Least Authority
PRIVACY MATTERS

OPRF Circom Circuits
Security Audit Report

TACEO

Combined Final Audit Report: 26 January 2026

Table of Contents

Overview

Background

TACEO has requested Least Authority perform a security audit of OPRF circom circuits intended for deployment in the upgraded version of the network, as well as a follow-up audit of the changes to those circuits since the previous audit.

Audit 1

Project Dates

- **October 27, 2025 - November 07, 2025:** Initial Code Review *(Completed)*
- **November 10, 2025:** Delivery of Initial Audit Report *(Completed)*
- **November 10, 2025:** Verification Review *(Completed)*
- **November 10, 2025:** Delivery of Final Audit Report *(Completed)*
- **January 26, 2026** : Delivery of the Combined Report *(Completed)*

Review Team

- Jasper Hepp, Security / Cryptography Researcher and Engineer
- Mirco Richter, Cryptography Researcher and Engineer
- Burak Atasoy, Project Manager
- Jessy Bissal, Technical Editor

Audit 2

Project Dates

- **January 12, 2026 - January 20, 2026:** Initial Code Review *(Completed)*
- **January 20, 2026:** Delivery of Initial Audit Report *(Completed)*
- **January 20, 2026:** Verification Review *(Completed)*
- **January 21, 2026**: Delivery of Final Audit Report *(Completed)*
- **January 26, 2026:** Delivery of the Combined Report *(Completed)*

Review Team

- Jasper Hepp, Security / Cryptography Researcher and Engineer
- Burak Atasoy, Project Manager
- Jessy Bissal, Technical Editor

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of TACEO's OPRF circom circuits and the changes since the previous audit, followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repositories are considered in scope for the review:

- Circuits: https://github.com/TaceoLabs/nullifier-oracle-service/tree/main/circom/main
  - `oprf_nullifier.circom`
  - `oprf_query.circom`

  - `oprf_rpid_query.circom`
  - `keygen.circom`
  - `keygen_single_party.circom`
  - `verify_dlog.circom`
  - `eddsaposeidon2.circom`
  - `binary_merkle_root.circom`
  - `encode_to_curve_babyjj.circom`
  - `inverse_or_zero.circom`
  - `quadratic_residue.circom`
  - `sgn0.circom`
  - `babyjubjub.circom`
  - `correct_sub_group.circom`
  - `poseidon2.circom`
  - `poseidon2_constants.circom`

Specifically, we examined the following Git revision for our initial review:

- `2d5ff3589b497ede496dc1bc9acc259dd3397d50`

We also reviewed the code changes between the previous audit and the current version, specifically:

- https://github.com/TaceoLabs/oprf-service/compare/2d5ff35...5793aa7

For the review, these repositories were cloned for use during the audit and for reference in this report:

- https://github.com/LeastAuthority/TaceoLabs-nullifier-oracle-service
- https://github.com/LeastAuthority/TaceoLabs-nullifier-oracle-service/tree/audit2

For the verification, we examined the following Git revision and PR:

- `a10f5b7111c4155cd65634d0e9747aa9fa65e379`
- https://github.com/TaceoLabs/oprf-service/pull/371

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- Whitepaper: *A Nullifier Protocol based on a Verifiable, Threshold OPRF*: https://github.com/TaceoLabs/oprf-service/blob/main/docs/oprf.pdf
- Website: https://taceo.io

In addition, this audit report references the following documents:

- JP. Aumasson, D. Khovratovich, B. Mennink, and P. Quine, "SAFE: Sponge API for Field Elements." *IACR Cryptology ePrint Archive,* 2023, [AKM+23]
- S. S. Chow, C. Ma, and J. Weng, "Zero-Knowledge Argument for Simultaneous Discrete Logarithms." *ACM Digital Library,* 2012, [CMW12]
- L. Grassi, D. Khovratovich, and M. Schofnegger, "Poseidon2: A Faster Version of the Poseidon Hash Function." *IACR Cryptology ePrint Archive*, 2023, [GKS23]

- RFC 9380 | Hashing to Elliptic Curves: https://www.rfc-editor.org/rfc/rfc9380.html
- `circom` 2 Documentation: https://docs.circom.io/circom-language/tags

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Whether any constraints are missing in the circuits;
- Vulnerabilities within each component and whether the interaction between the components is secure;
- Whether requests are passed correctly to the network core;
- Key management, including secure private key storage and management of encryption and signing keys;
- Denial of Service (DoS) and other security exploits that would impact the intended use or disrupt the execution;
- Protection against malicious attacks and other ways to exploit;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

The Nullifier Oracle Service implements publicly verifiable, privacy-preserving nullifiers through a verifiable, threshold OPRF (Oblivious Pseudorandom Function). The `circom` subdirectory provides the circuits that include the prover logic for the OPRF protocol (OPRF Nullifier Specification), including the client and server side. Reusable primitives (Poseidon2, `BabyJubJub` operations, Merkle tree, encode-to-curve, DLEQ verifier) and OPRF key generation circuits support the protocol's threshold setup and proof composition.

System Design

OPRF Circuits

Our team examined the design of TACEO's `circom` circuits and found that security has generally been taken into consideration. The implementation adheres to Circom best practices and is based on the OPRF Nullifier Specification. Below, we provide additional details on the system's design and security considerations.

Across all reviewed modules, we first verified essential constraints such as booleanity, range bounds, subgroup membership, division-by-zero guards, and logic invariants, applying this uniformly across all files mentioned.

We subsequently audited the OPRF-based nullifier and query logic against the OPRF Nullifier Specification and evaluated `verify_dlog` as a noninteractive Chaum-Pedersen proof against [CMW12]. For signatures and hashing, we compared `eddsaposeidon2.circom` and `poseidon2.circom` to `EdDSA-on-BabyJubJub` best practices and the Poseidon2 specification [GKS23]. For curve arithmetic and subgroup implementation, we reviewed `babyjubjub.circom` and `correct_sub_group.circom` against the BabyJubJub requirements and determined that the EscalarMulFixScalar template misclassifies the non-identity 2-torsion point (Issue C). For hashing to curve, we examined the Elligator2 implementation in encode_to_curve_babyjj.circom and the sgn0 function against RFC-9380, identifying that the HashToField template omits the required domain separator (Issue F). In addition, we compared binary_merkle_root.circom to standard binary-Merkle constructions and audited quadratic_residue.circom and inverse_or_zero.circom against field-arithmetic best practices, noting that the IsQuadraticResidueOrZero template omits a required constraint (Issue D).

We reviewed the key generation protocol for the Shamir-shared OPRF key in circom/oprf_keys and compared it against the protocol described in the OPRF Nullifier Specification (Appendix B, Proposal 2). We did not identify any issues in the template KeyGen, which is the variant used by the currently deployed system. However, we identified three issues in the alternative template for single key generation, KeyGenSinglePartyVar (Issue A, Issue B, Issue E). The code does not yet implement the reshare protocol described in the OPRF Nullifier Specification. We also observed that the circuits for single key generation do not constrain the indices to be nonzero. The TACEO team stated that this condition is enforced elsewhere in the system and that, as a result, the system should not be susceptible to zero-share attacks.

We reviewed the template AuthenticatedEncryption in circom/client_side_proofs/oprf_delegate.circom and compared it against the specification (Algorithm 7 in [AKM+23]). The template implements a hard-wired setup and functions only for fixed parameters. Therefore, the code omits the io_count check in FINISH, which we consider acceptable for this configuration. The implementation otherwise closely follows the specification, and we did not identify any deviations or missing constraints.

Follow-up Audit

Our team examined the changes to TACEO's circom circuits since the previous audit and found that security has generally been taken into consideration. The implementation is based on the OPRF Nullifier Specification. Below, we provide additional details on the changes and security considerations.

The major changes between this audit and the previous audit are contained in the following three PRs. First, PR 341 reimplements the BabyJubJub subgroup check as a fixed-scalar multiplication in Twisted Edwards form using a double-and-add ladder, eliminating the prior Montgomery-form path (and its mapping/addition edge cases). Second, PR 308 removes the credential signature check from the query proof and adds a further check on genesis_issued_at. Third, PR 346 corrects a typographical error in escalarmulfix.circom. During our review of these changes, we did not identify any security issues.

Code Quality

We performed a manual review of the repositories in scope and found the code to be well organized, of high quality, and closely aligned with development best practices. However, we noted a few areas where minor improvements could be implemented (Suggestion 1, Suggestion 2, and Suggestion 3).

Tests

In the first audit, the project was noted to include test coverage; however, tests were out of scope, and coverage was not assessed for sufficiency. In the second audit, which focused on changes since the first audit, our team assessed the unit tests in circom/tests and found coverage to be sufficient.

Documentation and Code Comments

The project documentation provided by the TACEO team, including the accompanying OPRF Nullifier Specification, clearly outlines the project's purpose and sufficiently describes the system's intended functionality. The codebase also includes descriptive comments, which aid in understanding the intended behavior of the relevant components.

Scope

In the first audit, the scope of this review included the contents of the `circom` folder, excluding `circomlib` and the file `circom/client_side_proofs/oprf_delegate.circom`, except for the template `AuthenticatedEncryption`.

We note that several important checks are intentionally omitted from the circuit to minimize the number of constraints and are instead expected to be enforced by the verifier. Because the verifier was not included in this review, the code was audited under the assumption that these checks are correctly performed by the verifier. In particular, the following checks must be executed by the verifier (in addition to those described in the whitepaper as part of the protocol):

- Here the `current_time_stamp` is not range checked;
- Here and in all other circuits, the public key `oprf_pk` must be a valid BabyJubJub point in the correct subgroup;
- Here the verifier should enforce `depth <= MAX_DEPTH` outside of the circuit.

Since the verifier itself was outside the scope of this review, our team could not reason about a complete list of checks the verifier must perform and therefore recommends conducting a comprehensive follow-up security assessment of this component.

In the second audit, which focused on changes since the first audit, the scope of the review was sufficient and covered the changes since the previous audit.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	SEVERITY	STATUS
Issue A: Incorrect Accumulator Initialization in MulModPVar Breaks Modular Multiplication	Medium	Resolved
Issue B: Party Index Not Publicly Bound in KeyGenSinglePartyVar	Medium	Resolved
Issue C: Incorrect Identity Check in BabyJubJub Fixed-Scalar Multiplication Misclassifies the 2-Torsion Point	Medium	Resolved
Issue D: Missing Constraint in IsQuadraticResidueOrZero	Medium	Resolved
Issue E: Typo in Template Name Causes Undefined Component and Compilation Failure	Low	Resolved
Issue F: Missing Domain Separation Tag (DST) in HashToField	Low	Resolved
Suggestion 1: Improve Code Quality	Informational	Resolved

Suggestion 2: Improve Code Quality in Circom Tag Handling	Informational	Resolved
Suggestion 3: Update oprf_delegate Circuit	Informational	Resolved

Issue A: Incorrect Accumulator Initialization in MulModPVar Breaks Modular Multiplication

Location

circom/oprf_keys/keygen.circom#L134

circom/oprf_keys/keygen_single_party.circom#L55

Synopsis

The Circom template `MulModPVar` intends to compute `out = (a · b) mod q` over the `BabyJubJub` scalar field using a double-and-add ladder with b provided by `Num2Bits`. The initializer for the most significant bit uses the constant 8 instead of a, which renders the accumulator independent of a when the top bit of b is 1. This causes `out` to diverge from a·`b mod q`. As a result, the circuit can accept witnesses where polynomial evaluation and derived ciphertexts are incorrect while verification passes.

Impact

High.

In the case of b having its top bit equal to 1 and a not being equal to 8, the code produces a proof where `out` is not equal to a·`b mod q`. This error propagates to an incorrect `share` in `EvalPolyModPVar`, and subsequently to the `ciphertext` and `comm_share`, which become inconsistent with the committed polynomial in `KeyGenSinglePartyVar`. This violates soundness and compromises integrity of key share generation.

Feasibility

Low.

When the top bit of b equals 1 and a is not equal to 8, the code produces incorrect outputs. It is estimated that this occurs in approximately 50% of the cases. However, the system Taceo is currently deploying uses the 1-proof-for-n-shares variant `KeyGen`, not `KeyGenSinglePartyVar`. There remains a risk that the template `KeyGenSinglePartyVar` could be integrated into the system, in which case the issue would persist. Hence, the feasibility is set to Low.

Severity

Medium.

Preconditions

For this issue to occur, the following must hold true:

- The function `MulModPVar` must be used on a path that proves arithmetic for public outputs, for example via the function `EvalPolyModPVar` in the function `KeyGenSinglePartyVar`.
- The multiplier b given to the function `MulModPVar` must have its most significant bit set under `B_NUM_BITS` while a must not equal to 8.
- The template `KeyGenSinglePartyVar` must be integrated into the deployed system.

Technical Details
The function `MulModPVar` implements multiplication in the BabyJubJub scalar field `Fr` using a bitwise ladder: it decomposes b with `Num2Bits`, doubles the accumulator at each step with `AddModP`, and conditionally adds a with `Add3ModP`. It initializes `result[0]` as `Mux1()([0, 8], b_bits[B_NUM_BITS - 1])` rather than `Mux1()([0, a], ...)`, so when the top bit of b equals 1, the accumulator starts at 8 instead of a.

This substitution replaces the contribution of the highest bit of b with 8, producing an `out` value that differs from $a \cdot b \mod q$ for all inputs where the top bit is set and a does not equal 8. Because this incorrect value propagates through `EvalPolyModPVar` and into `EncryptAndCommit`, a prover can generate ciphertext and commitments for a share that does not correspond to the committed polynomial while the circuit constraints still verify successfully.

Remediation
We recommend replacing the initializer
`result[0] <== Mux1()([0, **8**], b_bits[B_NUM_BITS - 1])`
in the function `MulModPVar` with
`result[0] <== Mux1()([0, **a**], b_bits[B_NUM_BITS - 1])`.

Status
The TACEO team has resolved the issue as suggested.

Verification
Resolved.

Issue B: Party Index Not Publicly Bound in KeyGenSinglePartyVar

Location
`circom/oprf_keys/keygen_single_party.circom#L59`

Synopsis
The circuit constructs an encrypted share and commitments for a designated party in the function `KeyGenSinglePartyVar`. The variable `party_index` is a private input and is not part of the public statement, which allows a proof for one index to be presented as though it were intended for a different index. This may cause state updates, share attributions, or commitments to bind to an incorrect party.

Impact
High.

A malicious actor can generate a valid proof using one private `party_index` while the verifier records results under a different external index, corrupting protocol state or misattributing shares and commitments.

Feasibility
Low.

A malicious actor with access to proof generation can freely choose the private `party_index` and submit the proof wherever the verifier or contract supplies or records a separate index.

The system Taceo is currently deploying uses the 1-proof-for-n-shares variant `KeyGen` rather than `KeyGenSinglePartyVar`. However, there remains a risk that the template `KeyGenSinglePartyVar` could be integrated into the system, in which case the issue would persist. Hence, the feasibility is set to Low.

Severity
Medium.

Preconditions
For this issue to occur, the following must hold true:

- The verifier must use `party_index` outside the proof and must expect the proof to bind to it.
- The deployed verifying key must correspond to the current circuit where `party_index` is private.
- The adversary must be able to produce valid Groth16 proofs for the circuit.
- The template `KeyGenSinglePartyVar` must be integrated into the deployed system.

Technical Details
The function `KeyGenSinglePartyVar` takes `party_index` as a private input and computes the `share` via `EvalPolyModPVar(MAX_DEGREE, MAX_INDEX_BITS)(keygen_commit.poly_checked, party_index + 1)`. The outputs include `ciphertext`, `comm_share`, `comm_input_share`, `comm_coeffs`, and `my_pk`, all verified against a statement that excludes `party_index`.

Because `party_index` is private, the Groth16 input accumulator `vk_x` does not include it, and therefore the verifier cannot constrain which party the proof targets. A proof valid for index `i` may verify while external logic interprets it as corresponding to index `j`, leading to index substitution and misbinding.

Remediation
We recommend declaring `party_index` as a public input in the function `KeyGenSinglePartyVar`.

Status
The TACEO team has resolved the issue as recommended.

Verification
Resolved.

Issue C: Incorrect Identity Check in BabyJubJub Fixed-Scalar Multiplication Misclassifies the 2-Torsion Point

Location
`circom/babyjubjub/correct_sub_group.circom`

Synopsis
The `EscalarMulFixScalar` template treats any point with `x == 0` as the identity and returns the generator G. The BabyJubJub points `(0,1)` (identity) and `(0,-1)` (2-torsion) both satisfy `x == 0`, causing the gadget to incorrectly map `(0,-1)` to G. Subgroup checks that rely on this gadget can therefore yield incorrect results.

Impact
Medium.

A crafted input of `(0,-1)` can cause subgroup-membership logic and downstream computations to accept or operate on a non-prime-order point, undermining assumptions about prime-order inputs and potentially enabling constraint bypass in circuits that depend on those checks.

Feasibility
High.

Supplying the specific point `(0,-1)` is sufficient where inputs are attacker-controlled.

Severity
Medium.

Preconditions
For this issue to occur, an adversary must be able to provide a `BabyJubJub` point to a circuit path that uses `EscalarMulFixScalar` (or a subgroup check implemented with it) and must rely on the absence of independent constraints enforcing prime-order subgroup membership or a correct identity test.

Technical Details
The gadget branches on `x == 0` to special-case the identity and returns G. Because both `(0,1)` and `(0,-1)` satisfy `x == 0`, the branch conflates the 2-torsion element with the identity, producing an incorrect result. Where subgroup checks compute with this gadget (for example, using the prime order as a scalar), the check can return the incorrect result for `(0,-1)`, defeating the intended validation.

Remediation
We recommend replacing the `x-only` identity heuristic with a full identity check `(x == 0 ∧ y == 1)`.

Status
The TACEO team has resolved the issue as recommended.

Verification
Resolved.

Issue D: Missing Constraint in IsQuadraticResidueOrZero

Location
`circom/quadratic_residue/quadratic_residue.circom#L94`

Synopsis
When `a = 0`, the constraint intended to link the Legendre symbol `l` and witness b to input a collapses, allowing a prover to select `l ∈ {-1, 0, 1}` and still satisfy the template. Because `l` is assigned rather than constrained, the prover can cause the circuit to accept either `out = 0` or `out = 1` for the same input `a = 0`, breaking soundness for this edge case.

Impact
Medium.

A malicious prover can pass the quadratic-residue-or-zero check with an inconsistent `l` and `out` when `a = 0`, potentially bypassing logic that relies on this predicate for gating or correctness, and resulting in the acceptance of invalid statements.

Feasibility
High.

The attack requires only setting `a = 0` and choosing `l` accordingly, with no special capabilities required beyond constructing a witness.

Severity
Medium.

Preconditions
The circuit must run over a prime field Fp with `p ≠ 2`, the witness must allow `a = 0`, and `l` must not otherwise be constrained to the true Legendre symbol of a.

Technical Details
The gadget enforces `l(l-1)(b^2 - n*a) + (l+1)(b^2 - a) == 0`. Setting `a = 0` reduces this to `(l^2 + 1) * b^2 == 0`. For `l ∈ {-1, 0, 1}`, it follows that `l^2 + 1 ∈ {1, 2}`, which are invertible in Fp when `p ≠ 2`. The constraint then forces `b = 0` but imposes no restriction on `l`, allowing the prover to satisfy the gadget for both `out = 0` and `out = 1` with `a = 0`.

Mitigation
We suggest avoiding proofs where the gadget may receive `a = 0` until a code fix is deployed, rejecting instances with `a = 0` at the application layer if a is public, and treating statements depending on this gadget as unsafe if a is private and cannot be externally validated.

Remediation
We recommend constraining `l` when `a = 0` and, more generally, binding `l` to the true Legendre symbol of a. A minimal adjustment would be to compute `z = IsZero(a)` and enforce `z * l === 0` so that `a == 0` implies `l == 0`.

Status
The TACEO team has resolved the issue as recommended.

Verification
Resolved.

Issue E: Typo in Template Name Causes Undefined Component and Compilation Failure

Location
`circom/oprf_keys/keygen_single_party.circom#L33`

`circom/oprf_keys/keygen_single_party.circom#L83`

`circom/oprf_keys/keygen.circom#L226`

Synopsis
The single party key generation circuits instantiate a commitment template during compilation. The function `KeyGenSingleParty` and the function `KeyGenSinglePartyVar` instantiate `KeyGenCommmit`, but only the template `KeyGenCommit` exists, which makes the component undefined. As a result, the compile step fails and would stop any build or proof generation that references these templates.

Impact
Low.

This issue is expected to be detected during testing, since the compile step fails and consequently halts any build or proof generation that references these templates.

Feasibility
Low.

Severity
Low.

Technical Details
The function `KeyGenSingleParty` at line 33 and the function `KeyGenSinglePartyVar` at line 83 instantiate a component named `KeyGenCommmit`. The file `keygen.circom` defines the template `KeyGenCommit` at line 226, and no template named `KeyGenCommmit` exists.

Remediation
We recommend replacing `KeyGenCommmit` with `KeyGenCommit` in the function `KeyGenSingleParty` and the function `KeyGenSinglePartyVar`.

Status
The TACEO team has resolved the issue as recommended.

Verification
Resolved.

Issue F: Missing Domain Separation Tag (DST) in HashToField

Location
`circom/encode_to_curve_babyjj/encode_to_curve_babyjj.circom#L15`

Synopsis
The `HashToField` template does not accept or apply a domain separation tag (`DST`) as required by RFC 9380, and as a result, outputs are not bound to a specific protocol context. This enables cross-protocol collisions because the same input, when mapped in different contexts, yields identical field elements.

Impact
Low.

An attacker can reuse values across contexts or protocol stages that implicitly assume independent random oracles, enabling cross-protocol confusion, replay, or malleability. This does not directly expose secrets but can undermine binding and uniqueness assumptions in proofs or commitments.

Feasibility
Medium.

Exploitation requires another component or deployment that reuses the same `HashToField` without a distinct DST and the ability to influence or replay inputs.

Severity
Low.

Preconditions
The same `HashToField` construction must be reused in multiple contexts without effective domain separation, and an adversary must be able to supply or replay inputs between those contexts.

Technical Details

`HashToField` reduces inputs to the target field without a DST parameter. RFC 9380 specifies that hash-to-field and hash-to-curve constructions must include a domain separation tag so that different protocols instantiate independent random oracles. In the absence of a DST, for any message `m`, `HashToField(m)` in context A equals `HashToField(m)` in context B, allowing a value derived for one domain to be accepted in another. Because the construction is unkeyed and the output distribution is shared across uses, independence assumptions are invalidated even if the underlying hash is sound.

Remediation

We recommend adding an explicit `dst: bytes` parameter to `HashToField`, requiring a non-empty DST with length ≤ 255 bytes (or using the oversized-DST construction), and propagating it through all call sites with reviewed, stable constants.

Status

The TACEO team has resolved the issue as recommended.

Verification

Resolved.

Suggestions

Suggestion 1: Improve Code Quality

Location

`circom/eddsa_poseidon2/eddsaposeidon2.circom#L42`

`circom/merkle_tree/binary_merkle_root.circom#L38`

Synopsis

In the `EdDSAPoseidon2Verifier` template, a leftover assignment to `s_f` remains while `mulFix.e` is driven by `s_range.out_bits`, rendering the `S < subgroup_order` bound implicit and susceptible to misreading.

In the `BinaryMerkleRoot` template, the array `should_be_zero` should be declared as a `MAX_DEPTH`-sized array for clarity and consistency.

Mitigation

We recommend removing the leftover assignment and updating the array size.

Status

The TACEO team has implemented the mitigation as recommended (see here and here).

Verification

Resolved.

Suggestion 2: Improve Code Quality in Circom Tag Handling

Location

`circom/babyjubjub/babyjubjub.circom#L71-L74` circom/client_side_proofs/oprf_query.circom#L104-L110

Synopsis

We found two opportunities for code quality improvements related to the use of `circom` tags (see documentation here):

1. Here, the code manually assigns the tag `twisted_edwards` and performs the check with `BabyCheck()` instead of calling `BabyJubJubCheck()`.
2. Here, the code manually assigns the tag `twisted_edwards_in_subgroup` instead of adding the tag in the template `EncodeToCurveBabyJubJub`. This would be more maintainable because it would add the tag at the point where the circuit enforces the property.

Mitigation

We recommend addressing the two items above.

Status

The TACEO team has addressed the two items as recommended.

Verification

Resolved.

Suggestion 3: Update oprf_delegate Circuit

Location

circom/client_side_proofs/oprf_delegate.circom#L126-L134

Synopsis

In this code segment, the code first calls `BabyJubJubCheck` and then `BabyJubJubCheckInCorrectSubgroup`. The resulting `p_check` has the tag `twisted_edwards`, although it should have `twisted_edwards_in_subgroup` because `BabyJubJubScalarMul` expects this tag. The analogous logic in the `oprf_nullifier` circuit is implemented correctly. Because the `oprf_delegate` circuit is not in use, this issue has no negative impact. Nevertheless, we recommend updating the circuit because it may become relevant in the future.

Mitigation

We recommend calling `BabyJubJubCheckAndSubgroupCheck` directly.

Status

The TACEO team has addressed the suggestion as recommended.

Verification

Resolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.