Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

RISC ZERO

risc0-solana

Veridise Inc.

Sep 26, 2024

► **Prepared For:**

RISC Zero
https://risczero.com/

► **Prepared By:**

Alp Bassa
Jon Stephens
Mark Anthony
Evgeniy Shishkin

► **Contact Us:**

contact@veridise.com

► **Version History:**

Sep. 16, 2024 V1
Sep. 26, 2024 V2

Contents

1 Executive Summary

From Sep. 4, 2024 to Sep. 6, 2024, RISC Zero engaged Veridise to review the security of their risc0-solana library. The review covered a library which provides functionality for utilizing and verifying RISC Zero proofs on the Solana blockchain. Veridise conducted the assessment over 9 person-days, with 3 engineers reviewing code over 3 days on commit `84a1929`. The auditing strategy involved an extensive manual code review performed by Veridise engineers.

Project summary. The security assessment covered the risc0-solana library which implements a Groth16[1] verifier that can be used to verify RISC Zero proofs on the Solana blockchain. The verifier is instantiated by passing the proof, the public inputs and the verification key as inputs. The proof can then be verified by performing pairing computations and validating the result.

The library also contains a non-solana module which includes functionality for serialization/de-serialization of the proof and the verification key, methods for point compression and decompression and utilities for negating G1 points among other utilities. This module is largely meant for off-chain operations and testing purposes.

Code assessment. The risc0-solana developers provided the source code of the risc0-solana Groth16 proof verifier for review. The source code appears to be mostly original code written by the risc0-solana developers. It contains some documentation in the form of a README, but the Veridise auditors noted that documentation comments on functions and storage variables were missing. To facilitate the Veridise auditors' understanding of the code, the risc0-solana developers shared an end to end example implementation which depicts how a proof is prepared and then verified within a Solana program using the risc0-solana library.

The source code contained a test suite, which the Veridise auditors noted tested sunny day scenarios for proof verification as well as most of the helper utilities.

Summary of issues detected. The audit uncovered 8 issues, 5 of which are assessed to be of a warning severity by the Veridise auditors. Specifically, V-RSL-VUL-003 details how the preparation of public inputs can fail if a mistake is made in the configuration of the public inputs or the verification key, V-RSL-VUL-004 explains how only the last byte of the pairing output being checked can be problematic if the method `alt_bn128_pairing` is ever replaced. The Veridise auditors also identified 3 informational findings which include V-RSL-VUL-007 that mentions how the point negation behaviour is not intuitive and should be well documented.

RISC Zero has fixed all of the identified issues.

Recommendations. After auditing the protocol, the auditors had a few suggestions to improve risc0-solana. The auditors recommend adding functional level documentation to better facilitate the understanding of future users of the library. The auditors also suggest adding negative tests to strengthen the testing suite. As a word of caution, the auditors recommend being vigilant with compiler flags, and to always have overflow checks enabled.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
risc0-solana	`84a1929`	Rust	Solana

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Sep. 4–Sep. 6, 2024	Manual	3	9 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Acknowledged	Fixed
Critical-Severity Issues	0	0	0
High-Severity Issues	0	0	0
Medium-Severity Issues	0	0	0
Low-Severity Issues	0	0	0
Warning-Severity Issues	5	5	5
Informational-Severity Issues	3	3	3
TOTAL	8	8	8

Table 2.4: Category Breakdown.

Name	Number
Maintainability	3
Data Validation	3
Usability Issue	2

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of risc0-solana's Groth16 proof verifier. In our audit, we sought to answer questions such as:

- Is the proof verification performed correctly?
- Is point representation consistent and unambiguous?
- Are appropriate validity checks applied to the public inputs and other group elements?
- Is the serialization and de-serialization of the group elements and the proof performed correctly?
- Is the Solana alt_bn128 library used correctly?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a thorough manual review of the protocol by human experts.

Scope. The scope of this audit is limited to the `src\lib.rs` file of the source code provided by the risc0-solana developers, which contains the Groth16 proof verifier of risc0-solana. The library also makes frequent use of the Solana `alt_bn_128` library which the Veridise auditors noted was out of scope for this audit.

Methodology. Veridise auditors inspected the provided tests, and went through the example implementation provided by the risc0-solana developers. They then began a manual review of the code. Before the audit, the Veridise auditors met with the risc0-solana developers to get an overview of the library and ask questions.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-RSL-VUL-001	Two step verification process is prone to . . .	Warning	Fixed
V-RSL-VUL-002	Pairing result check only applied to the . . .	Warning	Fixed
V-RSL-VUL-003	Length of public inputs not verified before . . .	Warning	Fixed
V-RSL-VUL-004	Missing input verification on point conversion	Warning	Fixed
V-RSL-VUL-005	Missing data validation on public inputs	Warning	Fixed
V-RSL-VUL-006	Unused code	Info	Fixed
V-RSL-VUL-007	Point negation behaviour not clearly . . .	Info	Fixed
V-RSL-VUL-008	Unrelated methods can be moved to the . . .	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-RSL-VUL-001: Two step verification process is prone to mistakes

Severity	Warning	**Commit**	84a1929
Type	Usability Issue	**Status**	Fixed
File(s)	`src/lib.rs`		
Location(s)	See description		
Confirmed Fix At	a9bbde1		

This library can be used to verify a Groth16 proof. The verification process happens in two steps. First, a new instance of the `Verifier` is instantiated by calling `new()`, and then `verify()` is called to perform the pairing and verify the proof. See the snippet below for the implementation of `verify()`.

The current process of initialisation and verification is prone to mistakes. For instance, a user of the library may call `new()` to initialize the `Verifier`, but may forget to call `verify()` to complete the verification process.

In similar projects, the construction of the `Verifier` and the verification of the proof tends to happen within the same function.

```
pub fn verify(&self) -> ProgramResult {
  let prepared_public = self.prepare_public_inputs()?;
  self.perform_pairing(&prepared_public)
}
```

Snippet 4.1: Snippet from `verify()`

Impact A user not familiar with the current verification process, may use the library incorrectly.

Recommendation Consider condensing the verification process into a single method. The public inputs, proof and the verification key can be passed into `verify()`.

4.1.2 V-RSL-VUL-002: Pairing result check only applied to the last byte of the pairing output

Severity	Warning	**Commit**	84a1929
Type	Maintainability	**Status**	Fixed
File(s)	`src/lib.rs`		
Location(s)	perform_pairing()		
Confirmed Fix At	51300b6		

The `perform_pairing` method takes in the prepared public inputs and performs a pairing by making use of the `alt_bn128_pairing` syscall from the Solana `alt_bn128` library, as shown in the snippet below.

```
let pairing_res =
    alt_bn128_pairing(&pairing_input).map_err(|_| Risc0SolanaError::PairingError)?;

if pairing_res[31] != 1 {
    return Err(Risc0SolanaError::VerificationError.into());
}
Ok(())
```

Snippet 4.2: Snippet from `perform_pairing()`

The result of a pairing is an arbitrary nonzero field element. But `alt_bn128_pairing` however, unlike the name suggests, does not return the result of the pairing computation. It returns `1` if the result is `1` and returns `0` otherwise, but does so as a 32 byte `BigInt`. In particular the return value is set as `0` but then corrected to `1` if the pairing computation gives `1`. See snippet below for context.

```
let mut result = BigInteger256::from(0u64);
let res = <Bn<Config> as Pairing>::multi_pairing(
    vec_pairs.iter().map(|pair| pair.0),
    vec_pairs.iter().map(|pair| pair.1),
);

if res.0 == ark_bn254::Fq12::one() {
    result = BigInteger256::from(1u64);
}

let output = result.to_bytes_be();
Ok(output)
```

Snippet 4.3: See snippet from `alt_bn128_pairing()`

Both the name (pairing) and the return type (`BigInt`) of `alt_bn128_pairing` are confusing, as it does not return the result of a pairing, but instead it returns `0` or `1` as a `BigInt`. This is then compared to the last byte to see if it is `0` or `1`. This is prone to issues should `alt_bn128_pairing` be replaced by something else in the future.

Impact The result of `alt_bn128_pairing` does not return the result of a pairing but instead returns a boolean. If the function is ever replaced with another without updating the implemented

pairing check accordingly, it can cause severe issues.

Recommendation Add in-line documentation which details the assumptions regarding the use of `alt_bn128_pairing`.

4.1.3 V-RSL-VUL-003: Length of public inputs not verified before iteration

Severity	Warning	**Commit**	84a1929
Type	Data Validation	**Status**	Fixed
File(s)	`src/lib.rs`		
Location(s)	prepare_public_inputs()		
Confirmed Fix At	02aa5db		

The method `prepare_public_inputs()` loops over the public inputs vector as shown in the snippet below.

Each loop iteration `i` also accesses the vector element `vk_ic[i+1][..]`. But the public inputs are iterated over without ensuring that the vector `vk_ic` has `public_inputs.len() + 1` elements. If that is not the case, then the preparation process and by extension the proof verification will fail.

As the public inputs are known and provided by the verifier, it may be less prone to mistakes. But, if this is used within a larger setup where the public inputs are sent again along with the proof or can be modified by the prover, it can be more prone to mistakes.

```
fn prepare_public_inputs(&self) -> Result<[u8; 64], Risc0SolanaError> {
    let mut prepared = self.vk.vk_ic[0];
    println!("Verification Key: {:?}", 0u8);
    for (i, input) in self.public.inputs.iter().enumerate() {
    let mul_res =
      alt_bn128_multiplication(&[&self.vk.vk_ic[i + 1][..], &input[..]].concat())
          .map_err(|_| Risc0SolanaError::ArithmeticError)?;
    prepared = alt_bn128_addition(&[&mul_res[..], &prepared[..]].concat())
      .unwrap()
      .try_into()
      .map_err(|_| Risc0SolanaError::ArithmeticError)?;
    }
    Ok(prepared)
}
```

Snippet 4.4: Snippet from `prepare_public_inputs()`

Impact If the lengths of the public inputs and the `vk_ic` do not match (as per the requirement), then the preparation of public inputs will not work and the proof verification will fail.

Recommendation Enforce that `self.public.inputs.len() + 1 == self.vk_ic.len()`.

4.1.4 V-RSL-VUL-004: Missing input verification on point conversion

Severity	Warning	**Commit**	84a1929
Type	Data Validation	**Status**	Fixed
File(s)	src/lib.rs		
Location(s)	convert_g1()		
Confirmed Fix At	83941eb		

The method `convertg1` takes in a vector `values` from which the points `pi_a`, `pi_b` and `pi_c` are parsed. See snippet below for context.

Let us have a look at how these points are represented. Within the struct `VerifyingKeyJson` , points on the elliptic curve are stored using homogeneous coordinates. So, rather than just two coordinates `x` and `y`, one has three coordinates `X`, `Y`, `Z`. These homogeneous coordinates correspond to the usual point `(x,y)` with `x=X/Z` and `y=Y/Z`. The representation in homogeneous coordinates is not unique, points obtained by scaling with a scalar are the same.

In this library, the `Z` coordinate is hardcoded as `1` which might be problematic. If someone can feed a verifying key where one of the points has a `Z` coordinate different from another `1`, it would be converted wrongly. The points would convert to `(X,Y)` rather than `(X/Z,Y/Z)`. Moreover if feeding in a point at infinity is possible, then it would be interpreted wrongly.

If the `Z` coordinate is always intended to be `1`, then it will not be an issue. But at the moment there are no checks in `convertg1` and `convertg2` which enforce this.

```
fn convert_g1(values: &[String]) -> Result<[u8; G1_LEN]> {
    if values.len() != 3 {
      return Err(anyhow!(
          "Invalid G1 point: expected 3 values, got {}",
          values.len()
      ));
    }

    let x = BigUint::parse_bytes(values[0].as_bytes(), 10)
      .ok_or_else(|| anyhow!("Failed to parse G1 x coordinate"))?;
    let y = BigUint::parse_bytes(values[1].as_bytes(), 10)
      .ok_or_else(|| anyhow!("Failed to parse G1 y coordinate"))?;

    let mut result = [0u8; G1_LEN];
    let x_bytes = x.to_bytes_be();
    let y_bytes = y.to_bytes_be();

    result[32 - x_bytes.len()..32].copy_from_slice(&x_bytes);
    result[G1_LEN - y_bytes.len()..].copy_from_slice(&y_bytes);

    Ok(result)
}
```

Snippet 4.5: Snippet from `convert_g1()`

Impact Hardcoding the `Z` coordinate as `1` can be problematic. If the `Z` coordinate passed as input is inconsistent, points may be converted wrongly and misinterpreted.

Recommendation Add an assert to ensure that the third element of the input vector passed to `convertg1` equals `1`. This element corresponds to the `Z` coordinate.

Similarly, perform a check for `convertg2` which ensures that the third input vector element equals `[1, 0]`.

4.1.5 V-RSL-VUL-005: Missing data validation on public inputs

Severity	Warning	**Commit**	84a1929
Type	Data Validation	**Status**	Fixed
File(s)	src/lib.rs		
Location(s)	new()		
Confirmed Fix At	f240072		

Inside `prepare_public_inputs()`, the `alt_bn128_multiplication` syscall takes as inputs the elements of vectors `vk_ic` and `inputs`. See snippet below for context.

The vector elements passed into `alt_bn128_multiplication` are not validated to be smaller than the field size. This can cause performance issues because `alt_bn128_multiplication` does not reduce scalars modulo the field size. If it performs scalar multiplication with a 256 bit integer, it will end up doing unnecessarily many operations.

```
fn prepare_public_inputs(&self) -> Result<[u8; 64], Risc0SolanaError> {
    let mut prepared = self.vk.vk_ic[0];
    println!("Verification Key: {:?}", 0u8);
    for (i, input) in self.public.inputs.iter().enumerate() {
    let mul_res =
        alt_bn128_multiplication(&[&self.vk.vk_ic[i + 1][..], &input[..]].concat())
             .map_err(|_| Risc0SolanaError::ArithmeticError)?;
          prepared = alt_bn128_addition(&[&mul_res[..], &prepared[..]].concat())
            .unwrap()
            .try_into()
            .map_err(|_| Risc0SolanaError::ArithmeticError)?;
    }
    Ok(prepared)
}
```

Snippet 4.6: Snippet from `prepare_public_inputs()`

Impact If inputs to the `alt_bn128_multiplication` syscall are larger than the field size, it will end up doing unnecessarily many operations.

Recommendation Verify that the inputs to `alt_bn128_multiplication` are less than the BN254 field prime.

4.1.6 V-RSL-VUL-006: Unused code

Severity	Info	**Commit**	84a1929
Type	Maintainability	**Status**	Fixed
File(s)	`src/lib.rs`		
Location(s)	See description		
Confirmed Fix At	c32ec45		

The project defines these error codes as part of `Risc0SolanaError`but does not use them:

- `InvalidPublicInput`
- `SerializationError`
- `DeserializationError`
- `InvalidInstructionData`

Impact Unused code can affect the maintainability of the project.

Recommendation Remove the error codes that are not used.

4.1.7 V-RSL-VUL-007: Point negation behaviour not clearly documented

Severity	Info	**Commit**	84a1929
Type	Usability Issue	**Status**	Fixed
File(s)	`src/lib.rs`		
Location(s)	negate_g1()		
Confirmed Fix At	08ed33c		

A Groth16 proof usually consists of three group elements, `pi_a`, `pi_b`, `pi_c`. In this library, the point `pi_a` is taken and negated through the method `negate_g1`. Let us look at the context in which the pairing is performed with a negated `pi_a`.

A general pairing equation is as follows-:

$$e(A, B) = e(\alpha \cdot g, \beta \cdot h) \cdot e(R, \gamma \cdot h) \cdot e(C, \delta \cdot h)$$

Alternatively, moving everything to the same side-:

$$e(A, B)^{(-1)} \cdot e(\alpha * g, \beta * h) \cdot e(R, \gamma \cdot h) \cdot e(C, \delta \cdot h) = 1$$

By the bilinearity of the pairing we have-:

$$e(A, B)^{(-1)} = e(-A, B)$$

So, we take the following pairs-:

$$-A, B; \alpha \cdot g, \beta \cdot h; R, \gamma \cdot h; C, \delta \cdot h$$

Then we take their pairings and multiply the results up. The pairing computation is implemented through the `alt_bn128_pairing` syscall.

These details about negating `pi_a` are not mentioned in the library. This is important, as a user of this library might not expect such a behaviour.

Recommendation Add in-line comments

- to the method `negate_g1()` which explain the motivation behind the negation of `pi_a` ,
- to the definition of the struct `Proof` which clarifies the discrepancy in the definition of `pi_a`.

4.1.8 V-RSL-VUL-008: Unrelated methods can be moved to the testing module

Severity	Info	**Commit**	84a1929
Type	Maintainability	**Status**	Fixed
File(s)	`src/lib.rs`		
Location(s)	public_inputs()		
Confirmed Fix At	17a40f1		

The following methods are related to tests, and hence should be moved out into the testing or client-side part.

- `public_inputs()`
- `digest_from_hex()`
- `split_digest_bytes()`
- `to_fixed_array()`
- `decompress_g1()`
- `decompress_g2()`

Developer Response The `public_inputs` is kept as it is necessary for on chain programs. The decompress helpers have been removed.

Bibliography

[1] J Groth. *On the Size of Pairing-Based Non-interactive Arguments*. Cryptology ePrint Archive, Paper 2016/260. https://eprint.iacr.org/2016/260. 2016 (cited on page 1).