Security Review Report

NM-0693 D - World ID Nullifier

January 12, 2026

NETHERMIND SECURITY

Contents

1 Executive Summary **2**

2 Audited Files **3**

3 Summary of Issues **4**

4 System Overview **5**

5 Risk Rating Methodology **7**

6 Issues **8**
6.1 [Low] Lack of uniqueness check of party ID's when computing lagrange coefficients . . . 8
6.2 [Low] Non-constant time implementation of field arithmetic leading to timing attack vulnerability . . . 8
6.3 [Low] EdDSA vulnerable to timing attacks . . . 9
6.4 [Low] `their_pk` is not validated during share generation . . . 10
6.5 [Info] Debug Assertions in Poseidon2 permutation . . . 10
6.6 [Info] G1/G2 point deserialization accepts point at infinity . . . 10
6.7 [Info] Some assertions can be made constant . . . 11
6.8 [Info] `combine_proofs` asks for `contributing_parties` . . . 11
6.9 [Info] Nonce can be reused during share generation . . . 12

7 Documentation Evaluation **13**

8 Test Suite Evaluation **14**
8.1 Compilation Output . . . 14
8.1.1 Eddsa BabyJubJub Compilation . . . 14
8.1.2 OPRF Service Compilation . . . 14
8.1.3 Circom-helpers Compilation . . . 14
8.2 Tests Output . . . 14
8.2.1 Poseidon2 tests . . . 14
8.2.2 ark-babyjubjub tests . . . 14
8.2.3 OPRF tests . . . 15

9 About Nethermind **17**

1 Executive Summary

This document presents the audit performed by Nethermind Security for the *World ID OPRF Nullifier* code from Nov. 24, 2025 and Dec. 18, 2025. The World Nullifier Service implements a privacy-preserving Verifiable Oblivious Pseudorandom Function for the World ID ecosystem. This audit focuses on validating the correctness of the cryptographic primitives, secret sharing protocols, and distributed proof generation mechanisms that enable relying parties to generate deterministic, unlinkable nullifiers without compromising user privacy or revealing sensitive credential information to any single party.

The audited code comprises 7790 lines of code written in the Rust language. **The audit was performed using** (a) manual analysis of the codebases and (b) automated analysis tools.

Along this document, we report nine point of attention, where four are classified as `Low` and five as `Informational`. **Fig. 1** summarizes the issues identified in the *codebases*.

This document is organized as follows. Section 2 presents the files in the scope. Section 3 summarizes the issues. Section 4 presents the system overview. Section 5 discusses the risk rating methodology. Section 6 details the issues. Section 7 discusses the documentation provided by the client for this audit. Section 8 concludes the document. Appendix describes the review of the deployment scripts.

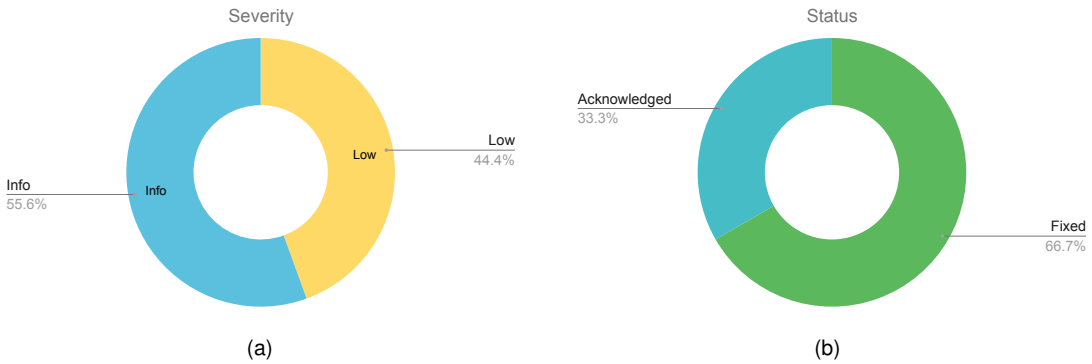


Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (4), **Undetermined** (0), **Informational** (5), **Best Practices** (0). **Distribution of status: Fixed** (6), **Acknowledged** (3), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	Dec 19, 2025
Final Report	January 12, 2026
Response from Client	Regular responses during audit engagement
OPRF Service Audit Commit	`473667cb16b8d8d6a76b79c4bfdf0b07315db07f`
Poseidon2/BabyJubJub Audit Commit	`ddf370922071c584a43aabe884616abd83520e4f`
Circom Helpers Audit Commit	`334b6b3c0607cd38893b6367b6f28d2ddaa1b570`
Final OPRF Service Audit Commit	`5dd490517f774458ce11894174c45bcffc9dabdb`
Final Poseidon2/BabyJubJub Audit Commit	`1740f6039ce7a43a00748806d4ed232d70207c02`
Final Circom Helpers Audit Commit	`334b6b3c0607cd38893b6367b6f28d2ddaa1b570`
Documentation	OPRF paper
Documentation Assessment	High
Test Assessment	High

Summary of Engagement

Dates	Methods	Consultants Engaged	Level of Effort
Nov. 24-Dec. 18, 2025	Manual	4	16-person weeks

2 Audited Files

	Off-chain code	LoC	Comments	Ratio	Blank	Total
1	oprf-core/src/keygen.rs	340	133	39.12%	64	537
2	oprf-core/src/ddlog_equality/shamir.rs	255	66	25.88%	37	358
3	oprf-core/benches/oprf.rs	244	2	0.82%	25	271
4	oprf-core/src/ddlog_equality.rs	161	48	29.81%	18	227
5	oprf-core/src/oprf.rs	147	30	20.41%	31	208
6	oprf-core/src/dlog_equality.rs	112	27	24.11%	18	157
7	oprf-core/src/shamir.rs	86	44	51.16%	10	140
8	oprf-core/src/lib.rs	6	7	116.67%	0	13
9	oprf-types/src/crypto.rs	166	66	39.76%	33	265
10	oprf-types/src/api/v1.rs	71	35	49.30%	10	116
11	oprf-types/src/chain.rs	56	48	85.71%	11	115
12	oprf-types/src/lib.rs	50	21	42.00%	13	84
13	oprf-types/src/api.rs	1	5	500.00%	0	6
19	ark-babyjubjub/src/lib.rs	15	15	100.00%	2	32
20	ark-babyjubjub/src/constraints/curves.rs	7	1	14.29%	3	11
21	ark-babyjubjub/src/constraints/fields.rs	6	1	16.67%	2	9
22	ark-babyjubjub/src/fields/fr.rs	6	0	0.00%	1	7
23	ark-serde-compat/src/lib.rs	523	301	57.55%	63	887
24	ark-serde-compat/src/babyjubjub.rs	214	64	29.91%	27	305
25	ark-serde-compat/src/impl_macro.rs	238	0	0.00%	23	261
26	circom-types/src/groth16/zkey.rs	592	77	13.01%	57	726
27	circom-types/src/groth16/verification_key.rs	213	13	6.10%	17	243
28	circom-types/src/groth16/zkey_to_ark.rs	200	9	4.50%	16	225
29	circom-types/src/groth16/proof.rs	105	9	8.57%	9	123
30	circom-types/src/groth16/public_input.rs	64	4	6.25%	10	78
31	circom-types/src/groth16/mod.rs	20	1	5.00%	1	22
32	circom-types/src/plonk/zkey.rs	305	87	28.52%	23	415
33	circom-types/src/plonk/verification_key.rs	204	18	8.82%	15	237
34	circom-types/src/plonk/proof.rs	189	21	11.11%	14	224
35	circom-types/src/plonk/mod.rs	12	1	8.33%	2	15
36	circom-types/src/bin/convert-zkey-to-ark.rs	73	8	10.96%	14	95
37	circom-types/src/r1cs.rs	250	50	20.00%	40	340
38	circom-types/src/traits.rs	189	27	14.29%	36	252
39	circom-types/src/witness.rs	117	15	12.82%	17	149
40	circom-types/src/binfile.rs	83	6	7.23%	12	101
41	circom-types/src/lib.rs	160	17	10.62%	21	198
42	groth16/src/lib.rs	195	28	14.36%	20	243
43	groth16/src/reduction.rs	241	16	6.64%	17	274
44	groth16-material/src/circom.rs	284	137	48.24%	38	459
45	groth16-material/src/lib.rs	12	9	75.00%	2	23
1	poseidon2/src/bn254/t16.rs	328	24	7.32%	10	362
2	poseidon2/src/bn254/t12.rs	276	24	8.70%	12	312
3	poseidon2/src/bn254/t8.rs	219	24	10.96%	12	255
4	poseidon2/src/bn254/t4.rs	211	26	12.32%	13	250
5	poseidon2/src/perm.rs	168	36	21.43%	18	222
6	poseidon2/src/bn254/t3.rs	153	24	15.69%	12	189
7	poseidon2/src/bn254/t2.rs	137	22	16.06%	13	172
8	poseidon2/src/bn254.rs	60	18	30.00%	9	87
9	poseidon2/benches/poseidon2.rs	22	0	0.00%	4	26
10	poseidon2/src/lib.rs	4	17	425.00%	1	22
	Total	**7790**	**1682**	**21.59%**	**876**	**10348**

3 Summary of Issues

	Finding	Severity	Update
1	Lack of uniqueness check of party ID's when computing lagrange coefficients	Low	Fixed
2	Non-constant time implementation of field arithmetic leading to timing attack vulnerability	Low	Acknowledged
3	EdDSA vulnerable to timing attacks	Low	Fixed
4	`their_pk` is not validated during share generation	Low	Fixed
5	Debug Assertions in Poseidon2 permutation	Info	Acknowledged
6	G1/G2 point deserialization accepts point at infinity	Info	Acknowledged
7	Some assertions can be made constant	Info	Fixed
8	`combine_proofs` asks for `contributing_parties`	Info	Fixed
9	Nonce can be reused during share generation	Info	Fixed

4 System Overview

This system implements an Oblivious PseudoRandom Function-based nullifier construction designed for threshold security and zero-knowledge compatibility. The deterministically derive unlinkable nullifiers from private inputs, while allowing a distributed set of servers to jointly evaluate the OPRF without revealing the underlying secret key or the client input.

The nullifier construction relies on SNARK-friendly cryptography to ensure efficient verification inside zero-knowledge circuits. The poseidon2 crate implements the Poseidon2 permutation over the BN254 scalar field, Poseidon2 is used to hash OPRF transcripts, derive nullifiers, and construct Poseidon-based Merkle trees used to commit to nullifier sets or protocol state.

BabyJubJub serves as the elliptic-curve group for the OPRF. The system uses arkworks-compatible BabyJubJub implementations for group operations and constraint-system support, and EdDSA-BabyJubJub for signing and authentication of protocol artifacts when required.

The oprf-core crate implements the OPRF-based nullifier protocol. To enforce correctness in a distributed setting, oprf-core implements discrete-log equality proofs, including double DLog equality constructions combined with Shamir secret sharing. These mechanisms ensure that each server's OPRF contribution is consistent with its secret share without revealing the share itself.

As can be seen in the diagram (Fig.2), the user hashes the identity commitment, relaying party ID and the action it wants to perform(e.g. a vote) to obtain a query. this query is blinded over the BabyJubJub elliptic curve with a random scalar β. It then send the blinded query to all servers, to compute an OPRF response ($C_i = B \cdot x_i$), and generates random commitments for the proof. These commitments are then aggregated through Lagrange interpolation reconstructing $B \cdot x$.

Over round 2, each server computes a challenge hash and generate proof shares, these shares are collected by the client and combines it into a single Chaum-Pedersen proof for which it can verify againt the public key. If everything works out, the client unblinds by multiplying β^{-1}, this is hashed one final time to produce the nullifier.

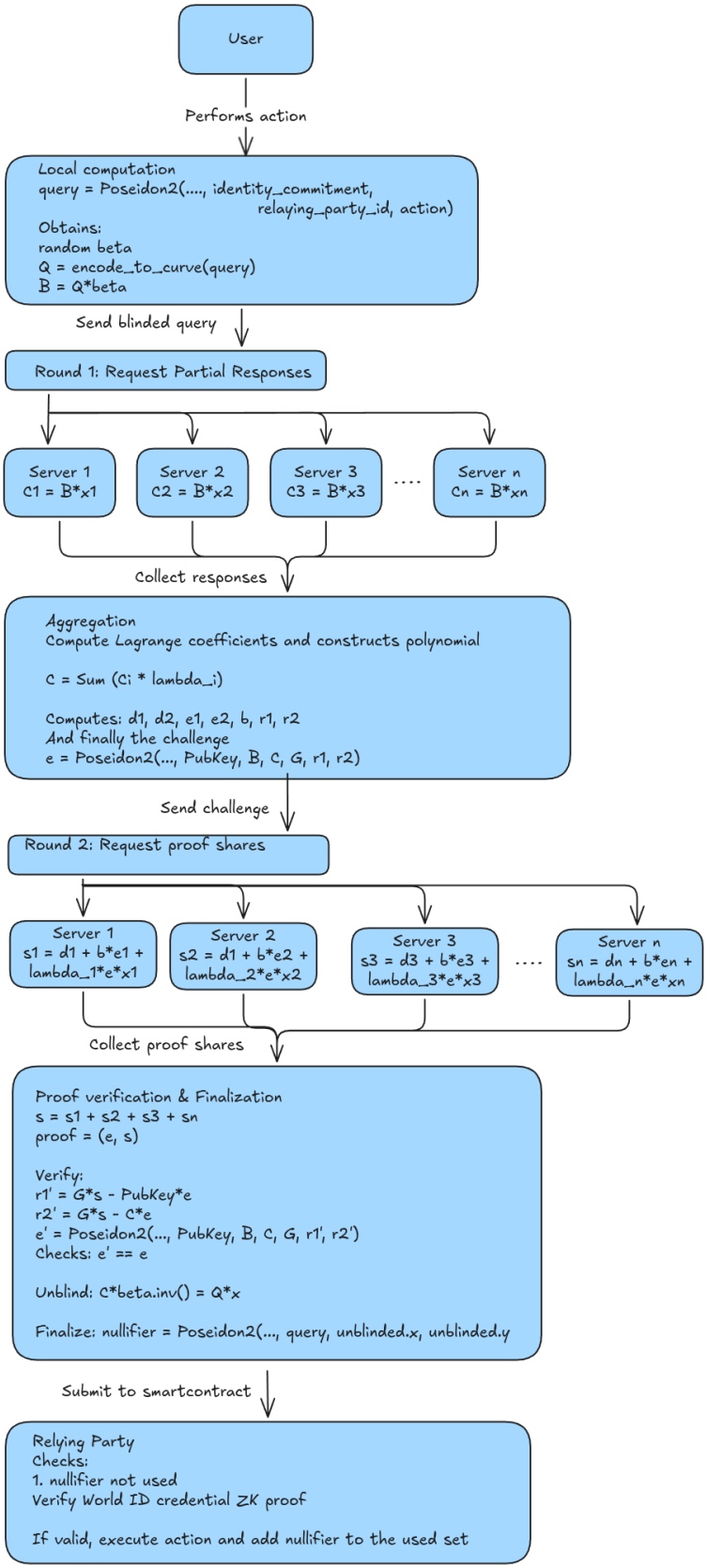


Fig. 2: OPRF core flow

5 Risk Rating Methodology

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

Likelihood measures how likely the finding is to be uncovered and exploited by an attacker. This factor will be one of the following values:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6 Issues

6.1 [Low] Lack of uniqueness check of party ID's when computing lagrange coefficients

File(s): oprf-core/src/ddlog_equality/shamir.rs

Description: In `combine_commitments(...)` it is required to compute the lagrange coefficients for the provided contributing parties. However, there is no check that the contributing parties are unique. This means it is possible to use duplicate party IDs and commitments. Maliciously this can be used to inflate the number of shares and produce an incorrect secret from the secret sharing.

Recommendation(s): Before computing the lagrange coefficients for the contributing parties, check that:

```
set(contributing_parties).len() == contributing_parties.len();
```

This will ensure only unique party IDs are being used in the commitment combination.

Status: Fixed

Update from the client: Fixed in PR #339.

6.2 [Low] Non-constant time implementation of field arithmetic leading to timing attack vulnerability

File(s): oprf-core/src/ddlog_equality/shamir.rs, oprf-core/src/oprf.rs, oprf-core/src/shamir.rs, oprf-core/src/dlog_equality.rs

Description: The underlying field arithmetic used for the following functions uses branching conditions that branch based on the secret values used. Here we provide a list of key functions we believe are vulnerable to such attacks and why:

1. `shamir.rs/dh_key_derivation(...)` performs multiplication of a scalarfield element and an affine element performing group multiplication for affine coordinates using the double-and-add method;
2. `oprf.rs/unblind_response(...)` is used to unblind the server response using the prepared blinding factor performed with the double-and-add method;
3. `dlog_equality.rs/partial_commitments(...)` performs multiplication over the secret share `x_share` using the double-and-add method;
4. `shamir.rs/single_lagrange_from_coeff(...)` and `shamir.rs/evaluate_poly(...)` performs field multiplication for a prime field, the `ScalarField` nor `BaseField` types do not implement constant-time multiplication;

Recommendation(s): Implement constant time field arithmetic.

Status: Acknowledged

Update from the client: Most low-level field arithmetic operations in ark-ff are constant time, with a few exceptions like the subtraction that does a branching on potential overflows. We accept this for now and will re-evaluate in the future if we want to address this further by changing out the underlying field arithmetic completely.

6.3 [Low] EdDSA vulnerable to timing attacks

File(s): eddsa-babyjubjub/src/lib.rs

Description: Timing attacks are side channel attacks that enable attackers to infer values of secret keys by noticing changes in the time it takes to perform cryptographic operations. As stated by RFC 8032, such attacks stem from the secret key influencing the set of instructions and memory accesses executed by the code. It was found that the implementation of EdDSA is vulnerable to such attacks after running `dudect` to measure the signature generation speed against a fixed key and a random key. Specifically, the code run was with 1 million samples:

```rust
/// Differential time analysis of signing, fixed key & msg vs. fixed msg & random key
fn eddsa_sign_ct_vary_sk(runner: &mut CtRunner, rng: &mut BenchRng) {
    let msg = BaseField::from(0u64);

    let mut sk_rng = rand::thread_rng();
    let mut keys = Vec::with_capacity(N_SAMPLES);
    let mut classes = Vec::with_capacity(N_SAMPLES);

    for _ in 0..N_SAMPLES {
        if rng.next_u32() & 1 == 0 {
            // Left: fixed key
            keys.push(EdDSAPrivateKey::from_bytes([0x13; 32]));
            classes.push(Class::Left);
        } else {
            // Right: random key
            keys.push(EdDSAPrivateKey::random(&mut sk_rng));
            classes.push(Class::Right);
        }
    }

    for (class, sk) in classes.into_iter().zip(keys.into_iter()) {
        runner.run_one(class, || {
            let sig = sk.sign(msg);
            core::hint::black_box(sig);
        });
    }
}
```

It was found that this issue stems from the branching condition found in the scalar multiplication in `ark-ec/src/twisted_edwards/mod.rs` which implements affine multiplication using the double-and-add method:

```rust
/// Default implementation of group multiplication for affine coordinates
fn mul_affine(base: &Affine<Self>, scalar: &[u64]) -> Projective<Self> {
    let mut res = Projective::<Self>::zero();
    for b in ark_ff::BitIteratorBE::without_leading_zeros(scalar) {
        res.double_in_place();
        if b {
            res += base
        }
    }

    res
}
```

The attack is possible here as an attacker can measure how many times a value is added and how many times it is multiplied. This is because the time to compute a multiplication and the time to complete an addition is different, allowing the attacker to infer information about the key.

Recommendation(s): We recommend using a constant time implementation of the fields to prevent this attack. However, as far as we understand, there does not exist a constant time implementation of the BabyJubJub curve enabling the usage of constant time curve operations. Arkworks does not implement constant time field operations. Thus, the suggestion is to reimplement a rust crate like Sylow, which uses constant time field operations for BN254, for BJJ.

For Arkworks, a solution could be to implement constant time scalar multiplication to bypass this through the Montgomery ladder method. Various solutions are provided in the paper Elliptic Curves and Side-Channel Analysis.

Status: Fixed

Update from the client: Fixed in PR #11

6.4 [Low] `their_pk` is not validated during share generation

File(s): `oprf-core/src/keygen.rs`

Description: `gen_share` public function accepts `their_pk` as an argument, but it does not validate it being on-curve.

There is no code calling this function in the audit scope (outside of local tests). There is a validation performed if `their_pk` is obtained specifically via checked versions of `deserialize_*` methods of `CanonicalDeserialize` trait.

If obtained by other means though, there's no safeguard against `their_pk` being off-curve, thus making private key vulnerable to invalid curve attack.

Recommendation(s): We recommend checking `their_pk` being on curve and in the correct subgroup inside the `gen_share` function itself.

Status: Fixed

Update from the client: Fixed in PR #339.

6.5 [Info] Debug Assertions in Poseidon2 permutation

File(s): `eddsa-babyjubjub/poseidon2/src/perm.rs`

Description: The Poseidon2 permutation file contains some debug assertions. These are not part of the release build and will enable users to instantiate Poseidon2 internal matrices with different parameters than the recommended.

```
/// The matrix multiplication in the internal rounds of the Poseidon2 permutation.
    pub fn matmul_internal(&self, input: &mut [F; T]) {
        match T {
            2 => {
                // Matrix [[2, 1], [1, 3]]
                debug_assert_eq!(self.mat_internal_diag_m_1[0], F::one());
                debug_assert_eq!(self.mat_internal_diag_m_1[1], F::from(2u64));
                let sum = input[0] + input[1];
                input[0] += &sum;
                input[1].double_in_place();
                input[1] += sum;
            }
            3 => {
                // Matrix [[2, 1, 1], [1, 2, 1], [1, 1, 3]]
                debug_assert_eq!(self.mat_internal_diag_m_1[0], F::one());
                debug_assert_eq!(self.mat_internal_diag_m_1[1], F::one());
                debug_assert_eq!(self.mat_internal_diag_m_1[2], F::from(2u64));
            [...]
```

Status: Acknowledged

Update from the client: They acknowledged they are aware of the issue. But it is a minor one, as the library is intended for internal use. Additionally, they would like to explore verification at compile time to avoid incurring additional runtime costs for the assertions. The issue is presented in an informational manner to help keep track of the problem.

6.6 [Info] G1/G2 point deserialization accepts point at infinity

File(s): `ark-serde-compat/src/lib.rs`

Description: `serde` deserialization of G1 points via intended `[serde(deserialize_with = "ark_serde_compat::CURVE::deserialize_g1")]` boils down to `G1Visitor` called with `const CHECK == true`. This `CHECK` enforces the deserialized point to be on curve and in the subgroup of the correct order, otherwise deserialization will error out. However, it does explicitly allow point at infinity (a.k.a zero point).

Using a point at infinity in certain context may lead to the soundness property of a proof system being broken.

All of the above applies to G2 points as well.

This deserialization routine is used e.g. in `VerificationKey` of `circom_types::plonk` and `circom_types::groth16`, where no additional validation is being done.

Recommendation(s): Care should be taken when using this deserialization, with additional validation performed accordingly on deserialized entities where this might impact security guarantees.

Status: Acknowledged

Update from the client: They say that the validation is done on the contract side, so PaI won't reach the deserialization.

6.7 [Info] Some assertions can be made constant

File(s): eddsa-babyjubjub/poseidon2/src/perm.rs

Description: The Poseidon2 permutation file contains some assertions that can be updated to be done at compile time.

```
/// Creates a new instance of the Poseidon2 permutation with given parameters
    pub const fn new(
        mat_internal_diag_m_1: [F; T],
        round_constants_external: [[F; T]; ROUNDS_F],
        round_constants_internal: [F; ROUNDS_P],
    ) -> Self {
        assert!(T == 2 || T == 3 || ((T <= 24) && (T % 4 == 0)));
        assert!(D % 2 == 1);
        assert!(ROUNDS_F % 2 == 0);

        Self {
            mat_internal_diag_m_1,
            round_constants_external,
            round_constants_internal,
        }
    }
```

These assertions only involved information known at compile time. They can be defined within a const block (stabilized since Rust 1.79)

```
pub const fn new(
    mat_internal_diag_m_1: [F; T],
    round_constants_external: [[F; T]; ROUNDS_F],
    round_constants_internal: [F; ROUNDS_P],
) -> Self {
    const {
            assert!(T == 2 || T == 3 || ((T <= 24) && (T % 4 == 0)));
            assert!(D % 2 == 1);
                        assert!(ROUNDS_F % 2 == 0);
                }
          // [...]
}
```

Alternatively, use static_assertions.

This ensures the checks are done at compile time instead of runtime:

- Avoiding wrong configurations for Poseidon2;
- Transforms a potential panic into a compilation error;

Status: Fixed

Update from the client: Fixed in PR #10

6.8 [Info] `combine_proofs` asks for `contributing_parties`

File(s): oprf-core/src/ddlog_equality/shamir.rs

Description: One argument to `DLogEqualityCommitments::combine_proofs` and `DLogCommitmentsShamir::combine_proofs` methods is `contributing_parties` listing parties that contributed to proof generation. However, `DLogEqualityCommitments` (and by extension, its wrapper `DLogCommitmentsShamir`) already contains `contributing_parties` vector.

Having `contributing_parties` argument is unnecessary and introduces risk of misusing it.

Recommendation(s): We recommend removing `contributing_parties` argument from `combine_proofs` and pass in.

Status: Fixed

Update from the client: Fixed in PR #339.

6.9 [Info] Nonce can be reused during share generation

File(s): `oprf-core/src/keygen.rs`

Description: `gen_share` public function accepts nonce as an argument. While there's no code calling this function in the audit scope (outside of local tests), it opens the opportunity for it to be misused by passing the same nonce multiple times. If allowed, it might make party's secret key vulnerable.

Recommendation(s): We recommend passing in the secure RNG and generating nonce in the `gen_share` function itself.

Status: Fixed

Update from the client: Fixed in PR #339.

7 Documentation Evaluation

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

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

Remarks about the World nullifier service

The **World** team thoroughly addressed all questions and concerns raised by the **Nethermind Security** team, providing valuable insights and a comprehensive understanding of the project's technical aspects.

8 Test Suite Evaluation

Initially tests were failing due to broken dependencies. This was fixed here

8.1 Compilation Output

8.1.1 Eddsa BabyJubJub Compilation

```
cargo build
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.03s
```

8.1.2 OPRF Service Compilation

```
cargo build
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.19s
warning: the following packages contain code that will be rejected by a future version of Rust: russh v0.54.5
note: to see what the problems were, use the option `--future-incompat-report`, or run `cargo report
 →  future-incompatibilities --id 3`
```

8.1.3 Circom-helpers Compilation

```
cargo build
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.10s
```

8.2 Tests Output

8.2.1 Poseidon2 tests

```
cargo test
    Finished `test` profile [unoptimized + debuginfo] target(s) in 0.03s
     Running unittests src/lib.rs (~/eddsa-babyjubjub/target/debug/deps/taceo_poseidon2-c9098a60311cc71c)

running 13 tests
test bn254::t2::tests::poseidon2_bn254_t2_kat1 ... ok
test bn254::t3::tests::poseidon2_bn254_t3_kat1 ... ok
test bn254::t4::tests::poseidon2_bn254_t4_kat1 ... ok
test bn254::t16::tests::poseidon2_bn254_t16_kat1 ... ok
test bn254::t4::tests::poseidon2_bn254_t4_kat2 ... ok
test bn254::t12::tests::poseidon2_bn254_t12_kat1 ... ok
test bn254::t8::tests::poseidon2_bn254_t8_kat1 ... ok
test bn254::t2::tests::poseidon2_bn254_t2_consistent_perm ... ok
test bn254::t3::tests::poseidon2_bn254_t3_consistent_perm ... ok
test bn254::t4::tests::poseidon2_bn254_t4_consistent_perm ... ok
test bn254::t8::tests::poseidon2_bn254_t8_consistent_perm ... ok
test bn254::t16::tests::poseidon2_bn254_t16_consistent_perm ... ok
test bn254::t12::tests::poseidon2_bn254_t12_consistent_perm ... ok

test result: ok. 13 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.01s

   Doc-tests taceo_poseidon2

running 2 tests
test poseidon2/src/bn254.rs - bn254 (line 12) ... ignored
test poseidon2/src/lib.rs - (line 12) ... ignored

test result: ok. 0 passed; 0 failed; 2 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

8.2.2 ark-babyjubjub tests

```
cargo test
    Finished `test` profile [unoptimized + debuginfo] target(s) in 0.02s
     Running unittests src/lib.rs (~/eddsa-babyjubjub/target/debug/deps/taceo_ark_babyjubjub-8cf5e12d0d642b54)

running 52 tests
test curves::tests::te::test_montgomery_conversion_test ... ok
```

```
test fields::tests::fq::test_montgomery_config ... ok
test fields::tests::fq::test_fft ... ok
test fields::tests::fq::test_constants ... ok
test fields::tests::fq::test_sum_of_products_edge_case ... ok
test curves::tests::te::test_te_properties ... ok
test fields::tests::fq::test_mul_by_base_field_elem ... ok
test fields::tests::fr::test_constants ... ok
test fields::tests::fr::test_fft ... ok
test fields::tests::fq::test_serialization ... ok
test fields::tests::fr::test_montgomery_config ... ok
test fields::tests::fr::test_mul_by_base_field_elem ... ok
test fields::tests::fq::test_frobenius ... ok
test fields::tests::fr::test_frobenius ... ok
test fields::tests::fr::test_serialization ... ok
test fields::tests::fq::test_mul_properties ... ok
test fields::tests::fr::test_mul_properties ... ok
test fields::tests::fr::test_sum_of_products_edge_case ... ok
test fields::tests::fq::test_pow ... ok
test fields::tests::test_fq_add ... ok
test fields::tests::test_fq_add_one ... ok
test fields::tests::test_fq_div ... ok
test fields::tests::test_fq_double_in_place_thrice ... ok
test fields::tests::test_fq_from_str ... ok
test fields::tests::test_fq_generate_random_ed_on_bn254_point ... ok
test fields::tests::test_fq_legendre ... ok
test fields::tests::test_fq_mul ... ok
test fields::tests::test_fq_square_in_place ... ok
test fields::tests::test_fq_sub ... ok
test fields::tests::test_fq_triple_mul ... ok
test fields::tests::test_fr_add ... ok
test fields::tests::test_fr_from_str ... ok
test fields::tests::test_fr_mul ... ok
test fields::tests::fq::test_sqrt ... ok
test curves::tests::te::test_sub_properties ... ok
test curves::tests::te::test_mixed_addition ... ok
test fields::tests::fr::test_pow ... ok
test fields::tests::fr::test_sqrt ... ok
test curves::tests::te::test_chunked_pippenger ... ok
test curves::tests::te::test_add_properties ... ok
test curves::tests::te::test_hashmap_pippenger ... ok
test fields::tests::fq::test_sum_of_products_tests ... ok
test fields::tests::fr::test_sum_of_products_tests ... ok
test curves::tests::te::test_var_base_msm ... ok
test curves::tests::te::test_serialization ... ok
test curves::tests::te::test_cofactor_ops ... ok
test curves::tests::te::test_affine_conversion ... ok
test fields::tests::fq::test_sub_properties ... ok
test fields::tests::fr::test_sub_properties ... ok
test fields::tests::fq::test_add_properties ... ok
test fields::tests::fr::test_add_properties ... ok
test curves::tests::te::test_mul_properties ... ok

test result: ok. 52 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 36.53s

   Doc-tests taceo_ark_babyjubjub

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

8.2.3 OPRF tests

```
cargo test
    Finished `test` profile [unoptimized + debuginfo] target(s) in 0.10s
     Running unittests src/lib.rs
      ↪  (~/nullifier-oracle-service/target/debug/deps/taceo_oprf_core-ab9ebd41115a6c77)

running 16 tests
test oprf::mappings::tests::test_ct_is_zero ... ok
test oprf::mappings::tests::test_map_to_curve_twisted_edwards ... ok
test oprf::mappings::tests::test_encode_to_curve ... ok
test oprf::mappings::tests::test_hash_to_curve ... ok
test oprf::mappings::tests::test_inv0 ... ok
test oprf::tests::test_oprf_determinism ... ok
```

```
test oprf::mappings::tests::test_ct_is_square ... ok
test dlog_equality::tests::test_dlog_equality ... ok
test oprf::mappings::tests::test_map_to_curve_twisted_edwards_rand ... ok
test keygen::tests::test_distributed_keygen_3_1 ... ok
test ddlog_equality::shamir::tests::test_distributed_dlog_equality_shamir_3_1 ... ok
test keygen::tests::test_reshare_3_1 ... ok
test oprf::tests::test_oprf_with_proof ... ok
test ddlog_equality::shamir::tests::test_distributed_dlog_equality_shamir_31_15 ... ok
test keygen::tests::test_distributed_keygen_31_15 ... ok
test keygen::tests::test_reshare_31_15 ... ok

test result: ok. 16 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 2.59s

   Doc-tests taceo_oprf_core

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

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.