s): transfer.circom

Description: When the `valueIn` provided to the circuit is `0`, the circuit does not perform checks on the Merkle tree inclusion of notes or nullifiers' correctness. If both `valueIn` and `valueOut` are provided as zero, the sums would be equal. The malicious user can provide a `token` that was not deposited by them since the Merkle tree inclusion is not checked. This may lead to a few vulnerabilities during a transfer.

Scenario 1: The provided `token` may be an NFT token identifier: `hash(tokenType: ERC721, tokenAddress, tokenSubID)` that exists in a wallet and was deposited by another user, e.g., `hash(ERC721, 0x123, ID: 1)`. In the case of `valueIn equals to valueOut equals to 0` provided in the circuit, the malicious user can provide such `token` and the circuit would execute correctly.

The proved input note would look like: `note equals to hash(holder_npk, token: hash(ERC721, 0x123, ID: 1), value: 0)` - such note would not exist in the tree (all ERC721 tokens must have `value = 1` during deposit) but since the Merkle tree inclusion is not checked the proof would be generated correctly. The malicious user could provide this token (deposited by another user) in `commitmentsOut: hash(malicious_receiver_address, token: hash(ERC721, 0x123, ID: 1), value: 0)`.

During the on-chain verification, the invalid input note would be added to a tree, invalid nullifiers would be marked, and the transfer would be executed correctly, effectively allowing to steal the token from another user. Note that the note is not checked to have `value == 1` during the public transfer.

Scenario 2: Another possible scenario is when a malicious user correctly deposited their NFT to the wallet and created a note: `hash(ERC721, 0x123, ID: 1)`. Then they use the exploit described above where the `valueIn == valueOut == 0` and provided `commitmentsIn` and `nullifiers` are incorrect and do not point to the correct note. The user transfers this NFT from the wallet, but the corresponding nullifier is unused. Then, the malicious user can use `selective transparency` to prove that they still hold the NFT in the wallet since the correct nullifier was not used.

The provided `token` may be an Ethereum address that does not hold any code. If such a transfer is public and called in a batch, then the ERC20 safe transfer would revert and the batch. This could lead to DOS of batches of transactions executed by a third party.

Recommendation(s): Consider removing the `isDummy` mechanism. Alternatively consider checking for Merkle root inclusion and nullifier correctness on every `value`, including zero. Additionally, consider asserting that the `note.value` equals `1` during the transfer of ERC721.

Status: Fixed

Update from the client: The `isDummy` mechanism is removed. Commit: 68d18a3.

6.3 [High] The feature user-controlled selective disclosure fails when there is no nullifier

File(s): selective-transparency.circom

Description: The SelectiveTransparency circuit verifies that a user owns more than a specified threshold amount (valueCheck) of a given token. In this design, the use of a nullifier is optional. The public inputs to this feature include:

```
merkleRoot, boundParamsHash, nullifiers[nInputs], valueCheck, token
```

As the valueCheck input must always be greater than zero, the proof cannot be generated if **there is no nullifier** for input notes. The scenarios and their implications are outlined below:

Scenario 1: nInputs > 0 and valueCheck > 0. In this case, the prover will fail due to the inability to generate the signature. The message hash to be signed is constructed as follows:

```
messageHash = hash(merkleRoot, boundParamsHash, nullifiers)
```

Here, nullifiers is a list of length nInputs. If the list does not contain nullifiers, the message generation for signing fails. Even if the message is signed with this list without any nullifier, the *Nullifier Check* step must be skipped because there is no nullifier to verify. For this step to succeed, the condition isDummy[i].out = 0 must hold for all indices in the range [0:nInputs). The process for handling nullifiers is defined as:

```
for (var i = 0; i < nInputs; i++) {
    nullifiersHash[i] = NullifierCheck();
    nullifiersHash[i].nullifyingKey <== nullifyingKey;
    nullifiersHash[i].leafIndex <== leavesIndices[i];
    nullifiersHash[i].nullifier <== nullifiers[i];
    // @audit To disable the nullifier check for the whole nullifiers array, isDummy[i].out should always be zero
    nullifiersHash[i].enabled <== 1-isDummy[i].out;
}
```

The isDummy component is set with valueIn[nInputs] private input to verify against zero values. Consequently, valueIn must be zero for all indices in the range [0:nInputs]. This results in a sumIn = 0, causing the verification sumIn === valueCheck to fail.

Scenario 2: nInputs = 0 and valueCheck > 0 This scenario also fails. With nInputs = 0, all inputs associated with it such as, nullifiers[nInputs] and private list inputs (e.g., valueIn[nInputs]) are empty. The message hash for signature generation simplifies to:

```
messageHash = hash(merkleRoot, boundParamsHash)
```

As nInputs = 0, nullifier and Merkle proof checks are skipped. The sumIn value remains zero, and the verification sumIn === valueCheck fails.

Scenario 3: nInputs = 0 and valueCheck = 0 This scenario passes. With nInputs = 0, all inputs associated with it, such as nullifiers[nInputs] and private list inputs (e.g., valueIn[nInputs]), are empty. The message hash for signature generation simplifies to:

```
messageHash = hash(merkleRoot, boundParamsHash)
```

As nInputs = 0, nullifier and Merkle proof checks are skipped. The sumIn value remains zero, and the verification sumIn === valueCheck passes. This allows a user to provide no nullifiers and generate a valid proof.

Recommendation(s): Ensure valueCheck is greater than zero.

Status: Fixed.

Update from the client: Fixed at commit 6a97b0c3

6.4 [Medium] The `nullifiers` are not checked during the selective-transparency

File(s): `selective-transparency.circom`

Description: The `selective-transparency` circuit allows users to prove holding some amount of funds. However, the `nullifiers` are not checked on-chain if they have not been used before. In effect, it is proved that the user held the disclosed amount at some point in history, but it is not ensured that the user still holds those funds. Note that anyone can still check the `nullifiers'` status on-chain but expects to perform additional checks from the information receiver. As a result, users can lie that they hold some value even if it was already spent if the information recipient won't compare it with the on-chain state.

Recommendation(s): Consider adding on-chain logic to ensure that the nullifiers are not already used.

Status: Fixed

Update from the client: Added the check on-chain to check if the nullifier is used or not. Commit: https://github.com/dop-labs/protocol-contracts/commit/c9f5f2bcdebe21dc0a6b7ab576549ca1bcd71441

6.5 [Medium] The strict equality check in selective transparency may lead to data leak

File(s): `selective-transparency.circom`

Description: The `selective-transparency` circuit does a strict equality check at the end, checking if the `sumIn` equals the public input `valueCheck`. This may lead to data leakage since if the user provided only one input note, it must be exactly equal to the `valueCheck`, revealing the exact note held in the tree.

Recommendation(s): Consider changing the strict equality check to a greater or equal one to ensure that data in the note is not revealed.

Status: Fixed

Update from the client: Fixed the equality check from equal to greater or equal. Commit: `cc4b3e2d`

6.6 [Info] Centralization Risk

File(s): `verifier.sol`

Description: The verification keys are set by the owner of the `Verifier.sol` contract, and it is possible to change the owner due to `OwnableUpgradeable`. Therefore, the new or original owner could maliciously change the values of each verifying key, thus preventing the circuits from verifying.

Recommendation(s): Create a function that allows only adding to the list of verifying keys.

Status: Acknowledged

Update from the client: We need the possibility for removing old keys if a vulnerability is found in a circuit so we can override a verification key for a certain number of inputs and outputs.

6.7 [Info] Not all public inputs are verified in the signature check

File(s): selective-transparency.circom

Description: The `SelectiveTransparency` computes the hash over public inputs to generate the signed message. These signals are listed below:

```
template SelectiveTransparency(nInputs, nOutputs, MerkleTreeDepth) {
    /* ========== PUBLIC SIGNALS ========== */

    signal input merkleRoot;
    signal input boundParamsHash;
    signal input nullifiers[nInputs];
    signal input valueCheck;
    signal input token;
...
}
```

However, the computed hash does not contain `valueCheck` and `token`:

```
    // 1. Compute hash over public signals to get the signed message
    var size = nInputs + 2;
    component messageHash = Poseidon(size);
    messageHash.inputs[0] <== merkleRoot;
    messageHash.inputs[1] <== boundParamsHash;

    for (var i = 0; i < nInputs; i++) {
        messageHash.inputs[i+2] <== nullifiers[i];
    }
   // @audit valueCheck and token are not hashed

    // 2. Verify EDDSA signature over hash of public inputs
    component eddsaVerifier = EdDSAPoseidonVerifier();
    eddsaVerifier.enabled <== 1;
    eddsaVerifier.Ax <== publicKey[0];
    eddsaVerifier.Ay <== publicKey[1];
    eddsaVerifier.R8x <== signature[0];
    eddsaVerifier.R8y <== signature[1];
    eddsaVerifier.S <== signature[2];
    eddsaVerifier.M <== messageHash.out;
```

If the message signed contains both `valueCheck` and `token`, the signature verification step will fail.

Recommendation(s): Add `valueCheck` and `token` to `messageHash`.

Status: Fixed

Update from the client: All public inputs are added to the signature now. Commit: cc4b3e2d

6.8 [Best Practices] `nOutputs` is not needed in SelectiveTransparency

File(s): selective-transparency.circom

Description: The `nOutputs` is not used in the selective transparency logic but is passed as an argument for the circuit.

```
template SelectiveTransparency(nInputs, nOutputs, MerkleTreeDepth) {
```

Recommendation: Remove the nOutputs

Status: Fixed

Update from the client: Removed. Commit: bc127005

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., nonzero).

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 and 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 were tested. It includes details on 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 on 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 Circuits documentation

The `DOP` team has provided code comments explaining the inputs and flow within the `Transfer` and `SelectiveTransparency` circuits. Their functionality is outlined in whitepaper, Solidity smart contracts with the commit hash 3bb990, and clarifications during meetings. However, there is no comprehensive specification of the system requirements that clearly outlines how the protocol should operate and the specific role of verifications within the circuits. This specification is crucial for identifying design flaws and validating the implementation of these requirements from a security point of view.

8 Test Suite Evaluation

8.1 Compilation Output

```
% ./2_compile_circuits.sh
Compiling circuits
template instances: 328
non-linear constraints: 55353
linear constraints: 55752
public inputs: 13
private inputs: 199
public outputs: 0
wires: 111277
labels: 175518
Written successfully: ./transfer_10x1.r1cs
Written successfully: ./transfer_10x1.sym
Constraints written in: ./transfer_10x1_constraints.json
Written successfully: ./transfer_10x1_cpp/transfer_10x1.cpp and ./transfer_10x1_cpp/transfer_10x1.dat
Written successfully: ./transfer_10x1_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_10x1_js/transfer_10x1.wat
Written successfully: ./transfer_10x1_js/transfer_10x1.wasm
Everything went okay
template instances: 318
non-linear constraints: 55731
linear constraints: 56019
public inputs: 14
private inputs: 201
public outputs: 0
wires: 111923
labels: 176441
Written successfully: ./transfer_10x2.r1cs
Written successfully: ./transfer_10x2.sym
Constraints written in: ./transfer_10x2_constraints.json
Written successfully: ./transfer_10x2_cpp/transfer_10x2.cpp and ./transfer_10x2_cpp/transfer_10x2.dat
Written successfully: ./transfer_10x2_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_10x2_js/transfer_10x2.wat
Written successfully: ./transfer_10x2_js/transfer_10x2.wasm
Everything went okay
template instances: 322
non-linear constraints: 56151
linear constraints: 56504
public inputs: 15
private inputs: 203
public outputs: 0
wires: 112829
labels: 177828
Written successfully: ./transfer_10x3.r1cs
Written successfully: ./transfer_10x3.sym
Constraints written in: ./transfer_10x3_constraints.json
Written successfully: ./transfer_10x3_cpp/transfer_10x3.cpp and ./transfer_10x3_cpp/transfer_10x3.dat
Written successfully: ./transfer_10x3_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_10x3_js/transfer_10x3.wat
Written successfully: ./transfer_10x3_js/transfer_10x3.wasm
Everything went okay
template instances: 326
non-linear constraints: 56571
linear constraints: 56997
public inputs: 16
private inputs: 205
public outputs: 0
wires: 113743
labels: 179231
Written successfully: ./transfer_10x4.r1cs
Written successfully: ./transfer_10x4.sym
```

```
Constraints written in: ./transfer_10x4_constraints.json
Written successfully: ./transfer_10x4_cpp/transfer_10x4.cpp and ./transfer_10x4_cpp/transfer_10x4.dat
Written successfully: ./transfer_10x4_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_10x4_js/transfer_10x4.wat
Written successfully: ./transfer_10x4_js/transfer_10x4.wasm
Everything went okay
template instances: 318
non-linear constraints: 60025
linear constraints: 60987
public inputs: 14
private inputs: 218
public outputs: 0
wires: 121201
labels: 190295
Written successfully: ./transfer_11x1.r1cs
Written successfully: ./transfer_11x1.sym
Constraints written in: ./transfer_11x1_constraints.json
Written successfully: ./transfer_11x1_cpp/transfer_11x1.cpp and ./transfer_11x1_cpp/transfer_11x1.dat
Written successfully: ./transfer_11x1_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_11x1_js/transfer_11x1.wat
Written successfully: ./transfer_11x1_js/transfer_11x1.wasm
Everything went okay
template instances: 322
non-linear constraints: 64739
linear constraints: 66440
public inputs: 15
private inputs: 237
public outputs: 0
wires: 131385
labels: 205536
Written successfully: ./transfer_12x1.r1cs
Written successfully: ./transfer_12x1.sym
Constraints written in: ./transfer_12x1_constraints.json
Written successfully: ./transfer_12x1_cpp/transfer_12x1.cpp and ./transfer_12x1_cpp/transfer_12x1.dat
Written successfully: ./transfer_12x1_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_12x1_js/transfer_12x1.wat
Written successfully: ./transfer_12x1_js/transfer_12x1.wasm
Everything went okay
template instances: 326
non-linear constraints: 69453
linear constraints: 71901
public inputs: 16
private inputs: 256
public outputs: 0
wires: 141577
labels: 220793
Written successfully: ./transfer_13x1.r1cs
Written successfully: ./transfer_13x1.sym
Constraints written in: ./transfer_13x1_constraints.json
Written successfully: ./transfer_13x1_cpp/transfer_13x1.cpp and ./transfer_13x1_cpp/transfer_13x1.dat
Written successfully: ./transfer_13x1_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_13x1_js/transfer_13x1.wat
Written successfully: ./transfer_13x1_js/transfer_13x1.wasm
Everything went okay
template instances: 318
non-linear constraints: 13005
linear constraints: 7136
public inputs: 4
private inputs: 28 (27 belong to witness)
public outputs: 0
wires: 20160
labels: 39325
Written successfully: ./transfer_1x1.r1cs
Written successfully: ./transfer_1x1.sym
Constraints written in: ./transfer_1x1_constraints.json
Written successfully: ./transfer_1x1_cpp/transfer_1x1.cpp and ./transfer_1x1_cpp/transfer_1x1.dat
Written successfully: ./transfer_1x1_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_1x1_js/transfer_1x1.wat
```

```
Written successfully: ./transfer_1x1_js/transfer_1x1.wasm
Everything went okay
template instances: 318
non-linear constraints: 13005
linear constraints: 7136
public inputs: 4
private inputs: 28 (27 belong to witness)
public outputs: 0
wires: 20160
labels: 39325
Written successfully: ./transfer_1x10.r1cs
Written successfully: ./transfer_1x10.sym
Constraints written in: ./transfer_1x10_constraints.json
Written successfully: ./transfer_1x10_cpp/transfer_1x10.cpp and ./transfer_1x10_cpp/transfer_1x10.dat
Written successfully: ./transfer_1x10_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_1x10_js/transfer_1x10.wat
Written successfully: ./transfer_1x10_js/transfer_1x10.wasm
Everything went okay
template instances: 326
non-linear constraints: 17925
linear constraints: 12285
public inputs: 16
private inputs: 52
public outputs: 0
wires: 30241
labels: 54545
Written successfully: ./transfer_1x13.r1cs
Written successfully: ./transfer_1x13.sym
Constraints written in: ./transfer_1x13_constraints.json
Written successfully: ./transfer_1x13_cpp/transfer_1x13.cpp and ./transfer_1x13_cpp/transfer_1x13.dat
Written successfully: ./transfer_1x13_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_1x13_js/transfer_1x13.wat
Written successfully: ./transfer_1x13_js/transfer_1x13.wasm
Everything went okay
template instances: 245
non-linear constraints: 13413
linear constraints: 7554
public inputs: 5
private inputs: 30
public outputs: 0
wires: 20987
labels: 40568
Written successfully: ./transfer_1x2.r1cs
Written successfully: ./transfer_1x2.sym
Constraints written in: ./transfer_1x2_constraints.json
Written successfully: ./transfer_1x2_cpp/transfer_1x2.cpp and ./transfer_1x2_cpp/transfer_1x2.dat
Written successfully: ./transfer_1x2_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_1x2_js/transfer_1x2.wat
Written successfully: ./transfer_1x2_js/transfer_1x2.wasm
Everything went okay
template instances: 321
non-linear constraints: 13830
linear constraints: 7995
public inputs: 6
private inputs: 32
public outputs: 0
wires: 21846
labels: 41865
Written successfully: ./transfer_1x3.r1cs
Written successfully: ./transfer_1x3.sym
Constraints written in: ./transfer_1x3_constraints.json
Written successfully: ./transfer_1x3_cpp/transfer_1x3.cpp and ./transfer_1x3_cpp/transfer_1x3.dat
Written successfully: ./transfer_1x3_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_1x3_js/transfer_1x3.wat
Written successfully: ./transfer_1x3_js/transfer_1x3.wasm
Everything went okay
template instances: 322
non-linear constraints: 14241
linear constraints: 8424
```

```
public inputs: 7
private inputs: 34
public outputs: 0
wires: 22687
labels: 43134
Written successfully: ./transfer_1x4.r1cs
Written successfully: ./transfer_1x4.sym
Constraints written in: ./transfer_1x4_constraints.json
Written successfully: ./transfer_1x4_cpp/transfer_1x4.cpp and ./transfer_1x4_cpp/transfer_1x4.dat
Written successfully: ./transfer_1x4_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_1x4_js/transfer_1x4.wat
Written successfully: ./transfer_1x4_js/transfer_1x4.wasm
Everything went okay
template instances: 321
non-linear constraints: 14646
linear constraints: 8835
public inputs: 8
private inputs: 36
public outputs: 0
wires: 23504
labels: 44363
Written successfully: ./transfer_1x5.r1cs
Written successfully: ./transfer_1x5.sym
Constraints written in: ./transfer_1x5_constraints.json
Written successfully: ./transfer_1x5_cpp/transfer_1x5.cpp and ./transfer_1x5_cpp/transfer_1x5.dat
Written successfully: ./transfer_1x5_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_1x5_js/transfer_1x5.wat
Written successfully: ./transfer_1x5_js/transfer_1x5.wasm
Everything went okay
template instances: 245
non-linear constraints: 17707
linear constraints: 12522
public inputs: 5
private inputs: 47
public outputs: 0
wires: 30265
labels: 54422
Written successfully: ./transfer_2x1.r1cs
Written successfully: ./transfer_2x1.sym
Constraints written in: ./transfer_2x1_constraints.json
Written successfully: ./transfer_2x1_cpp/transfer_2x1.cpp and ./transfer_2x1_cpp/transfer_2x1.dat
Written successfully: ./transfer_2x1_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_2x1_js/transfer_2x1.wat
Written successfully: ./transfer_2x1_js/transfer_2x1.wasm
Everything went okay
template instances: 321
non-linear constraints: 18124
linear constraints: 12963
public inputs: 6
private inputs: 49
public outputs: 0
wires: 31124
labels: 55719
Written successfully: ./transfer_2x2.r1cs
Written successfully: ./transfer_2x2.sym
Constraints written in: ./transfer_2x2_constraints.json
Written successfully: ./transfer_2x2_cpp/transfer_2x2.cpp and ./transfer_2x2_cpp/transfer_2x2.dat
Written successfully: ./transfer_2x2_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_2x2_js/transfer_2x2.wat
Written successfully: ./transfer_2x2_js/transfer_2x2.wasm
Everything went okay
template instances: 322
non-linear constraints: 18535
linear constraints: 13392
public inputs: 7
private inputs: 51
public outputs: 0
wires: 31965
labels: 56988
Written successfully: ./transfer_2x3.r1cs
```

```
Written successfully: ./transfer_2x3.sym
Constraints written in: ./transfer_2x3_constraints.json
Written successfully: ./transfer_2x3_cpp/transfer_2x3.cpp and ./transfer_2x3_cpp/transfer_2x3.dat
Written successfully: ./transfer_2x3_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_2x3_js/transfer_2x3.wat
Written successfully: ./transfer_2x3_js/transfer_2x3.wasm
Everything went okay
template instances: 321
non-linear constraints: 18940
linear constraints: 13803
public inputs: 8
private inputs: 53
public outputs: 0
wires: 32782
labels: 58217
Written successfully: ./transfer_2x4.r1cs
Written successfully: ./transfer_2x4.sym
Constraints written in: ./transfer_2x4_constraints.json
Written successfully: ./transfer_2x4_cpp/transfer_2x4.cpp and ./transfer_2x4_cpp/transfer_2x4.dat
Written successfully: ./transfer_2x4_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_2x4_js/transfer_2x4.wat
Written successfully: ./transfer_2x4_js/transfer_2x4.wasm
Everything went okay
template instances: 318
non-linear constraints: 19339
linear constraints: 14190
public inputs: 9
private inputs: 55
public outputs: 0
wires: 33569
labels: 59394
Written successfully: ./transfer_2x5.r1cs
Written successfully: ./transfer_2x5.sym
Constraints written in: ./transfer_2x5_constraints.json
Written successfully: ./transfer_2x5_cpp/transfer_2x5.cpp and ./transfer_2x5_cpp/transfer_2x5.dat
Written successfully: ./transfer_2x5_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_2x5_js/transfer_2x5.wat
Written successfully: ./transfer_2x5_js/transfer_2x5.wasm
Everything went okay
template instances: 321
non-linear constraints: 22418
linear constraints: 17931
public inputs: 6
private inputs: 66
public outputs: 0
wires: 40402
labels: 69573
Written successfully: ./transfer_3x1.r1cs
Written successfully: ./transfer_3x1.sym
Constraints written in: ./transfer_3x1_constraints.json
Written successfully: ./transfer_3x1_cpp/transfer_3x1.cpp and ./transfer_3x1_cpp/transfer_3x1.dat
Written successfully: ./transfer_3x1_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_3x1_js/transfer_3x1.wat
Written successfully: ./transfer_3x1_js/transfer_3x1.wasm
Everything went okay
template instances: 322
non-linear constraints: 22829
linear constraints: 18360
public inputs: 7
private inputs: 68
public outputs: 0
wires: 41243
labels: 70842
Written successfully: ./transfer_3x2.r1cs
Written successfully: ./transfer_3x2.sym
Constraints written in: ./transfer_3x2_constraints.json
Written successfully: ./transfer_3x2_cpp/transfer_3x2.cpp and ./transfer_3x2_cpp/transfer_3x2.dat
Written successfully: ./transfer_3x2_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
```

```
Written successfully: ./transfer_3x2_js/transfer_3x2.wat
Written successfully: ./transfer_3x2_js/transfer_3x2.wasm
Everything went okay
template instances: 321
non-linear constraints: 23234
linear constraints: 18771
public inputs: 8
private inputs: 70
public outputs: 0
wires: 42060
labels: 72071
Written successfully: ./transfer_3x3.r1cs
Written successfully: ./transfer_3x3.sym
Constraints written in: ./transfer_3x3_constraints.json
Written successfully: ./transfer_3x3_cpp/transfer_3x3.cpp and ./transfer_3x3_cpp/transfer_3x3.dat
Written successfully: ./transfer_3x3_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_3x3_js/transfer_3x3.wat
Written successfully: ./transfer_3x3_js/transfer_3x3.wasm
Everything went okay
template instances: 318
non-linear constraints: 23633
linear constraints: 19158
public inputs: 9
private inputs: 72
public outputs: 0
wires: 42847
labels: 73248
Written successfully: ./transfer_3x4.r1cs
Written successfully: ./transfer_3x4.sym
Constraints written in: ./transfer_3x4_constraints.json
Written successfully: ./transfer_3x4_cpp/transfer_3x4.cpp and ./transfer_3x4_cpp/transfer_3x4.dat
Written successfully: ./transfer_3x4_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_3x4_js/transfer_3x4.wat
Written successfully: ./transfer_3x4_js/transfer_3x4.wasm
Everything went okay
template instances: 324
non-linear constraints: 24059
linear constraints: 19647
public inputs: 10
private inputs: 74
public outputs: 0
wires: 43763
labels: 74647
Written successfully: ./transfer_3x5.r1cs
Written successfully: ./transfer_3x5.sym
Constraints written in: ./transfer_3x5_constraints.json
Written successfully: ./transfer_3x5_cpp/transfer_3x5.cpp and ./transfer_3x5_cpp/transfer_3x5.dat
Written successfully: ./transfer_3x5_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_3x5_js/transfer_3x5.wat
Written successfully: ./transfer_3x5_js/transfer_3x5.wasm
Everything went okay
template instances: 322
non-linear constraints: 27123
linear constraints: 23328
public inputs: 7
private inputs: 85
public outputs: 0
wires: 50521
labels: 84696
Written successfully: ./transfer_4x1.r1cs
Written successfully: ./transfer_4x1.sym
Constraints written in: ./transfer_4x1_constraints.json
Written successfully: ./transfer_4x1_cpp/transfer_4x1.cpp and ./transfer_4x1_cpp/transfer_4x1.dat
Written successfully: ./transfer_4x1_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_4x1_js/transfer_4x1.wat
Written successfully: ./transfer_4x1_js/transfer_4x1.wasm
Everything went okay
template instances: 321
non-linear constraints: 27528
linear constraints: 23739
```

```
public inputs: 8
private inputs: 87
public outputs: 0
wires: 51338
labels: 85925
Written successfully: ./transfer_4x2.r1cs
Written successfully: ./transfer_4x2.sym
Constraints written in: ./transfer_4x2_constraints.json
Written successfully: ./transfer_4x2_cpp/transfer_4x2.cpp and ./transfer_4x2_cpp/transfer_4x2.dat
Written successfully: ./transfer_4x2_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_4x2_js/transfer_4x2.wat
Written successfully: ./transfer_4x2_js/transfer_4x2.wasm
Everything went okay
template instances: 318
non-linear constraints: 27927
linear constraints: 24126
public inputs: 9
private inputs: 89
public outputs: 0
wires: 52125
labels: 87102
Written successfully: ./transfer_4x3.r1cs
Written successfully: ./transfer_4x3.sym
Constraints written in: ./transfer_4x3_constraints.json
Written successfully: ./transfer_4x3_cpp/transfer_4x3.cpp and ./transfer_4x3_cpp/transfer_4x3.dat
Written successfully: ./transfer_4x3_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_4x3_js/transfer_4x3.wat
Written successfully: ./transfer_4x3_js/transfer_4x3.wasm
Everything went okay
template instances: 324
non-linear constraints: 28353
linear constraints: 24615
public inputs: 10
private inputs: 91
public outputs: 0
wires: 53041
labels: 88501
Written successfully: ./transfer_4x4.r1cs
Written successfully: ./transfer_4x4.sym
Constraints written in: ./transfer_4x4_constraints.json
Written successfully: ./transfer_4x4_cpp/transfer_4x4.cpp and ./transfer_4x4_cpp/transfer_4x4.dat
Written successfully: ./transfer_4x4_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_4x4_js/transfer_4x4.wat
Written successfully: ./transfer_4x4_js/transfer_4x4.wasm
Everything went okay
template instances: 318
non-linear constraints: 28743
linear constraints: 24960
public inputs: 11
private inputs: 93
public outputs: 0
wires: 53777
labels: 89588
Written successfully: ./transfer_4x5.r1cs
Written successfully: ./transfer_4x5.sym
Constraints written in: ./transfer_4x5_constraints.json
Written successfully: ./transfer_4x5_cpp/transfer_4x5.cpp and ./transfer_4x5_cpp/transfer_4x5.dat
Written successfully: ./transfer_4x5_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_4x5_js/transfer_4x5.wat
Written successfully: ./transfer_4x5_js/transfer_4x5.wasm
Everything went okay
template instances: 321
non-linear constraints: 31822
linear constraints: 28707
public inputs: 8
private inputs: 104
public outputs: 0
wires: 60616
```

```
labels: 99779
Written successfully: ./transfer_5x1.r1cs
Written successfully: ./transfer_5x1.sym
Constraints written in: ./transfer_5x1_constraints.json
Written successfully: ./transfer_5x1_cpp/transfer_5x1.cpp and ./transfer_5x1_cpp/transfer_5x1.dat
Written successfully: ./transfer_5x1_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_5x1_js/transfer_5x1.wat
Written successfully: ./transfer_5x1_js/transfer_5x1.wasm
Everything went okay
template instances: 318
non-linear constraints: 32221
linear constraints: 29094
public inputs: 9
private inputs: 106
public outputs: 0
wires: 61403
labels: 100956
Written successfully: ./transfer_5x2.r1cs
Written successfully: ./transfer_5x2.sym
Constraints written in: ./transfer_5x2_constraints.json
Written successfully: ./transfer_5x2_cpp/transfer_5x2.cpp and ./transfer_5x2_cpp/transfer_5x2.dat
Written successfully: ./transfer_5x2_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_5x2_js/transfer_5x2.wat
Written successfully: ./transfer_5x2_js/transfer_5x2.wasm
Everything went okay
template instances: 324
non-linear constraints: 32647
linear constraints: 29583
public inputs: 10
private inputs: 108
public outputs: 0
wires: 62319
labels: 102355
Written successfully: ./transfer_5x3.r1cs
Written successfully: ./transfer_5x3.sym
Constraints written in: ./transfer_5x3_constraints.json
Written successfully: ./transfer_5x3_cpp/transfer_5x3.cpp and ./transfer_5x3_cpp/transfer_5x3.dat
Written successfully: ./transfer_5x3_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_5x3_js/transfer_5x3.wat
Written successfully: ./transfer_5x3_js/transfer_5x3.wasm
Everything went okay
template instances: 318
non-linear constraints: 33037
linear constraints: 29928
public inputs: 11
private inputs: 110
public outputs: 0
wires: 63055
labels: 103442
Written successfully: ./transfer_5x4.r1cs
Written successfully: ./transfer_5x4.sym
Constraints written in: ./transfer_5x4_constraints.json
Written successfully: ./transfer_5x4_cpp/transfer_5x4.cpp and ./transfer_5x4_cpp/transfer_5x4.dat
Written successfully: ./transfer_5x4_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_5x4_js/transfer_5x4.wat
Written successfully: ./transfer_5x4_js/transfer_5x4.wasm
Everything went okay
template instances: 323
non-linear constraints: 33460
linear constraints: 30415
public inputs: 12
private inputs: 112
public outputs: 0
wires: 63966
labels: 104835
Written successfully: ./transfer_5x5.r1cs
Written successfully: ./transfer_5x5.sym
Constraints written in: ./transfer_5x5_constraints.json
Written successfully: ./transfer_5x5_cpp/transfer_5x5.cpp and ./transfer_5x5_cpp/transfer_5x5.dat
```

```
Written successfully: ./transfer_5x5_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_5x5_js/transfer_5x5.wat
Written successfully: ./transfer_5x5_js/transfer_5x5.wasm
Everything went okay
template instances: 318
non-linear constraints: 36515
linear constraints: 34062
public inputs: 9
private inputs: 123
public outputs: 0
wires: 70681
labels: 114810
Written successfully: ./transfer_6x1.r1cs
Written successfully: ./transfer_6x1.sym
Constraints written in: ./transfer_6x1_constraints.json
Written successfully: ./transfer_6x1_cpp/transfer_6x1.cpp and ./transfer_6x1_cpp/transfer_6x1.dat
Written successfully: ./transfer_6x1_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_6x1_js/transfer_6x1.wat
Written successfully: ./transfer_6x1_js/transfer_6x1.wasm
Everything went okay
template instances: 324
non-linear constraints: 36941
linear constraints: 34551
public inputs: 10
private inputs: 125
public outputs: 0
wires: 71597
labels: 116209
Written successfully: ./transfer_6x2.r1cs
Written successfully: ./transfer_6x2.sym
Constraints written in: ./transfer_6x2_constraints.json
Written successfully: ./transfer_6x2_cpp/transfer_6x2.cpp and ./transfer_6x2_cpp/transfer_6x2.dat
Written successfully: ./transfer_6x2_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_6x2_js/transfer_6x2.wat
Written successfully: ./transfer_6x2_js/transfer_6x2.wasm
Everything went okay
template instances: 318
non-linear constraints: 37331
linear constraints: 34896
public inputs: 11
private inputs: 127
public outputs: 0
wires: 72333
labels: 117296
Written successfully: ./transfer_6x3.r1cs
Written successfully: ./transfer_6x3.sym
Constraints written in: ./transfer_6x3_constraints.json
Written successfully: ./transfer_6x3_cpp/transfer_6x3.cpp and ./transfer_6x3_cpp/transfer_6x3.dat
Written successfully: ./transfer_6x3_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_6x3_js/transfer_6x3.wat
Written successfully: ./transfer_6x3_js/transfer_6x3.wasm
Everything went okay
template instances: 323
non-linear constraints: 37754
linear constraints: 35383
public inputs: 12
private inputs: 129
public outputs: 0
wires: 73244
labels: 118689
Written successfully: ./transfer_6x4.r1cs
Written successfully: ./transfer_6x4.sym
Constraints written in: ./transfer_6x4_constraints.json
Written successfully: ./transfer_6x4_cpp/transfer_6x4.cpp and ./transfer_6x4_cpp/transfer_6x4.dat
Written successfully: ./transfer_6x4_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_6x4_js/transfer_6x4.wat
Written successfully: ./transfer_6x4_js/transfer_6x4.wasm
Everything went okay
template instances: 328
```

```
non-linear constraints: 38177
linear constraints: 35880
public inputs: 13
private inputs: 131
public outputs: 0
wires: 74165
labels: 120102
Written successfully: ./transfer_6x5.r1cs
Written successfully: ./transfer_6x5.sym
Constraints written in: ./transfer_6x5_constraints.json
Written successfully: ./transfer_6x5_cpp/transfer_6x5.cpp and ./transfer_6x5_cpp/transfer_6x5.dat
Written successfully: ./transfer_6x5_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_6x5_js/transfer_6x5.wat
Written successfully: ./transfer_6x5_js/transfer_6x5.wasm
Everything went okay
template instances: 324
non-linear constraints: 41235
linear constraints: 39519
public inputs: 10
private inputs: 142
public outputs: 0
wires: 80875
labels: 130063
Written successfully: ./transfer_7x1.r1cs
Written successfully: ./transfer_7x1.sym
Constraints written in: ./transfer_7x1_constraints.json
Written successfully: ./transfer_7x1_cpp/transfer_7x1.cpp and ./transfer_7x1_cpp/transfer_7x1.dat
Written successfully: ./transfer_7x1_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_7x1_js/transfer_7x1.wat
Written successfully: ./transfer_7x1_js/transfer_7x1.wasm
Everything went okay
template instances: 318
non-linear constraints: 41625
linear constraints: 39864
public inputs: 11
private inputs: 144
public outputs: 0
wires: 81611
labels: 131150
Written successfully: ./transfer_7x2.r1cs
Written successfully: ./transfer_7x2.sym
Constraints written in: ./transfer_7x2_constraints.json
Written successfully: ./transfer_7x2_cpp/transfer_7x2.cpp and ./transfer_7x2_cpp/transfer_7x2.dat
Written successfully: ./transfer_7x2_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_7x2_js/transfer_7x2.wat
Written successfully: ./transfer_7x2_js/transfer_7x2.wasm
Everything went okay
template instances: 323
non-linear constraints: 42048
linear constraints: 40351
public inputs: 12
private inputs: 146
public outputs: 0
wires: 82522
labels: 132543
Written successfully: ./transfer_7x3.r1cs
Written successfully: ./transfer_7x3.sym
Constraints written in: ./transfer_7x3_constraints.json
Written successfully: ./transfer_7x3_cpp/transfer_7x3.cpp and ./transfer_7x3_cpp/transfer_7x3.dat
Written successfully: ./transfer_7x3_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_7x3_js/transfer_7x3.wat
Written successfully: ./transfer_7x3_js/transfer_7x3.wasm
Everything went okay
template instances: 328
non-linear constraints: 42471
linear constraints: 40848
public inputs: 13
private inputs: 148
public outputs: 0
wires: 83443
```

```
labels: 133956
Written successfully: ./transfer_7x4.r1cs
Written successfully: ./transfer_7x4.sym
Constraints written in: ./transfer_7x4_constraints.json
Written successfully: ./transfer_7x4_cpp/transfer_7x4.cpp and ./transfer_7x4_cpp/transfer_7x4.dat
Written successfully: ./transfer_7x4_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_7x4_js/transfer_7x4.wat
Written successfully: ./transfer_7x4_js/transfer_7x4.wasm
Everything went okay
template instances: 318
non-linear constraints: 42849
linear constraints: 41115
public inputs: 14
private inputs: 150
public outputs: 0
wires: 84089
labels: 134879
Written successfully: ./transfer_7x5.r1cs
Written successfully: ./transfer_7x5.sym
Constraints written in: ./transfer_7x5_constraints.json
Written successfully: ./transfer_7x5_cpp/transfer_7x5.cpp and ./transfer_7x5_cpp/transfer_7x5.dat
Written successfully: ./transfer_7x5_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_7x5_js/transfer_7x5.wat
Written successfully: ./transfer_7x5_js/transfer_7x5.wasm
Everything went okay
template instances: 318
non-linear constraints: 45919
linear constraints: 44832
public inputs: 11
private inputs: 161
public outputs: 0
wires: 90889
labels: 145004
Written successfully: ./transfer_8x1.r1cs
Written successfully: ./transfer_8x1.sym
Constraints written in: ./transfer_8x1_constraints.json
Written successfully: ./transfer_8x1_cpp/transfer_8x1.cpp and ./transfer_8x1_cpp/transfer_8x1.dat
Written successfully: ./transfer_8x1_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_8x1_js/transfer_8x1.wat
Written successfully: ./transfer_8x1_js/transfer_8x1.wasm
Everything went okay
template instances: 323
non-linear constraints: 46342
linear constraints: 45319
public inputs: 12
private inputs: 163
public outputs: 0
wires: 91800
labels: 146397
Written successfully: ./transfer_8x2.r1cs
Written successfully: ./transfer_8x2.sym
Constraints written in: ./transfer_8x2_constraints.json
Written successfully: ./transfer_8x2_cpp/transfer_8x2.cpp and ./transfer_8x2_cpp/transfer_8x2.dat
Written successfully: ./transfer_8x2_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_8x2_js/transfer_8x2.wat
Written successfully: ./transfer_8x2_js/transfer_8x2.wasm
Everything went okay
template instances: 328
non-linear constraints: 46765
linear constraints: 45816
public inputs: 13
private inputs: 165
public outputs: 0
wires: 92721
labels: 147810
Written successfully: ./transfer_8x3.r1cs
Written successfully: ./transfer_8x3.sym
Constraints written in: ./transfer_8x3_constraints.json
Written successfully: ./transfer_8x3_cpp/transfer_8x3.cpp and ./transfer_8x3_cpp/transfer_8x3.dat
```

```
Written successfully: ./transfer_8x3_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_8x3_js/transfer_8x3.wat
Written successfully: ./transfer_8x3_js/transfer_8x3.wasm
Everything went okay
template instances: 318
non-linear constraints: 47143
linear constraints: 46083
public inputs: 14
private inputs: 167
public outputs: 0
wires: 93367
labels: 148733
Written successfully: ./transfer_8x4.r1cs
Written successfully: ./transfer_8x4.sym
Constraints written in: ./transfer_8x4_constraints.json
Written successfully: ./transfer_8x4_cpp/transfer_8x4.cpp and ./transfer_8x4_cpp/transfer_8x4.dat
Written successfully: ./transfer_8x4_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_8x4_js/transfer_8x4.wat
Written successfully: ./transfer_8x4_js/transfer_8x4.wasm
Everything went okay
template instances: 322
non-linear constraints: 47563
linear constraints: 46568
public inputs: 15
private inputs: 169
public outputs: 0
wires: 94273
labels: 150120
Written successfully: ./transfer_8x5.r1cs
Written successfully: ./transfer_8x5.sym
Constraints written in: ./transfer_8x5_constraints.json
Written successfully: ./transfer_8x5_cpp/transfer_8x5.cpp and ./transfer_8x5_cpp/transfer_8x5.dat
Written successfully: ./transfer_8x5_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_8x5_js/transfer_8x5.wat
Written successfully: ./transfer_8x5_js/transfer_8x5.wasm
Everything went okay
template instances: 323
non-linear constraints: 50636
linear constraints: 50287
public inputs: 12
private inputs: 180
public outputs: 0
wires: 101078
labels: 160251
Written successfully: ./transfer_9x1.r1cs
Written successfully: ./transfer_9x1.sym
Constraints written in: ./transfer_9x1_constraints.json
Written successfully: ./transfer_9x1_cpp/transfer_9x1.cpp and ./transfer_9x1_cpp/transfer_9x1.dat
Written successfully: ./transfer_9x1_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_9x1_js/transfer_9x1.wat
Written successfully: ./transfer_9x1_js/transfer_9x1.wasm
Everything went okay
template instances: 328
non-linear constraints: 51059
linear constraints: 50784
public inputs: 13
private inputs: 182
public outputs: 0
wires: 101999
labels: 161664
Written successfully: ./transfer_9x2.r1cs
Written successfully: ./transfer_9x2.sym
Constraints written in: ./transfer_9x2_constraints.json
Written successfully: ./transfer_9x2_cpp/transfer_9x2.cpp and ./transfer_9x2_cpp/transfer_9x2.dat
Written successfully: ./transfer_9x2_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_9x2_js/transfer_9x2.wat
Written successfully: ./transfer_9x2_js/transfer_9x2.wasm
Everything went okay
template instances: 318
```

```
non-linear constraints: 51437
linear constraints: 51051
public inputs: 14
private inputs: 184
public outputs: 0
wires: 102645
labels: 162587
Written successfully: ./transfer_9x3.r1cs
Written successfully: ./transfer_9x3.sym
Constraints written in: ./transfer_9x3_constraints.json
Written successfully: ./transfer_9x3_cpp/transfer_9x3.cpp and ./transfer_9x3_cpp/transfer_9x3.dat
Written successfully: ./transfer_9x3_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_9x3_js/transfer_9x3.wat
Written successfully: ./transfer_9x3_js/transfer_9x3.wasm
Everything went okay
template instances: 322
non-linear constraints: 51857
linear constraints: 51536
public inputs: 15
private inputs: 186
public outputs: 0
wires: 103551
labels: 163974
Written successfully: ./transfer_9x4.r1cs
Written successfully: ./transfer_9x4.sym
Constraints written in: ./transfer_9x4_constraints.json
Written successfully: ./transfer_9x4_cpp/transfer_9x4.cpp and ./transfer_9x4_cpp/transfer_9x4.dat
Written successfully: ./transfer_9x4_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_9x4_js/transfer_9x4.wat
Written successfully: ./transfer_9x4_js/transfer_9x4.wasm
Everything went okay
template instances: 326
non-linear constraints: 52277
linear constraints: 52029
public inputs: 16
private inputs: 188
public outputs: 0
wires: 104465
labels: 165377
Written successfully: ./transfer_9x5.r1cs
Written successfully: ./transfer_9x5.sym
Constraints written in: ./transfer_9x5_constraints.json
Written successfully: ./transfer_9x5_cpp/transfer_9x5.cpp and ./transfer_9x5_cpp/transfer_9x5.dat
Written successfully: ./transfer_9x5_cpp/main.cpp, circom.hpp, calcwit.hpp, calcwit.cpp, fr.hpp, fr.cpp, fr.asm and
↪  Makefile
Written successfully: ./transfer_9x5_js/transfer_9x5.wat
Written successfully: ./transfer_9x5_js/transfer_9x5.wasm
Everything went okay
Done
```

8.2 Tests Output

```
> protocol-circuits@2.0.0 test
> mocha --exit --timeout 30000



  Transfer 1x2
     Should verify correctness (2571ms)
     Should verify correctness of nullifier check (690ms)
     Should verify correctness of nullifier check with a dummy note (673ms)
    1) Should verify correctnes of a merkle root


  3 passing (5s)
  1 failing

  1) Transfer 1x2
       Should verify correctnes of a merkle root:
     Error: Error: Assert Failed.
Error in template ForceEqualIfEnabled_74 line: 56
Error in template MerkleProofVerifier_75 line: 39

      at /.../node_modules/circom_tester/wasm/witness_calculator.js:163:27
      at Array.forEach (<anonymous>)
      at WitnessCalculator._doCalculateWitness (node_modules/circom_tester/wasm/witness_calculator.js:138:14)
      at WitnessCalculator.calculateWitness (node_modules/circom_tester/wasm/witness_calculator.js:177:20)
      at WasmTester.calculateWitness (node_modules/circom_tester/wasm/tester.js:113:45)
      at Context.<anonymous> (test/transfer_1x2.test.js:233:39)
```

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.