Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

RISC ZERO

Keccak precompile

Veridise Inc.
March 24, 2025

► Prepared For:

RISC Zero
`https://risczero.com/`

► Prepared By:

Ian Neal
Kostas Ferles

► Contact Us:

`contact@veridise.com`

► Version History:

March 24, 2025	V2
February 21, 2025	V1

Contents

Contents **iii**

1 Executive Summary **1**

2 Project Dashboard **3**

3 Security Assessment Goals and Scope **4**

3.1 Security Assessment Goals 4

3.2 Security Assessment Methodology & Scope 4

3.3 Classification of Vulnerabilities 5

4 Vulnerability Report **6**

4.1 Detailed Description of Issues 7

4.1.1 V-RISC0-VUL-001: isFirst is not constrained to be a bit 7

4.1.2 V-RISC0-VUL-002: Incomplete documentation for keccak2 circuit 8

4.1.3 V-RISC0-VUL-003: Unnecessary constraints in component IsZero 9

4.1.4 V-RISC0-VUL-004: Unnecessary big endian conversion 10

Glossary **11**

1 Executive Summary

From Jan. 29, 2025 to Feb. 17, 2025, RISC Zero engaged Veridise to conduct a security assessment of their Keccak precompile. The security assessment covered the implementation of RISC Zero's Keccak Accelerator zero-knowledge circuits and the accompanying prover-side code. Veridise conducted the assessment over 4 person-weeks, with 2 security analysts reviewing the project over 2 weeks on commits `65645b0`, `b74b267`. The review strategy involved a thorough source code review performed by Veridise security analysts.

Project Summary. The security assessment covered the keccak builtin for the RISC Zero ZKVM. The keccak builtin is written in Zirgen circuits, which implement the keccak hashing operations used by the ZKVM. This project also includes the Foreign Function Interface (FFI) to bind generated code, the inclusion into external calls, and the batching mechanism for batching keccak operations to the circuits.

Code Assessment. The RISC Zero developers provided the source code of the RISC Zero for the code review. The source code appears to be mostly original code written by the RISC Zero developers. It contains some limited documentation in the form of READMEs and documentation comments on functions and circuit components. To facilitate the Veridise security analysts' understanding of the code, the RISC Zero developers also answered questions about parts of the infrastructure that were not properly documented.

The source code contained a test suite, which the Veridise security analysts noted that covered most of the common utilities in the system, but notably did not test larger core components of the system, such as the keccak permutation functions. Some tests for these core components were either commented out or only produced command-line output for visual inspection rather than assertions of correctness on the outputs.

Summary of Issues Detected. The security assessment uncovered 4 issues, 0 of which are assessed to be of high or critical severity by the Veridise analysts. Specifically, the Veridise analysts identified 2 low-severity issues, including a case where a mux selector was not constrained to be boolean (V-RISC0-VUL-001), as well as 2 informational findings.

The Keccak precompile developers have been notified about these issues and are currently reviewing them.

Recommendations. After conducting the assessment of the protocol, the security analysts had a few suggestions to improve the Keccak precompile.

Improve documentation. Despite the fact that the code base is well organized and relatively easy to reason about its safety and correctness, its documentation could be improved significantly. Currently, there are several assumptions on several modules of the code base that are not documented (see V-RISC0-VUL-002), including the fact that the keccak permutation functions produce transposed output relative to what is described in the official NIST specification. Documenting such assumptions will ensure that future versions of the code are less prone to errors and will be easier to undergo further security reviews.

Furthermore, it is also recommended to improve the documentation of Zirgen. The correctness of some modules appears to rely on some non-obvious features of the language, such as the usage of undocumented builtin components and how loops implement map operations (as loop semantics are also currently undocumented). Therefore, providing a more elaborate and complete manual of the language can also increase the project's resilience to future bugs.

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
Keccak precompile	`65645b0`, `b74b267`	Zirgen, Rust	RISC Zero ZKVM

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Jan. 29–Feb. 17, 2025	Manual	2	4 person-weeks

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	2	2	1
Warning-Severity Issues	0	0	0
Informational-Severity Issues	2	0	0
TOTAL	4	2	1

Table 2.4: Category Breakdown.

Name	Number
Under-constrained Circuit	1
Maintainability	1
Circuit Optimization	1
Optimization	1

3 Security Assessment Goals and Scope

3.1 Security Assessment Goals

The engagement was scoped to provide a security assessment of the Keccak precompile's code. During the assessment, the security analysts aimed to answer questions such as:

- Does the keccak circuit conform to the official NIST specification?
- Is the keccak circuit deterministic (i.e., properly constrained)?
- Can the circuit reject valid hash digests (i.e., over-constrained)?
- Are there any unsafe memory components in the prover?
- Does the prover follow the best practices for integrating the Rust Foreign Function Interface (FFI)?

3.2 Security Assessment Methodology & Scope

Security Assessment Methodology. To address the questions above, the Veridise analysts conducted a manual review of the prover code (written in Rust) and the zero-knowledge circuits (written in Zirgen).

Scope. The scope of this security assessment is limited to the following files from the zirgen and risc0 repositories, which contain the zero-knowledge circuits and the prover-side code, respectively. Additionally, in the risc0 repository, the scope was further restricted to only the parts of the listed files that were relevant to integrating the keccak accelerator into the ZKVM.

In-scope files from the zirgen repository:

- `zirgen/circuit/keccak2/arr.zir`
- `zirgen/circuit/keccak2/bits.zir`
- `zirgen/circuit/keccak2/is_zero.zir`
- `zirgen/circuit/keccak2/keccak.zir`
- `zirgen/circuit/keccak2/one_hot.zir`
- `zirgen/circuit/keccak2/pack.zir`
- `zirgen/circuit/keccak2/predicates.cpp`
- `zirgen/circuit/keccak2/sha2.zir`
- `zirgen/circuit/keccak2/top.zir`
- `zirgen/circuit/keccak2/xor5.zir`

In-scope files from the risc0 repository:

- `/risc0/circuit/keccak/src/lib.rs`
- `/risc0/circuit/keccak-sys/src/lib.rs`
- `/risc0/zkvm/platform/src/syscall.rs`
- `/risc0/zkvm/src/guest/env/batcher.rs`
- `/risc0/zkvm/src/guest/env/mod.rs`

Methodology. Veridise security analysts reviewed the reports of previous audits for Keccak precompile, inspected the provided tests, and read the Keccak precompile documentation. They then began a review of the code.

During the security assessment, the Veridise security analysts regularly met with the Keccak precompile developers to ask questions about the code.

3.3 Classification of Vulnerabilities

When Veridise security analysts discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise.

The severity of a vulnerability is evaluated according to the Table 3.1.

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

The likelihood of a vulnerability is evaluated according to the Table 3.2.

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

The impact of a vulnerability is evaluated according to the Table 3.3:

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

This section presents the vulnerabilities found during the security assessment. For each issue found, the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.) is specified. Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-RISC0-VUL-001	isFirst is not constrained to be a bit	Low	Fixed
V-RISC0-VUL-002	Incomplete documentation for keccak2 circuit	Low	Acknowledged
V-RISC0-VUL-003	Unnecessary constraints in component IsZero	Info	Open
V-RISC0-VUL-004	Unnecessary big endian conversion	Info	Open

4.1 Detailed Description of Issues

4.1.1 V-RISC0-VUL-001: isFirst is not constrained to be a bit

Severity	Low	**Commit**	65645b0
Type	Under-constrained Circuit	**Status**	Fixed
File(s)	zirgen/circuit/keccak2/top.zir		
Location(s)	Component Top		
Confirmed Fix At	https://github.com/risc0/zirgen/pull/204		

In circuit Top the value of isFirst is a result of a call to an untrusted extern function IsFirstCycle (see snippet below).

```
isFirst := NondetReg(IsFirstCycle());
cycleMux : WrapOneHot;
controlState : ControlState;
controlState := if (isFirst) {
  controlState@1.cycleType = CycleTypeShutdown();
  ControlState(CycleTypeInit(), 0, 0, 0)
} else {
  ComputeCurrentStep(cycleMux@1, controlState@1)
};
```

Snippet 4.1: Snippet from Top()

Since isFirst is used as a condition of an if statement, it is implicitly assumed to be a boolean.

Impact The trace is properly constrained even if isFirst is not constrained to be a boolean. However, it is unclear how if statements behave in case their conditional is not a bit.

Recommendation Add a constraint that forces isFirst to be a bit, i.e., AssertBit(isFirst).

Developer Response The developers have fixed this issue in commit 19b33a0.

4.1.2 V-RISC0-VUL-002: Incomplete documentation for keccak2 circuit

Severity	Low	**Commit**	65645b0
Type	Maintainability	**Status**	Acknowledged
File(s)	`zirgen/circuit/keccak2`		
Location(s)			
Confirmed Fix At	N/A		

The implementation of the keccak2 circuit is lacking some documentation. Even though the implementation is well designed and easy to reason about, it adds significant overhead for analysts and developers that attempt to understand the codebase.

Impact This can lead to maintainability issues in the future as more developers contribute to the codebase. It can also increase the possibility of security analysts missing bugs because there are a lot of implicit assumptions in the codebase.

Recommendation Improve the documentation in the codebase. It is recommended to add both a high-level design doc and inline technical docs.

Developer Response The developers have been notified of the issue and they have already planned to document the remaining parts of the code base.

4.1.3 V-RISC0-VUL-003: Unnecessary constraints in component IsZero

Severity	Info	**Commit**	65645b0
Type	Circuit Optimization	**Status**	Open
File(s)	`zirgen/circuit/keccak2/is_zero.zir`		
Location(s)	component IsZero		
Confirmed Fix At	N/A		

The following two constraints in component `IsZero` are not necessary.

1. `AssertBit(isZero)`: this constraint is implied by `val * inv = 1 - isZero` and `isZero * val = 0`
2. `isZero * inv = 0`: this constraint is only needed if `inv` is required to be deterministic. Otherwise, it does not add affect the correctness of the component.

Recommendation Remove the above constraints.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

4.1.4 V-RISC0-VUL-004: Unnecessary big endian conversion

Severity	Info	**Commit**	b74b267
Type	Optimization	**Status**	Open
File(s)	`risc0/circuit/keccak/src/lib.rs`		
Location(s)	Function compute_keccak_digest		
Confirmed Fix At	N/A		

Function `compute_keccak_digest` always converts the result of `sha::Impl::compress` to big endian format (see snippet below).

```rust
let mut digest = SHA256_INIT;
for halfs in bytemuck::cast_slice::<[u64; 32], [u64; 8]>(transcript.as_slice()) {
    let mut first_half = [0u8; DIGEST_BYTES];
    first_half.clone_from_slice(bytemuck::cast_slice(&halfs[0..4]));

    let mut second_half = [0u8; DIGEST_BYTES];
    second_half.clone_from_slice(bytemuck::cast_slice(&halfs[4..8]));

    digest = *sha::Impl::compress(
        &digest,
        &Digest::from_bytes(first_half),
        &Digest::from_bytes(second_half),
    );
}

// reorder to match the keccak accelerator
for word in digest.as_mut_words() {
    *word = word.to_be();
}
```

Snippet 4.2: Snippet from `compute_keccak_digest()`

However, this is not needed because `sha::Impl::compress` returns the digest in big endian format already.

Recommendation If the implementation of `sha::Impl::compress` is stable, remove the conversion to big endian in `compute_keccak_digest`.

Developer Response The developers have been notified of the issue but have yet to respond with acknowledgement or fixes.

Glossary

zero-knowledge circuit A cryptographic construct that allows a prover to demonstrate to a verifier that a certain statement is true, without revealing any specific information about the statement itself. See https://en.wikipedia.org/wiki/Zero-knowledge_proof for more. 1