



**Least Authority**  
PRIVACY MATTERS

ZK Prover

Security Audit Report

# Provekit

Updated Final Audit Report: 11 May 2026

# Table of Contents

## [Overview](#)

[Background](#)

[Project Dates](#)

[Review Team](#)

## [Coverage](#)

[Target Code and Revision](#)

[Supporting Documentation](#)

[Areas of Concern](#)

## [Findings](#)

### [General Comments](#)

[Dependencies](#)

[Code Quality](#)

[Documentation and Code Comments](#)

[Scope](#)

### [Specific Issues & Suggestions](#)

[Issue A: DSC Key Smuggling via Unsigned TBS Data Enables Forged Registration Proofs](#)

[Issue B: Claimed Public Input Values Are Not Bound to the Committed Witness](#)

[Issue C: Duplicate Witness Terms in R1CS Constraints Can Overwrite Coefficients and Underconstrain the Circuit](#)

[Issue D: Proof Soundness Can Break for u32 AND/XOR](#)

[Issue E: Unbound RSA Exponent Witness Enables Forged Passport Signature Verification](#)

[Issue F: Missing End-of-Transcript Validation Causes Proof Malleability](#)

[Issue G: Unbound LogUp Transcript Challenges Let the Prover Choose Lookup Randomness](#)

[Issue H: Unchecked Witness-Derived Indices Cause Deterministic Prover Panics in RAM and Multiplicity Builders](#)

[Issue I: Fiat-Shamir Transcript Does Not Explicitly Bind the Verification Key / Concrete R1CS](#)

[Issue J: Inconsistent Equality Polynomial Length Omits R1CS Constraints](#)

[Issue K: Unbound prover\\_hint\\_ark Values Leave a Latent Transcript-Binding Bug](#)

[Issue L: zkWHIR Batch Proofs Accept Alpha-Cancelled Forged Evaluations](#)

[Issue M: Unbound Step-2 G\\_j Claims Allow Rho-Adaptive Forged Evaluations](#)

[Issue N: zkWHIR Multi-Form Proofs Accept Form-RLC-Preserving Forged Evaluations](#)

[Issue O: zkWHIR Proofs Replay Under Alternate Linear Forms Sharing the Final-Point MLE Value](#)

[Issue P: WHIR Batched Verifier Accepts Vector-RLC-Cancelled Forged Evaluations Across Commitments](#)

[Issue Q: WHIR Proofs Replay Under Alternate Linear Forms Sharing the Final-Point MLE Value](#)

[Issue R: WHIR Multi-Form Proofs Accept Form-RLC-Preserving Forged Evaluations](#)

[Issue S: Proof Malleability Through Trailing Bytes and Non-Canonical Field Element Encodings](#)

[Issue T: WHIR Verifier Panics on Unchecked Divisions by Zero](#)

### [Suggestions](#)

[Suggestion 1: Implement Full PKI Checks for Passports](#)

[Suggestion 2: Improve Code Comments](#)

[Suggestion 3: Reject Malformed 32-Bit Constants to Prevent Proving a Different Statement Than the Noir Artifact Specifies](#)

[Suggestion 4: Refine Computation of Reported Security Level](#)

## [About Least Authority](#)

## [Our Methodology](#)

# Overview

## Background

Tools for Humanity has requested that Least Authority perform a security audit of its ZK Prover, a modular zero-knowledge proof toolkit optimized for mobile devices.

## Project Dates

- **February 23, 2026 - March 17, 2026:** Initial Code Review (*Completed*)
- **March 19, 2026:** Delivery of Initial Audit Report (*Completed*)
- **March 25, 2026:** First Round of Verification Review (*Completed*)
- **April 24, 2026:** Delivery of Updated Initial Audit Report (*Completed*)
- **May 8, 2026:** Second Round of Verification Review (*Completed*)
- **May 11, 2026:** Delivery of Updated Final Audit Report (*Completed*)

## Review Team

- Poulami Das, Security / Cryptography Researcher and Engineer
- George Gkitsas, 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 the ZK Prover 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:

- [https://github.com/worldfnd/provekit/blob/main/provekit/prover/src/whir\\_r1cs.rs](https://github.com/worldfnd/provekit/blob/main/provekit/prover/src/whir_r1cs.rs)
- <https://github.com/worldfnd/provekit/tree/main/provekit/prover/src/witness>
- <https://github.com/WizardOfMenlo/whir/tree/main/src/protocols/whir>
- <https://github.com/worldfnd/provekit/tree/main/noir-examples/noir-passport>
  - examples (circuits for the passport age check, written in Noir)
- <https://github.com/worldfnd/provekit/tree/main/provekit/r1cs-compiler>
  - Noir to R1CS compiler
- <https://github.com/worldfnd/provekit/tree/main/provekit/common>
  - the witness scheme, R1CS format, and related components that are shared between the prover and the verifier
- <https://github.com/worldfnd/provekit/tree/main/provekit/verifier>
  - Spartan verifier
- <https://github.com/WizardOfMenlo/whir/tree/main/src/protocols/whir>
  - WHIR prover and verifier
- [https://github.com/WizardOfMenlo/whir/tree/main/src/protocols/whir\\_zk](https://github.com/WizardOfMenlo/whir/tree/main/src/protocols/whir_zk)
  - ZK WHIR prover and verifier

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

- ProveKit: `4037ecbd65176e7425eaf612cd8b4e33bd09feb4`

- Whir: b3c86d750b4e4c02696b42fe7c9e38586459b24d

For the first round of verification, we examined the following Git revisions:

- ProveKit: e769e1c24d80894ff014eb6c23d2cc21dd33c926
- Whir: 56332ef919f43ba06d1dbd3a8ff474498130b5c9

For the second round of verification, we examined the following Git revisions:

- ProveKit: 87c96b7d833960cb888a85739632fb9f73edbf32
- Whir: 0a686278bce4bcad8892cd8b7bc1d9c79f2bdff5

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

- worldfnd-provekit:  
<https://github.com/LeastAuthority/worldfnd-provekit>
- WizardOfMenlo-whir:  
<https://github.com/LeastAuthority/WizardOfMenlo-whir>
- Whir proof size reduction PR:  
<https://github.com/WizardOfMenlo/whir/pull/239>

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:

- ProveKit Security Audit Guide:  
<https://attheon-labs.notion.site/ProveKit-Security-Audit-307803c72120809bac7cda39984118b8>
- zkWHIR protocol to reduce proof size:  
<https://attheon-labs.notion.site/Whir-Proof-Size-Reduction-32c803c72120802ca7d0d1a9617706f0>
- Website:  
<https://www.toolsforhumanity.com>

In addition, this audit report references the following documents:

- G. Arnon, A. Chiesa, G. Fenzi, and E. Yogev, "WHIR: Reed–Solomon Proximity Testing with Super-Fast Verification." *IACR Cryptology ePrint Archive*, 2024, [ACF+24]
- Doc 9303:  
<https://www.icao.int/publications/doc-series/doc-9303>

## Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- 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

Our team reviewed the ZK Prover, whose implementation is based on the well-formalized [\[ACF+24\]](#) research paper, and found that it has been well designed, with security at the forefront of its design.

#### **ProveKit, WHIR, and zkWHIR**

We audited the modules within `provekit` used for compiling Noir programs into an R1CS-based proof system, and generating and checking proofs. `provekit/r1cs-compiler` takes a compiled Noir circuit and turns it into the R1CS form that ProveKit uses to generate witnesses and proofs. `provekit/prover` executes the Noir program to build the witness, splits and commits it for the Whir-backed protocol, and produces a `NoirProof` containing an embedded `WhirR1CSProof`. `provekit/verifier` replays the transcript and verifies the embedded `WhirR1CSProof` against the circuit's R1CS and declared public inputs.

We also reviewed `whir/src/protocols/*`, which contains the protocol building blocks for the WHIR commitment and opening, all driven through a Fiat-Shamir transcript. `matrix_commit`, `merkle_tree`, and `irs_commit` handle committing to vectors and Reed-Solomon encoded matrices, while `sumcheck`, `challenge_indices`, `geometric_challenge`, and `proof_of_work` provide the interactive reduction, challenge sampling, and anti-spam machinery. `protocols/whir` composes those pieces into the main WHIR opening protocol for proving evaluations of committed multilinear data against linear forms. `protocols/whir_zk` adds the blinding and masked-subproof layer required to make that WHIR opening protocol zero-knowledge.

We found that the verifier does not recompute and compare the expected compressed public-input value against the committed witness prefix, which can result in the acceptance of proofs for a different claimed public statement than the one actually committed ([Issue B](#)). We also found that LogUp transcript challenges are sampled but never bound back to the witness or enforced in the verification equations, which can allow lookup constraints to be satisfied with attacker-chosen randomness rather than verifier-derived challenges ([Issue G](#)). In a related issue, the verifier does not perform end-of-transcript and end-of-hints validation, which can result in proof malleability through the acceptance of encodings with arbitrary trailing bytes ([Issue F](#)). Separately, witness-derived indices are used without proper bounds checks in RAM and multiplicity builders, which can result in deterministic prover panics and denial-of-service conditions under malformed inputs ([Issue H](#)). We observed that Fiat-Shamir challenges are sampled before the transcript absorbs a digest of the concrete verification key or R1CS, which can result in incomplete statement binding and possible cross-key or cross-circuit confusion in dynamic verification settings ([Issue I](#)). We also identified that values passed through `prover_hint_ark` are not absorbed into the Fiat-Shamir transcript state, which creates a latent transcript-binding risk if such hinted values are later assumed to be security-relevant ([Issue K](#)). In addition, the equality polynomial may be evaluated over a domain smaller than the full R1CS row space, which can result in trailing constraints receiving zero weight and being skipped during verification under inconsistent parameterization ([Issue J](#)).

### noir-passport Circuits and Related Compiler

We further examined the `noir-passport` example circuits and supporting Noir libraries that implement the passport registration and attestation flow. We identified two critical issues that allow the forging of passport-related proofs ([Issue A](#) and [Issue E](#)). We also found that the PKI checks are incomplete ([Suggestion 1](#)).

Within the R1CS compilation layer, we identified two critical soundness issues ([Issue D](#) and [Issue C](#)).

### zkWHIR Proof Size Reduction

As part of the review extension, we examined the [PR](#) that updates the zkWHIR wrapper protocol to reduce proof size. We found that the zkWHIR verifier collapses the public evaluations, the Step-2  $G_j$  blinding claims, the per-form linear-form claims, and the public linear forms themselves into single aggregates that are never rebound to the Fiat-Shamir transcript or checked individually. This can result in the acceptance of alpha-canceled forged evaluation rows under batching ([Issue L](#)), rho-adaptive forged evaluations that compensate a perturbed  $G_j$  message ([Issue M](#)), form-RLC-preserving forged evaluation vectors under multiple linear forms ([Issue N](#)), and replayed proofs under alternate linear forms that share the final-point MLE value ([Issue O](#)). During this PR review, we found that the WHIR verifier is also susceptible to similar attacks to those mentioned above. Specifically, the WHIR verifier never absorbs the public evaluations matrix or the public linear forms into Fiat-Shamir before sampling its RLC coefficients. This can result in the acceptance of vector-RLC-canceled forged evaluations across commitments ([Issue P](#)), replayed proofs under alternate linear forms that share the final-point MLE value ([Issue Q](#)), and form-RLC-preserving forged evaluation vectors under multiple linear forms ([Issue R](#)).

## Dependencies

Running `cargo audit` yielded low-significance warnings for both repositories. All warnings are in indirect dependencies except for `keccak`, which is used by out-of-scope code. While no issues were identified in the assessed warnings, no determination was made for `keccak`.

## Code Quality

We performed a manual review of the repositories in scope and found the code to be well-organized and aligned with development best practices.

### Tests

The repositories in scope demonstrate good test density, with a strong focus on low-level arithmetic logic, and include both unit tests and property tests.

## Documentation and Code Comments

The project documentation provided by the World team, comprising a research paper, documentation, and an audit guide, clearly outlines the project's purpose and sufficiently describes the system's intended functionality. However, there are insufficient code comments describing security-critical components and functions in the codebase. We recommend improving code comments ([Suggestion 2](#)).

## Scope

The scope of this review was sufficient. However, completing the review required extending the analysis to the `noir-bignum`, `noir_rsa_v092`, `noir`, and `noir-date` libraries. Despite that coverage, the complexity of the codebase and the number of potential failure modes across different paths and interactions made it impractical to evaluate every case and permutation exhaustively within the review. As a result, we recommend a comprehensive follow-up audit once the issues and recommendations identified in this report have been addressed and development is finalized.

## 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   |
|--------------------------------------------------------------------------------------------------------------------------------|----------|----------|
| <a href="#">Issue A: DSC Key Smuggling via Unsigned TBS Data Enables Forged Registration Proofs</a>                            | Critical | Resolved |
| <a href="#">Issue B: Claimed Public Input Values Are Not Bound to the Committed Witness</a>                                    | Critical | Resolved |
| <a href="#">Issue C: Duplicate Witness Terms in R1CS Constraints Can Overwrite Coefficients and Underconstrain the Circuit</a> | Critical | Resolved |
| <a href="#">Issue D: Proof Soundness Can Break for u32 AND/XOR</a>                                                             | Critical | Resolved |
| <a href="#">Issue E: Unbound RSA Exponent Witness Enables Forged Passport Signature Verification</a>                           | Critical | Resolved |
| <a href="#">Issue F: Missing End-of-Transcript Validation Causes Proof Malleability</a>                                        | High     | Resolved |
| <a href="#">Issue G: Unbound LogUp Transcript Challenges Let the Prover Choose Lookup Randomness</a>                           | High     | Resolved |
| <a href="#">Issue H: Unchecked Witness-Derived Indices Cause Deterministic Prover Panics in RAM and Multiplicity Builders</a>  | Medium   | Resolved |
| <a href="#">Issue I: Fiat-Shamir Transcript Does Not Explicitly Bind the Verification Key / Concrete R1CS</a>                  | Low      | Resolved |
| <a href="#">Issue J: Inconsistent Equality Polynomial Length Omits R1CS Constraints</a>                                        | Low      | Resolved |
| <a href="#">Issue K: Unbound prover_hint_ark Values Leave a Latent Transcript-Binding Bug</a>                                  | Low      | Resolved |
| <a href="#">Issue L: zkWHIR Batch Proofs Accept Alpha-Cancelled Forged Evaluations</a>                                         | Critical | Resolved |
| <a href="#">Issue M: Unbound Step-2 G_j Claims Allow Rho-Adaptive Forged Evaluations</a>                                       | Critical | Resolved |
| <a href="#">Issue N: zkWHIR Multi-Form Proofs Accept Form-RLC-Preserving Forged Evaluations</a>                                | Critical | Resolved |
| <a href="#">Issue O: zkWHIR Proofs Replay Under Alternate Linear Forms Sharing the Final-Point MLE Value</a>                   | Critical | Resolved |
| <a href="#">Issue P: WHIR Batched Verifier Accepts Vector-RLC-Cancelled Forged Evaluations Across Commitments</a>              | Critical | Resolved |

|                                                                                                                                           |               |            |
|-------------------------------------------------------------------------------------------------------------------------------------------|---------------|------------|
| <a href="#">Issue Q: WHIR Proofs Replay Under Alternate Linear Forms Sharing the Final-Point MLE Value</a>                                | Critical      | Resolved   |
| <a href="#">Issue R: WHIR Multi-Form Proofs Accept Form-RLC-Preserving Forged Evaluations</a>                                             | Critical      | Resolved   |
| <a href="#">Issue S: Proof Malleability Through Trailing Bytes and Non-Canonical Field Element Encodings</a>                              | Medium        | Unresolved |
| <a href="#">Issue T: WHIR Verifier Panics on Unchecked Divisions by Zero</a>                                                              | Medium        | Resolved   |
| <a href="#">Suggestion 1: Implement Full PKI Checks for Passports</a>                                                                     | Informational | Unresolved |
| <a href="#">Suggestion 2: Improve Code Comments</a>                                                                                       | Informational | Unresolved |
| <a href="#">Suggestion 3: Reject Malformed 32-Bit Constants to Prevent Proving a Different Statement Than the Noir Artifact Specifies</a> | Informational | Resolved   |
| <a href="#">Suggestion 4: Refine Computation of Reported Security Level</a>                                                               | Informational | Unresolved |

## Issue A: DSC Key Smuggling via Unsigned TBS Data Enables Forged Registration Proofs

### Location

[noir-passport/merkle\\_age\\_check/t\\_add\\_dsc\\_verify\\_1300/src/main.nr#L63](#)

[noir-passport/merkle\\_age\\_check/t\\_add\\_dsc\\_verify\\_1300/src/main.nr#L76](#)

[noir-passport/utils/data-check/tbs-pubkey/src/lib.nr#L11](#)

[noir-passport/merkle\\_age\\_check/t\\_add\\_id\\_data\\_1300/src/main.nr#L21](#)

### Synopsis

The 1,300-byte registration flow authenticates only the prefix `tbs_certificate[0..tbs_certificate_len)` under the CSCA signature, but later commits to and reuses the entire fixed-size 1,300-byte TBS buffer. Because trailing bytes are not constrained to zero and downstream key extraction only checks that the prover-chosen offset lies within the buffer, a malicious prover can inject an attacker-controlled DSC public key into unsigned tail bytes and satisfy the certificate checks.

### Impact

High.

An attacker can produce registration proofs that appear to bind passport data to a CSCA-certified DSC while actually using an attacker-controlled RSA key to sign the SOD-related inputs. This breaks the intended CSCA-to-DSC trust chain and allows synthetic identities to be registered and later attested as though they originated from a legitimate passport issuance chain.

### Feasibility

High.

**Severity**  
Critical.

**Preconditions**  
None.

#### Technical Details

In `t_add_dsc_verify_1300`, the circuit finalizes the SHA-256 hash using `tbs_certificate_len` and verifies the CSCA signature over that authenticated prefix only. It then outputs `comm_out = hash_salt_country_tbs(salt_out, country, tbs_certificate)`, which commits to all 1,300 bytes, including any attacker-controlled bytes beyond the signed length. There is no zero-padding constraint on the trailing region.

In `verify_rsa_pubkey_in_tbs`, the only safety check is `pubkey_offset + DSC_KEY_SIZE <= TBS_CERT_SIZE`. The function does not require `pubkey_offset + DSC_KEY_SIZE <= tbs_certificate_len`, nor does it validate that the offset corresponds to the actual ASN.1 SubjectPublicKeyInfo location. As a result, a prover can place an attacker-controlled 256-byte RSA modulus in the unsigned tail, choose `dsc_pubkey_offset_in_dsc_cert` to point at that tail region, and satisfy the byte-equality check.

Because `t_add_id_data_1300` then verifies the SOD signature using the attacker-chosen `dsc_pubkey`, the attacker can sign `signed_attributes` with the corresponding private key and pass the downstream signature checks. The forged chain can then propagate into `t_add_integrity_commit` and `t_attest`, allowing a maliciously registered leaf to produce valid attestation proofs.

#### Remediation

We recommend the following steps to remediate this issue:

- Enforcing zero-padding for every `tbs_certificate[i]` where `i >= tbs_certificate_len` in `t_add_dsc_verify_1300`.
- Binding `tbs_certificate_len` into the commitment chain, or committing only to the authenticated prefix and its length instead of the raw fixed-size 1,300-byte buffer.
- Changing `verify_rsa_pubkey_in_tbs` to accept the signed TBS length and requiring `pubkey_offset + DSC_KEY_SIZE <= tbs_certificate_len`.
- Adding structural validation so the extracted key corresponds to the expected SubjectPublicKeyInfo region rather than an arbitrary substring.
- Adding regression tests demonstrating that:
  - non-zero bytes in the unsigned tail fail;
  - a pubkey offset into the unsigned tail fails; and
  - an end-to-end forged registration or attestation chain, such as the validated PoC, fails after the fix.

#### Status

The World team has addressed this issue in [this PR](#), which includes a zero-padding constraint in `t_add_dsc_verify_1300`. Bytes with index `i >= tbs_certificate_len` must be zero. This closes the attack path where an attacker injects a forged DSC modulus into the unsigned trailing bytes. With tail bytes forced to zero, that attacker-controlled key material can no longer be introduced through the unsigned region.

The PR does not affect the following:

- [noir-examples/noir-passport/utils/data-check/tbs-pubkey/src/lib.nr#L10](#) still only checks `pubkey_offset + DSC_KEY_SIZE <= tbs_certificate_len`. It does not take `tbs_certificate_len` and it does not validate the ASN.1/SPKI location.
- [noir-examples/noir-passport/merkle\\_age\\_check/t\\_add\\_id\\_data\\_1300/src/main.nr#L21](#) still calls that helper without signed-length context.

The PR does not include any regression tests.

#### Verification

Resolved.

## Issue B: Claimed Public Input Values Are Not Bound to the Committed Witness

#### Location

[provekit/prover/src/whir\\_r1cs.rs#L219](#)

[provekit/prover/src/whir\\_r1cs.rs#L538](#)

[provekit/verifier/src/whir\\_r1cs.rs#L151](#)

[src/protocols/whir\\_zk/verifier.rs#L116](#)

#### Synopsis

The verifier accepts a proof without checking that the claimed public input values match the public portion of the committed witness.

#### Impact

High.

A malicious prover may cause acceptance of a proof for a different public statement, compromising statement integrity at the verifier.

#### Feasibility

High.

A prover with permission to submit proofs and a satisfiable witness can exploit this condition without breaking the commitment or transcript scheme, as the missing value comparison is structural.

#### Severity

Critical.

#### Preconditions

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

- At least one public input must be present.
- A malicious prover must be able to generate and submit a `WhirR1CSProof`.
- The verifier must rely on the claimed public inputs to define the accepted statement.

#### Technical Details

The function `get_public_weights` sends only the value `public_inputs.hash()` into the transcript and derives the variable `x` from that hash. In both branches of the function `verify`, the verifier accepts

the variable `public_eval` or the variable `public_1` as a valid opening through the function `self.whir_witness.verify` after constructing the linear form from the function `make_public_weight`, but it never recomputes the expected compressed value from the claimed public inputs.

A malicious prover can commit to a witness whose public prefix differs from the claimed public inputs, derive the variable `x` from the claimed hash, and produce the corresponding opening through the function `compute_public_eval` or the function `compute_public_weight_evaluation`. The WHIR layer only validates the commitment-opening relation for that supplied value, so the proof can verify for a different public input vector with the same length.

#### Remediation

We recommend replacing the current public-input validation with a verifier-side recomputation of the expected compressed value from the claimed public inputs under the variable `x`, and using the function `self.whir_witness.verify` only to prove that the committed witness prefix opens to that exact value.

#### Status

The World team has [resolved](#) the issue by adding a verifier-side validation to ensure that the prover's claimed public evaluation is bound to the public portion of the committed witness.

#### Verification

Resolved.

### Issue C: Duplicate Witness Terms in R1CS Constraints Can Overwrite Coefficients and Underconstrain the Circuit

#### Location

[provekit/common/src/r1cs.rs#L73](#)

[provekit/common/src/sparse\\_matrix.rs#L347](#)

[provekit/r1cs-compiler/src/noir\\_to\\_r1cs.rs#L260](#)

[provekit/r1cs-compiler/src/noir\\_to\\_r1cs.rs#L276](#)

[provekit/r1cs-compiler/src/poseidon2/permutation.rs#L95](#)

[provekit/r1cs-compiler/src/poseidon2/permutation.rs#L362](#)

[provekit/r1cs-compiler/src/poseidon2/permutation.rs#L501](#)

[provekit/r1cs-compiler/src/poseidon2/permutation.rs#L707](#)

#### Synopsis

The compiler emits R1CS rows as raw lists of (`coefficient`, `witness`) terms, but the backend does not accumulate repeated witness indices within a row. Instead, later writes overwrite earlier ones. As a result, if the same witness appears more than once in a linear combination, the emitted row no longer represents the intended expression.

### Impact

High.

This is a soundness issue. The generated R1CS can accept witnesses that satisfy the weaker compiled constraints while violating the intended circuit semantics.

### Feasibility

High.

### Severity

Critical.

### Preconditions

None.

### Technical Details

`R1CS::add_constraint` writes each term in a row by calling `SparseMatrix::set`. `set` performs a keyed update on `(row, column)` and overwrites any existing entry instead of adding coefficients together. Accordingly, the backend only stores one coefficient for each witness in a given row.

Some code paths compensate for this by manually combining duplicate witnesses before emitting a row. For example, `spread_decompose` does so explicitly. However, the compiler does not enforce this invariant globally.

Two generic helpers remain unsafe:

- `add_sum` forwards summands directly into `add_constraint` with no canonicalization.
- `add_acir_assert_zero` builds the linear row from cached products and ACIR linear terms and emits it unchanged.

Poseidon2 provides a concrete and currently reachable manifestation of the issue. In `mds_t_block_forms`, each output lane for widths 8, 12, and 16 is built as `arr[j] + col_acc[j]`, even though `col_acc[j]` already contains `arr[j]`. Because `LinearForm::add_mut` only concatenates terms, duplicate witness indices are created by construction. Those raw duplicate terms are then passed into `linear_form_pow5` and into the final output constraints at `permutation.rs#L707`, where they are silently weakened by overwrite semantics.

As a result, the verifier checks a different relation from the one the compiler author intended.

### Remediation

We recommend applying remediation at the abstraction boundary rather than leaving it to individual call sites, through the following measures:

- Canonicalizing linear combinations inside `R1CS::add_constraint` before inserting them into the sparse matrices, or changing the sparse-matrix write path so repeated `(row, witness)` inserts accumulate coefficients instead of overwriting them.
- Adding regression tests that fail if any emitted row contains duplicate witness indices.
- Retaining local canonicalization in complex gadgets as an additional layer of security, but not relying on it as the primary safety mechanism.

### Status

The World team has implemented the remediation in [this PR](#). `canonicalize_terms` is now applied before any matrix writes, creating an abstraction that merges duplicate witness indices and drops zero

coefficients. This covers `add_sum`, `add_acir_assert_zero`, and `Poseidon2`. `sparse_matrix` is also hardened so that a duplicate insert is treated as error instead of allowing a silent overwrite. The PR also adds regression tests.

#### Verification

Resolved.

### Issue D: Proof Soundness Can Break for u32 AND/XOR

#### Location

[provekit/r1cs-compiler/src/binops.rs](#)

[provekit/r1cs-compiler/src/binops.rs](#)

[provekit/r1cs-compiler/src/noir\\_to\\_r1cs.rs](#)

[provekit/r1cs-compiler/src/digits.rs](#)

#### Synopsis

When the combined AND/XOR lookup path selects an atomic digit width that does not divide 8, the subsequent decomposition of each intermediate “byte” overallocates bit capacity. In the case where `atomic_bits = 3`, each byte is represented as three 3-bit digits, which permits values in `[0, 511]` rather than `[0, 255]`. Because the byte witnesses created during u32 binop lowering are not independently range-checked, a malicious prover can use non-canonical byte representations that still recompose to the same 32-bit word while causing the per-byte AND/XOR constraints to validate an incorrect result.

#### Impact

High.

This is a soundness failure in the compiled relation for Noir u32 bitwise AND/XOR. A malicious prover may be able to produce a proof that satisfies the R1CS constraints even though the claimed bitwise output does not match Noir’s intended 32-bit semantics.

#### Feasibility

High.

The vulnerable path is automatically selected when the circuit contains enough affected binops for `get_optimal_binop_width` to choose `atomic_bits = 3`. Once that occurs, the prover only needs to exploit the non-canonical intermediate byte representation permitted by the current constraints.

#### Severity

Critical.

#### Preconditions

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

- The compiled circuit must contain Noir u32 bitwise AND and/or XOR operations that lower to `BlackBoxFuncCall::AND` or `BlackBoxFuncCall::XOR`.
- The optimizer must select `atomic_bits = 3`.

### Technical Details

`process_binop_opcode` first decomposes each 32-bit operand and output into four base-256 limbs that are treated as bytes. However, those limbs are only tied back to the original 32-bit value through a recomposition constraint; they are not individually range-checked as true bytes.

Later, `add_combined_binop_constraints` may choose `atomic_bits = 3`. In that case, it computes  $d = \text{ceil}(8 / 3) = 3$  and uses `log_bases = [3, 3, 3]` for the decomposition of each intermediate byte. This causes each byte witness  $B$  to satisfy:

$$B = d_0 + 2^3 d_1 + 2^6 d_2 = d_0 + 8*d_1 + 64*d_2$$

where each digit is constrained only to lie in  $[0, 7]$  by the lookup table. The maximum representable value is therefore:

$$7 + 8*7 + 64*7 = 511$$

As a result, the system no longer constrains each intermediate “byte” to 8 bits. For example, the value 256 is invalid as a byte, but it is accepted by the widened decomposition because:

$$256 = 0 + 8*0 + 64*4$$

with valid 3-bit digits  $(0, 0, 4)$ .

This permits non-canonical decompositions of a 32-bit word. A prover can choose byte witnesses that still recombine to the correct 32-bit input, while the lookup enforces AND/XOR on those limbs rather than on canonical 8-bit bytes. Consequently, the lookup can accept a result that differs from the intended one.

### Remediation

We recommend applying a remediation that restores canonical 8-bit byte semantics for the intermediate data by:

- Restricting `get_optimal_binop_width` so it cannot select values that do not divide by 8.
- Adding explicit range checks to ensure every intermediate byte witness used by the combined binop path lies in  $[0, 255]$ .
- Adding regression tests covering `atomic_bits = 3`, non-canonical byte attempts, and end-to-end u32 AND and XOR correctness.

### Status

The World team has [added](#) a width restriction to  $\{2, 4, 8\}$ , a corresponding assertion in the cost function, and related tests.

### Verification

Resolved.

## Issue E: Unbound RSA Exponent Witness Enables Forged Passport Signature Verification

### Location

[provekit/noir-examples/noir-passport/utis/data-check/tbs-pubkey/src/lib.nr#L10-L15](https://github.com/ProveKit/noir-examples/blob/main/noir-passport/utis/data-check/tbs-pubkey/src/lib.nr#L10-L15)

[provekit/noir-examples/noir-passport/merkle\\_age\\_check/t\\_add\\_id\\_data\\_720/src/main.nr#L22-L29](#)

[provekit/noir-examples/noir-passport/merkle\\_age\\_check/t\\_add\\_id\\_data\\_1300/src/main.nr#L21-L29](#)

[provekit/noir-examples/noir-passport/merkle\\_age\\_check/t\\_add\\_dsc\\_720/src/main.nr#L23-L30](#)

[provekit/noir-examples/noir-passport/merkle\\_age\\_check/t\\_add\\_dsc\\_verify\\_1300/src/main.nr#L66-L71](#)

### Synopsis

The RSA public exponent is treated as a free witness and is not cryptographically bound to the authenticated public key material. As a result, the circuits prove signature validity for an attacker-chosen key ( $n$ ,  $e$ ) rather than the certified key. For DSC, the certificate check only constrains the modulus bytes, not the exponent. Because the pinned `noir_rsa` dependency accepts arbitrary positive exponents below  $2^{17}$ , a malicious prover can choose  $e = 1$  and bypass signature verification.

### Impact

High.

This breaks the soundness of the passport authenticity checks. A malicious prover can satisfy CSC→DSC and DSC→SOD RSA verification without possessing the corresponding private keys, allowing proofs over attacker-chosen DSC certificates, signed attributes, and downstream identity data. In practice, this undermines the claim that the proved passport data is authenticated by the issuing PKI.

### Feasibility

High.

A malicious prover controls private witnesses and needs only to provide a chosen exponent, a matching modulus-derived reduction parameter, and a crafted signature equal to the PKCS#1 v1.5 encoded message block.

### Severity

Critical.

### Preconditions

None.

### Technical Details

The affected circuits pass exponent directly into RSA verification as a private input. For the DSC→SOD verification, `verify_rsa_pubkey_in_tbs` only checks that the modulus bytes appear at the claimed offset in the DSC certificate. The exponent is never validated against the one encoded in the public key. For the CSC→DSC verification, the exponent is independent of the trusted CSC modulus.

The RSA gadget `zkpassport/noir_rsa@v0.9.2` enforces only  $0 < \text{exponent} < 131072$  and then computes  $\text{sig}^e \bmod n$  before checking PKCS#1 formatting. It does not reject  $e = 1$ .

A malicious prover can therefore:

- Compute the SHA-256 hash of a selected message.

- Construct the correct PKCS#1 v1.5 encoded message  $EM = 0x00 \ || \ 0x01 \ || \ FF \dots FF \ || \ 0x00 \ || \ DigestInfo \ || \ hash$ .
- Set the witness exponent  $= 1$ .
- Set the witness signature  $= EM$ .
- Use the certificate's modulus  $n$ .

Inside the RSA gadget, verification becomes  $signature^{e \bmod n} = signature$ , so the recovered value is exactly  $EM$ . The PKCS#1 comparison then succeeds even though no private key was used. Because the circuit never checks that the chosen exponent matches the exponent encoded in the certified public key, the proof is accepted against an attacker-chosen key  $(n, 1)$ .

An attacker can exploit the DSC→SOD check directly using any real DSC modulus, or chain the flaw twice to forge both the CSC→DSC and the DSC→SOD verifications.

### Mitigation

If the set of valid exponents for passports is fixed, we suggest using an assertion for those values.

### Remediation

For full remediation, we recommend:

- Binding the exponent to the same authenticated key material as the modulus.
- Validating, for DSC, the full RSA public key encoding inside the signed certificate within the circuit, rather than only the modulus bytes.
- Including, for CSC, the exponent as well as the modulus in the trusted key representation and, if the CSC key is supplied directly or through a Merkle commitment, committing to the full SPKI DER or a hash over  $(modulus, exponent)$  rather than the modulus alone.
- Recomputing the Barrett or reduction parameter from the authenticated modulus rather than accepting it as an unrelated witness.
- Adding regression tests to ensure that:
  - exponent  $= 1$  with a crafted PKCS#1 block fails; and
  - a correct modulus with a mismatched exponent fails.

### Status

The World team has closed the  $e = 1$  attack path in [this PR](#) by rejecting any non-allowlisted exponent in both RSA verification paths. It binds the DSC exponent to bytes in the authenticated TBS certificate instead of leaving it as a free witness, and binds the CSC exponent into a hash over  $(modulus, exponent)$ . The PR also adds a unit test.

### Verification

Resolved.

## Issue F: Missing End-of-Transcript Validation Causes Proof Malleability

### Location

[provekit/verifier/src/whir\\_r1cs.rs#L38](#)

[src/transcript/mod.rs#L284](#)

### Synopsis

The verifier accepts proofs with unparsed trailing bytes because it does not enforce end-of-input validation for the transcript and hints buffers.

### Impact

Medium.

An attacker can produce multiple distinct encodings of the same valid proof by appending arbitrary bytes to `WhirR1CSProof.narg_string` or `WhirR1CSProof.hints`, potentially undermining proof canonicalization assumptions in downstream logic.

### Feasibility

High.

Any party that can submit a proof can append arbitrary trailing bytes without affecting verification because the function `WhirR1CSScheme::verify` never calls the function `VerifierState::check_eof`.

### Severity

High.

### Preconditions

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

- The attacker must be able to submit a `WhirR1CSProof` to a verifier using the function `WhirR1CSScheme::verify`.
- A downstream component must treat the serialized proof bytes as a canonical or unique representation.

### Technical Details

The function `WhirR1CSScheme::verify` constructs the variable `arthur` as a `VerifierState` over `WhirR1CSProof.narg_string` and `WhirR1CSProof.hints`, consumes only the messages required by the protocol, and then returns `Ok()` without calling the function `VerifierState::check_eof`.

The function `VerifierState::check_eof` is implemented to validate that the underlying transcript `self.inner.check_eof()` succeeds and that the hints slice `self.hints` is empty. Skipping it permits adding trailing bytes in either buffer.

Given any valid proof blob  $\pi$ , an attacker can create  $\pi'$  by appending arbitrary bytes to `WhirR1CSProof.narg_string` and/or `WhirR1CSProof.hints`. Verification succeeds because parsing stops after the expected interactions, and the appended bytes are silently ignored.

### Remediation

We recommend calling the function `VerifierState::check_eof` at the end of the function `WhirR1CSScheme::verify` and propagating its error so that any trailing bytes in `WhirR1CSProof.narg_string` or `WhirR1CSProof.hints` cause verification to fail.

### Status

The World team has [resolved](#) the issue as recommended.

### Verification

Resolved.

## Issue G: Unbound LogUp Transcript Challenges Let the Prover Choose Lookup Randomness

### Location

[provekit/provekit/verifier/src/whir\\_r1cs.rs#L58](#)

[provekit/provekit/verifier/src/whir\\_r1cs.rs#L151](#)

[provekit/provekit/common/src/witness/witness\\_builder.rs#L81](#)

### Synopsis

The verifier samples LogUp challenges from the transcript but does not bind them to the committed witness or the verification equations.

### Impact

High.

A malicious prover could select lookup randomness for the challenge witness slots and satisfy lookup constraints under that forged randomness rather than under the verifier-derived transcript challenges.

### Feasibility

Medium.

An attacker with the ability to generate proofs could exploit this by constructing a witness and proof in which the challenge slots differ from the verifier-sampled values because the verifier performs no binding check.

### Severity

High.

### Preconditions

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

- The parameter `WhirR1CSScheme.num_challenges` must be greater than zero.
- A circuit must rely on the type `WitnessBuilder::Challenge` or other challenge-indexed LogUp witnesses for lookup-style gadgets.
- The verifier must accept proofs through the function `WhirR1CSScheme::verify` without an external check that binds challenge witness slots to transcript challenges.

### Technical Details

In the function `WhirR1CSScheme::verify`, after receiving `commitment_1`, the verifier samples `_logup_challenges` using the function `arthur.verifier_message_vec(self.num_challenges)` and discards the result. The verifier does not open or constrain the witness coordinates designated for challenge values, and does not incorporate the sampled challenges into any subsequent linear form, opening claim, or final consistency equation.

The type `WitnessBuilder::Challenge` represents a Fiat-Shamir challenge, and related constructions, including `IndexedLogUpDenominator`, `LogUpDenominator`, and `LogUpInverse`, are parameterized by challenge witness indices. If those witness slots are unconstrained, a prover can populate them with

arbitrary field elements and satisfy challenge-dependent lookup equations under attacker-chosen randomness while the verifier independently samples unused transcript challenges.

#### Remediation

We recommend binding the outputs of the function `arthur.verifier_message_vec(self.num_challenges)` to the proved instance by adding verification constraints that open the `WitnessBuilder::Challenge` coordinates and compare them against the verifier-derived challenges, or by removing challenge values from the witness and recomputing challenge-dependent terms on the verifier side from the transcript challenges.

#### Status

The World team has [resolved](#) the issue as recommended.

#### Verification

Resolved.

## Issue H: Unchecked Witness-Derived Indices Cause Deterministic Prover Panics in RAM and Multiplicity Builders

#### Location

[provekit/prover/src/witness/ram.rs](#)

[provekit/prover/src/witness/witness\\_builder.rs](#)

#### Synopsis

Witness-derived indices are used without bounds checks, which allows malformed witness values to deterministically panic the prover.

#### Impact

Low.

An attacker could cause denial of service in proof generation by triggering out-of-bounds panics in RAM and multiplicity witness builders.

#### Feasibility

High.

The attack is straightforward when an attacker can submit or influence witness inputs that reach the functions `solve_ram_witness`, `MultiplicitiesForRange`, or `MultiplicitiesForBinOp`.

#### Severity

Medium.

#### Preconditions

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

- The prover must process attacker-influenced inputs that reach the affected witness builders.
- The prover must treat panics as process or request failures that impact availability.

#### Technical Details

The function `solve_ram_witness` derives an address index from `addr.into_bigint().0[0]` as `usize` and uses it to index `rt_final` and `rv_final` without validating the address range, which triggers an out-of-bounds panic for oversized addresses.

In the witness builder `MultiplicitiesForRange`, the statement `multiplicities[value as usize] += 1` assumes `value < range_size`, and in `MultiplicitiesForBinOp`, the variable `index`, computed as `(lhs_limb << atomic_bits) + rhs_limb`, is used as an unchecked table index, which allows adversarial limbs to deterministically panic the prover.

#### Remediation

We recommend adding explicit bounds checks for each witness-derived index and returning structured `Result` errors from the affected builders rather than panicking.

#### Status

The World team has [resolved](#) the issue as recommended.

#### Verification

Resolved.

## Issue I: Fiat-Shamir Transcript Does Not Explicitly Bind the Verification Key / Concrete R1CS

#### Location

[provekit/verifier/src/whir\\_r1cs.rs#L44](#)

[provekit/verifier/src/whir\\_r1cs.rs#L221](#)

[src/protocols/whir\\_zk/verifier.rs#L62](#)

[provekit/verifier/src/whir\\_r1cs\\_scheme.rs#L18](#)

#### Synopsis

The verifier samples Fiat-Shamir challenges before absorbing a digest of the concrete verification key material, including the R1CS encoded in the verifier-held `.pkv`. As a result, the transcript is not explicitly bound to the exact circuit or verification key instance being verified.

#### Impact

Medium.

If verification-key selection or material becomes attacker-influenced in a multi-key or dynamic-key context, the absence of transcript-level statement binding could enable cross-circuit or cross-key confusion that degrades soundness.

#### Feasibility

Low.

Exploitation would require the attacker to influence verification-key selection, substitute verification-key material, or otherwise place the verifier into a multi-key or multi-circuit context where lack of transcript binding becomes security-relevant.

### Severity

Low.

### Preconditions

For this issue to become security-relevant, at least one of the following must hold true:

- The verifier must accept attacker-influenced `.pkv` or verification-key selection.
- The verifier must support multiple circuits or dynamically chosen verification keys.
- Verification-key material must be loaded from an untrusted or insufficiently authenticated source.
- A substitute verification key or circuit must remain compatible with the verifier-side configuration and the later algebraic checks.

### Technical Details

In `WhirR1CSVerifier::verify`, the transcript is initialized with:

```
let ds = self.create_domain_separator().instance(&Empty);
```

This initializes the Fiat-Shamir transcript without absorbing a digest of the concrete R1CS or the `.pkv` contents. The verifier then samples early challenges through `verifier_message_vec` and `verifier_message` before any explicit circuit-dependent digest is committed into the transcript. In particular, `run_sumcheck_verifier` samples `r`, `rho`, and `alpha` from the transcript before the later R1CS-dependent computations are enforced.

Similarly, in `Config::verify`, the values `blinding_challenge`, `masking_challenge`, `tau1`, and `tau2` are sampled from the transcript before the later checks over circuit-derived weights and claims are completed.

The builder code in `WhirR1CSScheme::new_for_r1cs` derives verifier parameters such as `m_0`, `m`, and `a_num_terms` from the verifier-side `r1cs`, which strengthens coupling between the verifier configuration and the circuit shape. However, this does not change the transcript behavior itself: the Fiat-Shamir transcript still does not explicitly commit to a collision-resistant digest of the exact verification key or R1CS before challenge generation.

Accordingly, statement binding relies on external verifier-side key management rather than on transcript-level binding of the exact verification object.

### Remediation

We recommend absorbing a collision-resistant digest of the concrete verification key material before any call to `verifier_message` or `verifier_message_vec` in both the proving and verifying paths. This digest should cover the exact `.pkv` contents or, at minimum, the concrete R1CS and all verifier-relevant parameters derived from it. This would explicitly bind the Fiat-Shamir transcript to the statement being proven and verified.

### Status

The World team has [resolved](#) the issue as recommended.

### Verification

Resolved.

## Issue J: Inconsistent Equality Polynomial Length Omits R1CS Constraints

### Location

[provekit/common/src/utils/sumcheck.rs#L116](#)

[provekit/common/src/utils/sumcheck.rs#L214](#)

### Synopsis

The verifier omits constraints from the weighted row aggregation when the equality polynomial input length does not cover the full R1CS row domain.

### Impact

Medium.

If a verifier processes an R1CS with more constraints than the alpha-indexed row domain, trailing constraints could receive zero weight and be excluded from checks, which could permit acceptance of an invalid statement.

### Feasibility

Low.

Exploitation requires control over verifier inputs, along with a parameterization mismatch between the variable alpha and the supplied R1CS.

### Severity

Low.

### Preconditions

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

- The verifier must accept an R1CS with more than `1usize << alpha.len()` constraints.
- The function `multiply_transposed_by_eq_alpha` must process that R1CS without rejecting the mismatch.
- The attacker must influence the supplied R1CS or the verifier configuration.

### Technical Details

The function `calculate_evaluations_over_boolean_hypercube_for_eq` computes `full_size = 1usize << r.len()` and allocates the output vector with length `num_entries`. The bound `num_entries <= full_size` is checked only with `debug_assert!`, so release builds permit oversized inputs. As a result, the function `eval_eq` fills only the first logical `full_size` leaves and leaves the remaining entries of the returned vector at zero.

The function `multiply_transposed_by_eq_alpha` passes `r1cs.num_constraints()` as `num_entries` and multiplies the hydrated transposed matrices by the resulting slice. Any row at an index greater than or equal to `1usize << alpha.len()` therefore receives a zero coefficient in the aggregation, which permits appended failing constraints to be omitted when the verifier accepts an inconsistent pairing of alpha and the R1CS. The function `new_for_r1cs` sizes the variable `m_0` from `r1cs.num_constraints()`, but access to the function `verify` is required to determine whether external callers can still trigger the mismatch.

### Remediation

We recommend replacing the `debug_assert!` in the function `calculate_evaluations_over_boolean_hypercube_for_eq` and using a release-checked validation that rejects any `num_entries` greater than `1usize << r.len()` before matrix aggregation.

### Status

The World team has [resolved](#) the issue as recommended.

### Verification

Resolved.

## Issue K: Unbound prover\_hint\_ark Values Leave a Latent Transcript-Binding Bug

### Location

[provekit/provekit/prover/src/whir\\_r1cs.rs#L166](#)

[provekit/provekit/verifier/src/whir\\_r1cs.rs#L117](#)

[whir/src/transcript/mod.rs#L214](#)

### Synopsis

The transcript API accepts prover\_hint\_ark values without absorbing them into the Fiat-Shamir transcript state.

### Impact

Low.

A prover could present verifier-consumed hinted values that are not bound to later challenges, which could enable proof soundness degradation if future protocol logic treats those values as transcript-bound.

### Feasibility

Medium.

An attacker with the ability to produce proofs could exploit this issue if a caller places security-relevant claims in prover\_hint\_ark and assumes transcript binding without additional verification.

### Severity

Low.

### Preconditions

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

- A proving or verification flow must use the function `ProverState::prover_hint_ark` for values that influence later Fiat-Shamir challenges or acceptance conditions.
- A caller must assume that prover\_hint\_ark values are bound to the transcript in the same manner as transcript messages.
- Downstream algebraic checks that currently reject malformed hinted claims must be absent, bypassed, or refactored.

### Technical Details

The function `ProverState::prover_hint_ark` writes serialized values into the hints buffer without absorbing them into the duplex sponge, and the function `VerifierState::prover_hint_ark` reads from the same buffer without affecting challenge generation. As a result, the verifier challenge stream derived from the Fiat-Shamir sponge remains independent of any values transported through this hint channel.

In the current WHIR R1CS integration, the function `WhirR1CSScheme::prove` transmits evaluation claims, including `evals`, `evals_1`, `evals_2`, and `public_eval`, via `prover_hint_ark`, and the function `WhirR1CSScheme::verify` consumes them as inputs to zkWHIR verification. A release-mode forged proof attempt that substituted  $(A, B, C)$  with  $(A + 1, B, C + B)$  while preserving the outer relation  $A * B - C$  still fails verification, which indicates that current downstream checks provide binding in this specific flow, but the transcript boundary remains non-binding for hinted values.

#### Remediation

We recommend replacing security-relevant uses of the function `ProverState::prover_hint_ark` with transcript-bound messages or absorbing the hinted values into the sponge state on both prover and verifier sides.

#### Status

The World team has [resolved](#) the issue as recommended.

#### Verification

Resolved.

## Issue L: zkWHIR Batch Proofs Accept Alpha-Cancelled Forged Evaluations

#### Location

[src/protocols/whir\\_zk/verifier.rs](#)

[src/protocols/whir\\_zk/prover.rs](#)

#### Synopsis

When zkWHIR is used with more than one committed witness polynomial, the verifier collapses the claimed public evaluations of each form into a single alpha-weighted sum and never rebinds the individual per-polynomial entries, which allows a malicious prover to alter those entries while preserving the aggregate.

#### Impact

High.

A malicious prover can cause the verifier to accept forged per-polynomial public evaluations that are not consistent with the committed witnesses, which breaks the soundness of the proof system.

#### Feasibility

High.

Given the preconditions, the attack reduces to a constant-time algebraic adjustment of two evaluations after reading the sampled `alpha_coeffs` from the transcript.

#### Severity

Critical.

#### Preconditions

For this issue to occur, the following must hold:

- zkWHIR must be configured with `batch_size > 1`.
- The caller must not absorb the full row-major public evaluations matrix into `DomainSeparator::instance(...)` before the verifier samples `alpha_coeffs`.

### Technical Details

The verifier samples the batching scalar `alpha_coeffs` and reduces every public row to  $\text{combined\_eval\_j} = \sum_i \alpha_i * \text{evaluations}[j, i]$ . The prover mirrors the same batching. After this collapse, later phases operate only on the alpha-batched witness-side statement through  $\text{batching\_weights} = \text{tensor\_product}(\alpha\_coeffs, eq\_weights)$  and the Step-7 diagonal Lambda checks, none of which reintroduce the original per-polynomial claim  $\langle w_j, f_i \rangle$  as a standalone value.

After reading the sampled alpha from the transcript, the prover can substitute, for two batched polynomials,  $e'_0 = e_0 + \Delta$  together with  $e'_1 = e_1 - \Delta / \alpha$  for any non-zero alpha, which preserves the alpha-weighted row sum and, hence, `combined_eval`.

### Remediation

We recommend absorbing the full row-major public evaluations matrix into the Fiat-Shamir transcript inside the zkWHIR verifier before sampling `alpha_coeffs` and `rho`, instead of leaving statement binding as the undocumented responsibility of the `DomainSeparator::instance(...)` caller.

### Status

The World team has [resolved](#) the issue as recommended by absorbing the public evaluations array into the Fiat-Shamir transcript as prover messages before any of the affected challenges are sampled.

### Verification

Resolved.

## Issue M: Unbound Step-2 $G_j$ Claims Allow Rho-Adaptive Forged Evaluations

### Location

[src/protocols/whir\\_zk/verifier.rs#L131-L162](#)

[src/protocols/whir\\_zk/prover.rs#L170-L191](#)

### Synopsis

The zkWHIR verifier accepts the prover-sent Step-2 linear-form claims  $G_j$  only through the combined value  $\rho * \text{eval} + G_j$  and never rebinds them as standalone claims, which allows a malicious prover to compensate a forged  $G'_j$  with a rho-adaptive forged public evaluation.

### Impact

High.

A malicious prover can cause the verifier to accept a forged public evaluation  $e'_j$  that is not consistent with the committed witness, which breaks the soundness of the proof system.

### Feasibility

High.

Given the preconditions, the exploit consists of perturbing the Step-2 message by a known offset and then computing a single modular inverse of the sampled `rho`.

### Severity

Critical.

### Preconditions

The caller must not absorb the full public statement, including the claimed evaluations, into `DomainSeparator::instance(...)` before the verifier samples  $\rho$ .

### Technical Details

The prover sends  $G_j = \langle w_j, g \rangle$ , and the verifier uses those values only to build  $\text{combined\_claim}_j = \rho * \sum_i \alpha_i * \text{evaluations}[j, i] + G_j$ , after which Step 4 immediately collapses the result into  $\text{the\_sum}$ . The later Gamma consistency check and the Step-7 diagonal checks verify only query-time blinding openings, and no later step proves that the original Step-2  $G_j$  transcript messages are the linear-form claims implied by the committed blinding polynomials.

A malicious prover can submit a forged  $G'_j = G_j + \text{delta}$ , wait for the transcript to sample  $\rho$ , and then claim a forged public evaluation  $e'_j = e_j - \text{delta} / \rho$ , which preserves the combined relation  $\rho * e'_j + G'_j = \rho * e_j + G_j$ .

### Remediation

We recommend absorbing the full public statement, including the claimed evaluations, into the Fiat-Shamir transcript inside the zkWHIR verifier before sampling  $\rho$ , instead of leaving statement binding as the undocumented responsibility of the `DomainSeparator::instance(...)` caller.

### Status

The World team has [resolved](#) the issue as recommended by absorbing the public evaluations array into the Fiat-Shamir transcript as prover messages before any of the affected challenges are sampled.

### Verification

Resolved.

## Issue N: zkWHIR Multi-Form Proofs Accept Form-RLC-Preserving Forged Evaluations

### Location

[src/protocols/whir\\_zk/verifier.rs#L149-L167](#)

[src/protocols/whir\\_zk/verifier.rs#L381-L387](#)

### Synopsis

When zkWHIR is used with more than one public linear form, the verifier collapses the per-form claims into a single random linear combination and never rebinds each public evaluation individually, which allows a malicious prover to reuse one proof under a different evaluation vector that preserves the form RLC.

### Impact

High.

A malicious prover can cause the verifier to accept a forged public evaluation vector that is not consistent with the committed witness, which breaks the soundness of the proof system.

### Feasibility

High.

Given the preconditions, the attack reduces to replaying the verifier's transcript prefix to read `constraint_rlc_coeffs` and applying one scalar adjustment per form.

#### Severity

Critical.

#### Preconditions

For this issue to occur, the following must hold:

- zkWHIR must be configured with `num_forms > 1`.
- The caller must not absorb the full public statement, including the claimed evaluations, into `DomainSeparator::instance(...)` before the verifier samples `constraint_rlc_coeffs`.

#### Technical Details

After Step 3, the verifier forms one `combined_claim_j = rho * eval_j + G_j` per public linear form, and Step 4 collapses those per-form claims into `the_sum = Σ_j constraint_rlc_coeffs[j] * combined_claim_j`. The later post-sumcheck check compares only the RLC value `Σ_j constraint_rlc_coeffs[j] * w_j(point)` against `linear_form_rlc`, and Step 7 verifies only blinding-vector openings, so the original per-form public evaluations are never checked individually.

After replaying the verifier transcript prefix to obtain `constraint_rlc_coeffs`, the prover picks offsets `delta_j` satisfying `Σ_j constraint_rlc_coeffs[j] * delta_j = 0`, for example, `(delta, -(c0 / c1) * delta)` when `c1 != 0`, and submits the forged evaluation vector `e'_j = e_j + delta_j` alongside the unchanged proof bytes.

#### Remediation

We recommend absorbing the full public evaluations vector into the Fiat-Shamir transcript inside the zkWHIR verifier before sampling `constraint_rlc_coeffs`, instead of leaving statement binding as the undocumented responsibility of the `DomainSeparator::instance(...)` caller.

#### Status

The World team has [resolved](#) the issue as recommended by absorbing the public evaluations array into the Fiat-Shamir transcript as prover messages before any of the affected challenges are sampled.

#### Verification

Resolved.

## Issue O: zkWHIR Proofs Replay Under Alternate Linear Forms Sharing the Final-Point MLE Value

#### Location

[src/protocols/whir\\_zk/verifier.rs#L113-L120](src/protocols/whir_zk/verifier.rs#L113-L120)

[src/protocols/whir\\_zk/verifier.rs#L381-L387](src/protocols/whir_zk/verifier.rs#L381-L387)

#### Synopsis

The zkWHIR verifier never binds the public linear forms into the Fiat-Shamir transcript and checks them only through a single random-point MLE equality, which allows a malicious prover to reuse one proof under a different form whose MLE agrees at the final evaluation point.

### Impact

High.

A malicious prover can cause the verifier to accept the false public statement  $\langle w', f \rangle = e$  for a committed witness  $f$ , which breaks the soundness of the proof system.

### Feasibility

High.

Given the preconditions, the exploit reduces to replaying the verifier transcript to derive the final `evaluation_point` and constructing a two-index correction to the form.

### Severity

Critical.

### Preconditions

The caller must not absorb the full public linear forms into `DomainSeparator::instance(...)` before the verifier samples any Fiat-Shamir challenge.

### Technical Details

The function `prepare_and_sumcheck` reads `weights` only to determine the number of forms by computing `num_forms = weights.len()`. The sole verifier-side use of the actual `weights` content is the post-sumcheck RLC check, which compares `expected_rlc =  $\sum \text{constraint\_rlc\_coeffs}[j] * \text{weights}[j].\text{mle\_evaluate}(\text{evaluation\_point})$`  against the proof-derived `linear_form_rlc`. Since `linear_form_rlc` depends only on the proof bytes, the verifier accepts any alternate form whose MLE at `evaluation_point` matches that of the honest form.

A prover can therefore substitute, with a covector  $w$ , the alternate form  $w' = w + \text{eq}(r, b) \cdot e_a - \text{eq}(r, a) \cdot e_b$  for any indices  $a$  and  $b$  such that  $\text{eq}(r, b) \cdot f[a] - \text{eq}(r, a) \cdot f[b] \neq 0$ . This yields  $w'.\text{mle\_evaluate}(r) = w.\text{mle\_evaluate}(r)$  while  $\langle w', f \rangle \neq \langle w, f \rangle$ . The attack applies to any `num_forms`  $\geq 1$ , including a single form, and is therefore strictly more general than the multi-form collapse described in [Issue N](#).

### Remediation

We recommend absorbing the serialized public linear forms into the Fiat-Shamir transcript inside the zkWHIR verifier before sampling any Fiat-Shamir challenge, instead of relying on the post-sumcheck single-point RLC as the sole statement-binding mechanism.

### Status

The World team has [resolved](#) the issue by documenting safe usage of `Config::verify` rather than implementing the recommended remediation.

### Verification

Resolved.

## Issue P: WHIR Batched Verifier Accepts Vector-RLC-Cancelled Forged Evaluations Across Commitments

### Location

<src/protocols/whir/verifier.rs#L52-L103>

### Synopsis

When WHIR is used in a batched configuration, the verifier collapses the public evaluations matrix into a single aggregate weighted by geometric coefficients and never rebinds the per-commitment entries, which allows a malicious prover to substitute any evaluation vector whose row dot product with the sampled geometric challenge matches the honest one.

### Impact

High.

A malicious prover can cause the verifier to accept forged per-commitment public evaluations that are not consistent with the committed vectors, which breaks the soundness of the proof system.

### Feasibility

High.

Given the preconditions, the attack reduces to replaying the verifier's transcript prefix to read the geometric challenge gamma and applying one scalar adjustment per row.

### Severity

Critical.

### Preconditions

For this issue to occur, the following must hold:

- WHIR must be used with `num_vectors > 1`, including multiple commitments, stacked commitments with `initial_committer.num_vectors > 1`, or both.
- The caller must not absorb the full row-major evaluations matrix into `DomainSeparator::instance(...)` before the verifier samples `vector_rlc_coeffs`.

### Technical Details

The verifier samples `vector_rlc_coeffs = geometric_challenge(verifier_state, num_vectors)` and then reduces each public row to `dot(vector_rlc_coeffs, row)` inside `the_sum`, which is the only point at which `evaluations[j * num_vectors + i]` enters the verification. The remaining phases, including the round-0 commitment, the STIR reconstruction, the sumcheck chain, and the deferred `FinalClaim::verify`, operate solely on the collapsed aggregate and the per-form RLC. No step rebinds any individual `e_{j,i}`.

A malicious prover who reads gamma from the transcript can, for `num_vectors = 2`, replace row `j` with  $(e_{j,0} + \Delta, e_{j,1} - \Delta / \text{gamma})$  when `gamma != 0`, which leaves the row dot product invariant. In the degenerate case `gamma = 0`, every entry `e_{j,1..n-1}` drops out of `the_sum` and can be replaced arbitrarily.

### Remediation

We recommend absorbing the full row-major public evaluations matrix into the Fiat-Shamir transcript inside the WHIR verifier before sampling `vector_rlc_coeffs`, instead of leaving statement binding as the undocumented responsibility of the `DomainSeparator::instance(...)` caller.

### Status

The World team has [resolved](#) the issue as recommended by absorbing the public evaluations array into the Fiat-Shamir transcript as prover messages before any of the affected challenges are sampled.

## Verification

Resolved.

## Issue Q: WHIR Proofs Replay Under Alternate Linear Forms Sharing the Final-Point MLE Value

### Location

[src/protocols/whir/verifier.rs#L38-L228](#)

[src/protocols/whir/mod.rs#L60-L82](#)

### Synopsis

The WHIR API leaves the public linear forms outside the Fiat-Shamir transcript and checks them only through the deferred single random-point MLE equality in `FinalClaim::verify`, which allows a malicious prover to reuse one proof under a different form whose MLE agrees at the final evaluation point.

### Impact

High.

A malicious prover can cause the verifier to accept the false public statement  $\langle w', f \rangle = e$  for a committed witness  $f$ , which breaks the soundness of the proof system.

### Feasibility

High.

Given the preconditions, the exploit reduces to replaying the verifier transcript to derive the final `evaluation_point` and constructing a two-index correction to the form.

### Severity

Critical.

### Preconditions

The caller must not absorb the full public linear forms into `DomainSeparator::instance(...)` before calling `whir::Config::verify`.

### Technical Details

The function `whir::Config::verify` takes commitments and evaluations as public inputs, not linear forms. The public forms are consumed only inside `FinalClaim::verify`, which compares  $rlc = \sum rlc\_coefficients[i] * linear\_forms[i].mle\_evaluate(self.evaluation\_point)$  against the proof-derived `linear_form_rlc`. Since `linear_form_rlc` is fixed by the proof bytes, any form whose MLE agrees at `evaluation_point` satisfies the check regardless of its evaluation on the committed witness.

For a single form, with `rlc_coefficients = [1]`, the prover can construct  $w' = w + \Delta$ , where  $\Delta.mle\_evaluate(r) = 0$  but  $\Delta(f) \neq 0$ . One example is the sparse two-index delta  $\Delta = eq\_r[j] \cdot e_i - eq\_r[i] \cdot e_j$ , which preserves the final-point MLE while altering the actual linear-form value. This attack applies to any linear-form count, including a single form, and operates without modifying the proof bytes.

### Remediation

We recommend absorbing the serialized public linear forms into the Fiat-Shamir transcript inside the base WHIR verifier before sampling any Fiat-Shamir challenge, instead of relying on the deferred single-point RLC in `FinalClaim::verify` as the sole statement-binding mechanism.

### Status

The World team has [resolved](#) the issue by documenting safe usage of `Config::verify` rather than implementing the recommended remediation.

### Verification

Resolved.

## Issue R: WHIR Multi-Form Proofs Accept Form-RLC-Preserving Forged Evaluations

### Location

[src/protocols/whir/verifier.rs#L86-L103](#)

[src/protocols/whir/mod.rs#L72-L83](#)

### Synopsis

When WHIR is used with more than one public linear form, the verifier collapses the claimed evaluations into a single random linear combination scalar and never rebinds each evaluation individually, which allows a malicious prover to reuse one proof under a different evaluation vector that preserves the form RLC.

### Impact

High.

A malicious prover can cause the verifier to accept a forged public evaluation vector that is not consistent with the committed witness, which breaks the soundness of the proof system.

### Feasibility

High.

Given the preconditions, the attack reduces to replaying the verifier transcript prefix to read `constraint_rlc_coeffs` and applying one scalar adjustment per form.

### Severity

Critical.

### Preconditions

For this issue to occur, the following must hold:

- WHIR must be configured with `num_linear_forms >= 2`.
- The caller must not absorb the full public evaluations vector into `DomainSeparator::instance(...)` before the verifier samples `constraint_rlc_coeffs`.

### Technical Details

The verifier samples `constraint_rlc_coeffs = geometric_challenge(verifier_state, oods_evals.len() + num_linear_forms)` and folds the public evaluation rows into `the_sum` through the slice `initial_form_rlc_coeffs`, with row `j` contributing `c_j * the_sum[j]`.

`dot(vector_rlc_coeffs, row_j)`, without absorbing evaluations into the Fiat-Shamir sponge. After the sumcheck chain, the deferred `FinalClaim::verify` compares only the RLC of the linear-form MLEs at a single random point against the proof-derived `linear_form_rlc`, so no later step rechecks each original public evaluation row or scalar entry.

In the single-vector case, for two forms with coefficients  $[1, c_1]$  and nonzero  $c_1$ , the prover replaces  $(e_0, e_1)$  with  $(e_0 + \Delta, e_1 - \Delta / c_1)$ , which preserves  $\sum_i c_i * \text{evaluations}[i]$  and, hence, `the_sum`. In the degenerate case  $c_1 = 0$ , the second evaluation drops out of the checked relation and can be replaced arbitrarily.

#### Remediation

We recommend absorbing the full public evaluations vector into the Fiat-Shamir transcript inside the base WHIR verifier before sampling `constraint_rlc_coeffs`, instead of leaving statement binding as the undocumented responsibility of the `DomainSeparator::instance(...)` caller.

#### Status

The World team has [resolved](#) the issue as recommended by absorbing the public evaluations array into the Fiat-Shamir transcript as prover messages before any of the affected challenges are sampled.

#### Verification

Resolved.

## Issue S: Proof Malleability Through Trailing Bytes and Non-Canonical Field Element Encodings

#### Location

[src/transcript/mod.rs#L283-L300](#)

[src/protocols/whir/verifier.rs#L38-L228](#)

[src/protocols/whir\\_zk/verifier.rs#L598-L622](#)

[spongefish/src/drivers/ark\\_ff\\_impl.rs#L44-L66](#)

[spongefish/src/narg\\_verifier.rs#L35-L41](#)

#### Synopsis

The public verifier entry points `whir::Config::verify` and `whir_zk::Config::verify` never invoke `VerifierState::check_eof`, and the field-element deserializer they rely on reduces input bytes modulo the field prime instead of rejecting noncanonical encodings. Together, these behaviors allow a malicious prover to produce unboundedly many distinct byte-level encodings of a proof that all verify the same semantic claim.

#### Impact

Medium.

A malicious prover can produce unboundedly many distinct byte-level encodings of a proof that all verify the same claim, which can bypass deduplication, identifier-based indexing, or signature-binding guarantees in downstream consumers.

#### Feasibility

Medium.

The trailing-byte vector reduces to appending arbitrary bytes to the `narg_string` and `hints` buffers of a valid Proof and succeeds unconditionally. The in-place field-element vector succeeds with probability approximately  $2^{-32}$  per prover-message element on the shipped Goldilocks field and becomes deterministic on any field whose modulus bit-length is strictly less than  $8 \cdot \text{base\_field\_size}$ . Obtaining practical value from either vector requires a downstream consumer that treats raw proof bytes as a canonical per-claim identifier.

### Severity

Medium.

### Preconditions

A downstream consumer must treat the raw Proof bytes, namely `narg_string` and `hints`, as a canonical per-claim identifier. For example, the consumer must use those bytes as a deduplication key, a transaction identifier, or as input to a commit-and-reveal construction.

### Technical Details

The verifier exposes two orthogonal malleability surfaces on the Proof byte string. Neither `whir::Config::verify` nor `whir_zk::Config::verify` calls `VerifierState::check_eof` before returning, so trailing bytes in `narg_string` and `hints` are silently accepted. The external `Spongefish` deserializer calls `from_le_bytes_mod_order(chunk)` for field-element prover messages. It then absorbs the canonical reencoding into the sponge, which makes the sponge state invariant under noncanonical input while leaving the raw `narg_string` unchanged from the byte sequence provided by the prover.

A malicious prover can append arbitrary bytes to `narg_string` and `hints`. Alternatively, the prover can substitute the canonical encoding of any value  $v$  satisfying  $v + p < 2^{8 \cdot \text{base\_field\_size}}$  with the noncanonical encoding of  $v + p$ . In either case, the mutated proof still returns `Ok(FinalClaim)` with identical contents. The two vectors compose multiplicatively, yielding  $2^k$  distinct accepting proof byte strings per semantic claim when  $k$  malleable field-element spans are cross-multiplied with arbitrary trailing suffixes.

### Remediation

We recommend invoking `VerifierState::check_eof` at the end of each top-level `verify` and replacing the `from_le_bytes_mod_order` call inside `deserialize_from_narg` with a range check that rejects any chunk whose little-endian integer is at or above the field modulus.

### Status

The World team has proposed deferring this issue, as it does not introduce any significant security risk in their current usage.

- Vector 1 (`check_eof` in `verifier`): `check_eof` is enforced at the Provekit level and considered a caller responsibility. Implementing it within WHIR would require significant API and transcript ownership changes.
- Vector 2 (non-canonical field elements in `sponge`): The team has acknowledged this issue and will address it in a future update. Newer versions of the `spongefish` library already address it, with an upgrade planned in WHIR 3.0, which is targeted for Q2.

### Verification

Unresolved.

## Issue T: WHIR Verifier Panics on Unchecked Divisions by Zero

### Location

[src/protocols/whir/verifier.rs#L61-L67](#)

[src/protocols/whir/verifier.rs#L236-L239](#)

[src/protocols/whir\\_zk/verifier.rs#L365-L368](#)

### Synopsis

The `whir::Config::verify` entry point performs two unchecked divisions by zero. It first computes the integer division `evaluations.len() / num_vectors` before the `num_vectors == 0` early-return guard. Later, it computes the field division `the_sum / poly_eval` without first checking whether `poly_eval != F::ZERO`. Either condition causes the verifier process to panic instead of returning `Err(VerificationError)`.

### Impact

Medium.

A caller with empty inputs or a malicious prover committing to the all-zero witness vector can abort a deployed verifier with an unhandled panic instead of receiving a clean verification error.

### Feasibility

Medium.

Both panics fire deterministically on their respective triggers and require no cryptographic effort.

### Severity

Medium.

### Preconditions

For this issue to occur, the following must hold:

- A caller must pass empty commitments and evaluations slices, or construct the `Config` with `ProtocolParameters.batch_size = 0`.
- A malicious prover must commit to the all-zero witness vector.

### Technical Details

The WHIR verifier computes `evaluations.len() / num_vectors` before the `num_vectors == 0` early return, so empty input triggers a native integer division by zero. It then computes `the_sum / poly_eval` without a `verify!(poly_eval != F::ZERO)` guard. (The analogous zkWHIR verifier already includes that guard). A zero witness passes the in-domain check and drives `poly_eval` deterministically to zero, causing `ark_ff's Field::Div` to panic through `Option::unwrap()` on zero.

### Remediation

We recommend moving the `num_vectors == 0` early return before the integer division and adding a `verify!(poly_eval != M::Target::ZERO)` guard before the field division.

### Status

The World team has [resolved](#) the issue as recommended.

## Verification

Resolved.

## Suggestions

### Suggestion 1: Implement Full PKI Checks for Passports

#### Location

[noir-examples/noir-passport/utils/data-check/tbs-pubkey/src/lib.nr#L1-L16](#)

[noir-examples/noir-passport/merkle\\_age\\_check/t\\_add\\_dsc\\_720/src/main.nr#L22-L33](#)

[noir-examples/noir-passport/merkle\\_age\\_check/t\\_add\\_dsc\\_verify\\_1300/src/main.nr#L65-L74](#)

[noir-examples/noir-passport/merkle\\_age\\_check/t\\_add\\_id\\_data\\_720/src/main.nr#L22-L34](#)

[noir-examples/noir-passport/merkle\\_age\\_check/t\\_add\\_id\\_data\\_1300/src/main.nr#L21-L34](#)

[noir-examples/noir-passport/utils/sig-check/rsa/src/lib.nr#L11-L21](#)

[noir-examples/noir-passport/merkle\\_age\\_check/t\\_attest/src/main.nr#L79-L86](#)

[noir-examples/noir-passport-monolithic/complete\\_age\\_check/src/main.nr#L17-L21](#)

[playground/passport-input-gen/src/lib.rs#L201-L300](#)

#### Synopsis

Currently, the noir-passport circuit is in an early development stage. To complete the PKI requirements for passports under [ICA0 Doc-9303](#), the following should be addressed:

- The proof does not verify that the CSCA key is anchored in a trusted master list or trust store.
- The proof does not bind the CSCA key to the claimed issuing country.
- The proof does not bind the full DSC public key encoded in the certificate.
- The proof checks only that the DSC modulus bytes appear at a supplied offset.
- The proof does not parse ASN.1/X.509 to prove that the matched bytes correspond to the subjectPublicKeyInfo.
- The proof does not verify certificate issuer/subject chain linkage.
- The proof does not verify Subject Key Identifier and Authority Key Identifier consistency.
- The proof does not verify BasicConstraints.
- The proof does not verify KeyUsage.
- The proof does not verify ExtendedKeyUsage.
- The proof does not verify certificate policies.
- The proof does not verify name constraints.
- The proof does not verify path length constraints.
- The proof does not verify DSC certificate validity dates (notBefore / notAfter).
- The proof does not verify CSCA certificate validity dates (notBefore / notAfter).
- The proof does not verify certificate revocation status via CRLs or OCSP.

- The proof does not prove compliance with the full eMRTD PKI profile; it proves only signature consistency over supplied inputs.

#### Mitigation

We suggest implementing the checks described above. Specific solutions cannot be recommended because implementation can vary.

#### Status

The World team has acknowledged the gap and provided the following responses:

- CSCA keys will be anchored via a Merkle tree in future updates.
- TBS certificate matching is already verified by the circuit, so a malicious prover cannot exploit offsets if the certificate is valid.
- CRL checks are better handled at the application or integration layer. The World team will coordinate with the integration team to fetch and maintain CRLs.
- The remaining X.509 checks, including issuer and subject linkage, SKI, AKI, BasicConstraints, and KeyUsage, are not applicable because they exist for policy enforcement in variable-depth PKI.

#### Verification

Unresolved.

## Suggestion 2: Improve Code Comments

#### Synopsis

There are insufficient code comments explaining the rationale behind certain critical lines of code. This reduces the readability of the code and, as a result, makes reasoning about the security of the system more difficult. Comprehensive in-line documentation helps provide reviewers of the code with a better understanding and ability to reason about the system design.

#### Mitigation

We recommend expanding and improving the code comments to facilitate reasoning about the security properties of the system.

#### Status

The World team has not addressed this suggestion.

#### Verification

Unresolved.

## Suggestion 3: Reject Malformed 32-Bit Constants to Prevent Proving a Different Statement Than the Noir Artifact Specifies

#### Location

[provekit/r1cs-compiler/src/noir\\_to\\_r1cs.rs#L325-L418](#)

[provekit/r1cs-compiler/src/binops.rs#L72-L91](#)

[acir/src/circuit/opcodes/black\\_box\\_function\\_call.rs#L73-L80](#) (out of scope)

### Synopsis

The provekit R1CS compiler assumes Noir/ACIR has already validated that 32-bit black-box constants fit within 32 bits. For AND and XOR, provekit does not enforce that invariant locally and instead truncates constants to their low 32 bits during compilation. If malformed ACIR were ever admitted, two distinct constants that differ only above bit 31 would compile to the same relation. The SHA-256 constant path has a related issue, but in the current stack it is more accurately described as inconsistent constant handling than as a straightforward silent miscompile.

These behaviors represent a latent bug but not a vulnerability under the current threat model and the project's intended usage.

### Mitigation

We recommend revalidating 32-bit width assumptions locally in provekit before compiling AND and XOR operations.

### Status

The World team has eliminated silent truncation in [this PR](#) by asserting that the constant fits in 32 bits before any `u32` or `to_le_bytes()` conversion.

It also guards the digit extraction used by combined binop constraints, so malformed AND/XOR constants no longer collapse to the same 32 bit relation.

The PR also fixes the related SHA-256 constant truncation through validation.

### Verification

Resolved.

## Suggestion 4: Refine Computation of Reported Security Level

### Location

[src/protocols/irs\\_commit.rs#L253-L263](#)

[src/protocols/challenge\\_indices.rs#L43-L47](#)

[src/protocols/whir/config.rs#L197-L228](#)

### Synopsis

The function `rbr_queries` multiplies `self.in_domain_samples` by the per-sample round-by-round soundness contribution unconditionally and does not read `self.deduplicate_in_domain`, while `challenge_indices` applies `sort_unstable` and `dedup` when the flag is set. When a caller sets `deduplicate_in_domain` to `true`, the actual distinct query count drops below `in_domain_samples` on any collision, and the reported security level from `Config::security_level` and the `Display` implementation overstates the actual round-by-round soundness. As a result, an integrator sizing a deployment against the printed value may underprovision. The flag defaults to `false` in `irs_commit::Config::new` and no shipped configuration enables it, so the mismatch is an accounting discrepancy rather than a security flaw.

### Mitigation

We recommend replacing the nominal `in_domain_samples` factor in `rbr_queries` with a deduplication-aware unique query count when `deduplicate_in_domain` is set.

**Status**

The World team stated that it will consider this mitigation as part of upcoming protocol changes.

**Verification**

Unresolved.

# 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.