Axiom Halo2 Library Upgrades

Security Assessment

October 2, 2023

Prepared for:
Yi Sun
Jonathan Wang
Axiom

Prepared by: **Tjaden Hess, Will Song, and Joop van de Pol**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Brooke Langhorne, Project Manager
brooke.langhorne@trailofbits.com

The following engineering director was associated with this project:

Jim Miller, Engineering Director, Cryptography
james.miller@trailofbits.com

The following consultants were associated with this project:

Tjaden Hess, Consultant
tjaden.hess@trailofbits.com

Will Song, Consultant
will.song@trailofbits.com

Joop van de Pol, Consultant
joop.vandepol@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
September 8, 2023	Pre-project kickoff call
September 15, 2023	Status update meeting #1
September 22, 2023	Status update meeting #2
October 2, 2023	Delivery of report draft
October 2, 2023	Report readout meeting
October 23, 2023	Delivery of comprehensive report
November 27, 2023	Delivery of comprehensive report with fix review appendix

Executive Summary

Engagement Overview

Axiom engaged Trail of Bits to review the security of changes to Axiom's `halo2-lib` and `snark-verifier` repositories. This review was scoped to cover specific changes made to the two libraries with respect to the versions audited by Trail of Bits on June 16, 2023.

A team of three consultants conducted the review from September 11 to September 29, 2023 for a total of six engineer-weeks of effort. Our testing efforts focused on verifying the soundness of generated constraints in the `halo2-lib` gadgets and in the `snark-verifier` aggregation and recursive verification code. With full access to the source code and documentation, we performed static and dynamic testing of the `halo2-lib` and `snark-verifier` codebases, using automated and manual processes. We did not perform in-depth reviews of code that existed prior to the specific commits listed in the Project Coverage section.

Observations and Impact

We did not observe any issues in the new changes that impact the security of the codebase. However, we discovered a soundness issue (TOB-AXIOMv2-3) that was present in the codebase before the addition of the changes under audit. In many cases, the documentation has not been updated to reflect the new design and functionality that the reviewed changes introduce (TOB-AXIOMv2-4).

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that Axiom take the following steps prior to finalizing a new library release:

- **Remediate the findings disclosed in this report.** The soundness issue described in finding TOB-AXIOMv2-3 is of particularly high severity, and downstream users of the library should be informed that the new release contains security-relevant fixes. These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.

- **Review generated documentation for correctness and completeness.** Before release, the documentation should be built with `cargo doc`, and new features should be reviewed. Additionally, perform a search on the codebase for any documentation or comment references to symbols removed by recent changes.

- **Review dependencies for security issues.** Run `cargo audit` and remediate any findings before issuing a new release.

- **Integrate documentation builds and dependency audits into the CI pipeline.** Use `cargo audit` and `cargo doc` in the CI pipeline to ensure that the documentation and dependencies are always up-to-date.

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	1
Medium	0
Low	0
Informational	5
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Data Validation	1
Documentation	1
Patching	1
Testing	2
Undefined Behavior	1

Project Goals

The engagement was scoped to provide a security assessment of specific changes to the Axiom `halo2-lib` and `snark-verifier` codebases. Specifically, we sought to answer the following non-exhaustive list of questions:

- Do changes to incorporate updates to the `halo2curves` and `ff` dependency traits preserve the semantics of the proof system?
- Does the universal aggregation circuit constrain all necessary aspects of the recursive verification?
- Does the universal aggregation functionality have sufficient documentation and assertions to prevent likely accidental misuse?
- Does the addition of universal aggregation preserve the soundness of existing applications using non-universal aggregation?
- Is the new variable-length Poseidon functionality in `halo2-lib` sound and resistant to collisions between different-length inputs?
- Does the new variable-length Keccak circuit soundly compute the padding and commitments for variable-length inputs?
- Does the Solidity-based EVM loader preserve the functionality of the previous Yul-based implementation?
- Do any of the changes cause previously sound applications to become unsound?

Project Targets

The engagement involved a review and testing of the following targets.

halo2-lib

Repository	https://github.com/axiom-crypto/halo2-lib
Version	`release-0.4.0-rc`
	`e36c45b09ac06390cac51a8d6dca74f7835ba80d`
Types	Rust, halo2
Platform	Native

snark-verifier

Repository	https://github.com/axiom-crypto/snark-verifier
Version	`release-0.1.6-rc0`
	`d8400e60ae5762b9d3234f2fda68a556c463fa1a`
Types	Rust, halo2
Platform	Native

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- Manual review of the `snark-verifier` recursion and aggregation universality updates
- Manual review of the Keccak “vanilla” circuit, especially with respect to variable-length inputs
- Manual review of the Solidity wrapper for the EVM verifier
- Manual review of the `pow_var`, `sub_mul`, and `select_array_by_indicator` gadgets
- Manual review of the `FpChip::range_check`, `ec_sub_unequal`, `left_pad`, and `inner_product_left` gadgets
- Manual review of the `SafeType` and `VariableByteArray` features
- Manual review of the `VirtualRegionManager` and dynamic lookup features
- Manual review of the integration of and changes to Poseidon in `halo2-lib`, paying particular attention to variable-length inputs, as this functionality is crucial to Keccak
- Manual review of updates made for the upgraded dependencies `halo2curves` and `ff`

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- This review did not cover any components except those listed in the pull requests associated with the `snark-verifier/release-0.1.6-rc0` and `halo2-lib/release-0.4.0-rc` releases. In particular, we did not perform an end-to-end review of `halo2-lib` or `snark-verifier`.
- This review did not cover the `halo2curves` or `ff` dependencies themselves, only changes to their usage in the `halo2-lib` and `snark-verifier` libraries.

Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in house, to perform automated testing of source code and compiled software.

Test Harness Configuration

We used the following tools in the automated testing phase of this project:

Tool	Description	Policy
Semgrep	An open-source static analysis tool for finding bugs and enforcing code standards when editing or committing code and during build time	`semgrep --config=auto`
Clippy	A collection of lints to catch common mistakes and improve Rust code	`cargo clippy --workspace -- -W clippy::pedantic`
`cargo audit`	An open-source tool for checking dependencies against the RustSec advisory database	`cargo audit`

Areas of Focus

Our automated testing and verification work focused on the following system properties:

- Known vulnerability patterns are not present.
- Rust best practices are followed.
- Dependencies do not have active security advisories.

Test Results

The results of this focused testing are detailed below.

`halo2-lib`

Property	Tool	Result
Known vulnerability patterns are not present.	Semgrep	**Passed**

Rust best practices are followed.	Clippy	**Passed**
Dependencies do not have active security advisories.	`cargo audit`	TOB-AXIOMv2-2

`snark-verifier`

Property	**Tool**	**Result**
Known vulnerability patterns are not present.	Semgrep	**Passed**
Rust best practices are followed.	Clippy	**Passed**
Dependencies do not have active security advisories.	`cargo audit`	TOB-AXIOMv2-2

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	We did not find any incidence of incorrect calculation or unexpected overflow. However, some circuits rely implicitly on the underlying field size to be large enough that overflow by repeated incrementing is impossible. This may not hold in smaller fields that could be introduced in the future.	**Satisfactory**
Auditing	Panics and errors typically have human-readable explanations. No logging is necessary at the library level.	**Satisfactory**
Authentication / Access Controls	There are no permissioned roles in the libraries.	**Not Applicable**
Complexity Management	The codebase is generally well separated into traits that manage independent concerns. However, complexity management is weakened by missing or outdated documentation (TOB-AXIOMv2-4) and inconsistent testing (TOB-AXIOMv2-5, TOB-AXIOMv2-6).	**Moderate**
Cryptography and Key Management	We found one underconstrained circuit (TOB-AXIOMv2-3). We did not find any severe issues with the new hash implementations or PLONK verification.	**Moderate**
Documentation	The documentation is out of date in various places, as it has not been updated in line with changes made to the codebase. This holds both for the user-facing documentation of the APIs (TOB-AXIOMv2-4) and documentation of the code itself through comments (appendix C). As a result, users may fail to satisfy the assumptions on the input made by the implementation. The sparse documentation also makes it more difficult to audit and maintain the code.	**Weak**

Memory Safety and Error Handling	The codebase does not rely on unsafe Rust.	**Strong**
Testing and Verification	Most functionalities in `snark-verifier` and `halo2-lib` have example end-to-end tests; however, most components do not have any edge-case or error-case testing. Standardized functionality such as Keccak are missing known-answer tests to verify the correctness of the implementation (TOB-AXIOMv2-5).	**Moderate**

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Solidity loader does not respect free memory pointer	Undefined Behavior	**Informational**
2	Libraries have unmaintained dependencies	Patching	**Informational**
3	FpChip::range_check is unsound on CRTInteger values greater than 2^n*k	Data Validation	**High**
4	Missing or inconsistent documentation	Documentation	**Informational**
5	Keccak lacks tests against known test vectors	Testing	**Informational**
6	Feature-gated test no longer compiles	Testing	**Informational**

Detailed Findings

1. Solidity loader does not respect free memory pointer	
Severity: **Informational**	Difficulty: **N/A**
Type: Undefined Behavior	Finding ID: TOB-AXIOMv2-1
Target: `snark-verifier/src/loader/evm/loader.rs`	

Description

The EVM verifier produced by `snark-verifier` does not respect the memory layout of Solidity; in particular, the "free memory pointer" at memory location `0x40` is ignored and clobbered by the verifier logic.

```
mstore(0x40, mod(calldataload(0x20), f_q))
mstore(0x60, mod(calldataload(0x40), f_q))
```

Figure 1.1: The Solidity PLONK verifier overwrites memory at `0x40`. (*`snark-verifier/snark-verifier-sdk/examples/StandardPlonkVerifier.sol#29–30`*)

Because the surrounding Solidity wrapper does not perform any nontrivial memory operations, and because the assembly block is not marked as `memory-safe`, the current wrapper does not exhibit any unexpected behavior. However, future updates or use cases that include the assembly block in a more complex Solidity contract may accidentally clobber the in-use memory of the wrapping contract.

Recommendations

Short term, change `EvmLoader::new` to begin allocation at location `0x80`, and change the verifier ASM prelude to assert that `mload(0x40)` is equal to `0x80`. This will prevent the generated code from being used in contexts in which the static assumptions on memory layout do not hold.

Long term, consider modifying the loader allocation routine to do dynamic allocation and return a free memory pointer variable rather than an integer.

2. Libraries have unmaintained dependencies	
Severity: **Informational**	Difficulty: **N/A**
Type: Patching	Finding ID: TOB-AXIOMv2-2
Target: `halo2-lib/hashes/zkevm/Cargo.toml`	

Description

The `zkevm-hashes` and `snark-verifier-sdk` packages use dependencies that have been marked as unmaintained by the dependency developers; specifically, `cargo audit` reports the use of the unmaintained `tui` and `criterion` crates. These crates may not receive bug fixes for security vulnerabilities discovered in the future. While both unmaintained crates are used only for testing and benchmarking, routinely auditing dependencies is an important step in the software development cycle.

```
> cargo audit
    Fetching advisory database from `https://github.com/RustSec/advisory-db.git`
      Loaded 570 security advisories (from /Users/tjaden/.cargo/advisory-db)
    Updating crates.io index
    Scanning Cargo.lock for vulnerabilities (499 crate dependencies)
Crate:     tui
Version:   0.19.0
Warning:   unmaintained
Title:     `tui` is unmaintained; use `ratatui` instead
Date:      2023-08-07
ID:        RUSTSEC-2023-0049
URL:       https://rustsec.org/advisories/RUSTSEC-2023-0049
Dependency tree:
tui 0.19.0
├── snark-verifier-sdk 0.1.6
└── snark-verifier 0.1.6
    └── snark-verifier-sdk 0.1.6
```

Figure 2.1: The `cargo audit` report for `snark-verifier`

```
> cargo audit
    Fetching advisory database from `https://github.com/RustSec/advisory-db.git`
      Loaded 570 security advisories (from /Users/tjaden/.cargo/advisory-db)
    Updating crates.io index
    Scanning Cargo.lock for vulnerabilities (358 crate dependencies)
Crate:     serde_cbor
Version:   0.11.2
Warning:   unmaintained
Title:     serde_cbor is unmaintained
```

```
Date:      2021-08-15
ID:        RUSTSEC-2021-0127
URL:       https://rustsec.org/advisories/RUSTSEC-2021-0127
Dependency tree:
serde_cbor 0.11.2
└── criterion 0.3.6
    └── zkevm-hashes 0.1.4
```

Figure 2.2: The `cargo audit` report for `halo2-lib`

Recommendations

Short term, upgrade `criterion` to at least version 0.4.0 and replace `tui` with `ratatui`.

Long term, integrate `cargo audit` into the CI and release workflow.

3. FpChip::range_check is unsound on CRTInteger values greater than 2^n*k

Severity: **High**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-AXIOMv2-3
Target: `halo2-ecc/src/fields/fp.rs`	

Description

The `FpChip` implementation of the `FieldChip::range_check` trait method checks only the `truncation` limbs of the input `CRTInteger`, ignoring the `native` field. If the `CRTInteger` value is greater than what can be represented in the truncated component, the `range_check` constraint system will be satisfiable as long as the truncated component is sufficiently small.

The `FieldChip::range_check` is not used by the broader `halo2-ecc` library in a way that leads to vulnerabilities in higher-level functions, but other users of the library may instantiate the gadget in ways that lead to exploitable vulnerabilities.

An excerpt of the `range_check` function is shown in figure 3.1. A `debug_assert` prevents honest provers from assigning overly large values to the argument a but does not constrain a malicious prover's assignment.

```rust
fn range_check(
    &self,
    ctx: &mut Context<F>,
    a: impl Into<CRTInteger<F>>,
    max_bits: usize, // the maximum bits that a.value could take
) {
    let n = self.limb_bits;
    let a = a.into();
    let mut remaining_bits = max_bits;

    debug_assert!(a.value.bits() as usize <= max_bits);

    // range check limbs of `a` are in [0, 2^n) except last limb should be in [0, 2^last_limb_bits)
    for cell in a.truncation.limbs {
        let limb_bits = cmp::min(n, remaining_bits);
        remaining_bits -= limb_bits;
        self.range.range_check(ctx, cell, limb_bits);
    }
}
```

Figure 3.1: `range_check` does not constrain `a.native`.
(`halo2-lib/halo2-ecc/src/fields/fp.rs#304–322`)

The only use of `range_check` inside of `halo2-lib` is in the `load_private` method, where it is called on a `ProperCRTUint`; thus, the argument is guaranteed to have a value less than 2^{n*k}. However, the `Fp` circuit and `FieldChip` trait are public exports of the `halo2-ecc` library and thus may be used by other developers or projects. These users would naturally expect, in the absence of any documentation to the contrary, that the `range_check` function is sound for all well-formed `CRTInteger` inputs.

Exploit Scenario

A third-party developer uses the Axiom `halo2-ecc` library and wants to write a gadget that extends `enforce_less_than` to work on an `UnsafeFieldPoint`. The developer relies on `range_check` to ensure that the difference between the modulus and an adversarially chosen witness does not overflow.

A malicious prover uses the fact that `range_check` does not soundly enforce the check for all inputs and constructs a `CRTInteger` with a value greater than 2^{n*k}, such that the value is slightly more than a multiple of 2^{n*k}. This `CRTInteger` has a small `truncation` component and thus passes the range check, leading to a potential forgery in the higher-level circuit.

Recommendations

Short term, change the `range_check` trait method to accept a `FieldPoint` rather than an `UnsafeFieldPoint`.

Long term, add unit tests for all `FpChip` methods. Ensure that all methods that accept `UnsafeFieldPoints` are tested with non-proper inputs.

4. Missing or inconsistent documentation	
Severity: **Informational**	Difficulty: **N/A**
Type: Documentation	Finding ID: TOB-AXIOMv2-4
Target: `halo2-lib`, `snark-verifier`	

Description
Some public library functions lack documentation or are missing documentation regarding assumptions on inputs.

- `VarLenByes::left_pad_to_fixed` panics at runtime if the input bytes array's `max_len` variable is equal to zero. More generally, the documentation for `left_pad_var_array_to_fixed` should be duplicated or linked from `VarLenByes::left_pad_to_fixed`.

- `FpChip::range_check` has two important assumptions on the input: the `max_bits` variable cannot be larger than the chip itself, and the number to be checked needs to have the same number of limbs as the chip. These assumptions are not listed for the `FieldVectorChip::range_check` function, which calls `FpChip::range_check` for all elements in the vector. More generally, the documentation for the individual `FpChip` functions should be duplicated or linked from the corresponding `FieldVectorChip` functions.

- `SafeTypeChip::raw_to_var_len_bytes` and `raw_to_var_len_bytes_vec` take a maximum length of type `usize`, which is incremented by one in a call to `RangeChip::check_less_than_safe`. On 64-bit systems, if the user uses `u64::MAX` as the maximum length, this will overflow, resulting in a maximum length of zero.

- Furthermore, `check_less_than_safe` calls `RangeChip::range_check`, which assumes that the result of `ceil(range_bits / lookup_bits) * lookup_bits` is less than or equal to the value of `F::CAPACITY`. This is guaranteed for large fields, as the maximum length is restricted to have at most 64 bits in the current implementation, but may not generically be true for all fields. More generally, any assumptions that propagate to calling functions should be documented for those calling functions (in this case, both `check_less_than_safe` and the aforementioned `SafeTypeChip` functions).

- `RangeChip::div_mod` computes a division with remainder, under the assumptions that the divisor is nonzero and that the number of bits of the number to be divided is less than or equal to the input parameter `a_num_bits`. However, an undocumented assumption is that this parameter, `a_num_bits`, must be strictly less than the `NUM_BITS` parameter of the corresponding field. If this assumption is not met, a malicious prover can find incorrect quotients and remainders that meet the constraints.

- The `README` file in `halo2-base` has not been updated to reflect the new `VirtualRegionManager` model. The example code snippets will no longer compile, and the outdated documentation may cause confusion for users.

- There are several function-level comments that describe a `GateStrategy` input. This type no longer exists, and the comments may cause confusion.

Recommendations

Short term, add or update the documentation for the listed functions.

Long term, ensure that, as a step of the release process, the libraries have documentation that is buildable with `cargo doc` and that public functions have sufficient documentation such that a caller can use them safely and without unexpected panics.

5. Keccak lacks tests against known test vectors

Severity: **Informational**	Difficulty: **N/A**
Type: Testing	Finding ID: TOB-AXIOMv2-5
Target: `halo2-lib`, `keccak`	

Description

The Keccak tests form only a simple test suite over the Keccak circuit. In particular, `packed_multi_keccak_simple` tests that the output witnesses are correct for a few static inputs, while `packed_multi_keccak_prover` tests the correctness of the entire prover from setup to proof generation to verification of the same inputs without output witness verification. It may be beneficial to incorporate some test vectors from the Keccak Known Answer Tests to better test a variety of cases. While the Keccak team has marked them as obsolete, these are still the latest test vectors published for Keccak and are worth incorporating to establish a more comprehensive test suite.

Recommendations

Short term, include a selection of the known test vectors in the Keccak test suite.

Long term, determine which Keccak test vector files are worth parsing and automating tests against all known tests within the file.

6. Feature-gated test no longer compiles

Severity: **Informational**	Difficulty: **N/A**
Type: Testing	Finding ID: TOB-AXIOMv2-6
Target: `halo2-lib/halo2-ecc/src/fields/tests/fp/mod.rs`	

Description

The `plot_fp` test produces a PLONK grid visualization for the `FpChip` constraint system. The test uses `GateThreadBuilder`, which no longer exists.

```
#[cfg(feature = "dev-graph")]
#[test]
fn plot_fp() {
    use plotters::prelude::*;

    let root = BitMapBackend::new("layout.png", (1024, 1024)).into_drawing_area();
    root.fill(&WHITE).unwrap();
    let root = root.titled("Fp Layout", ("sans-serif", 60)).unwrap();

    let k = K;
    let a = Fq::zero();
    let b = Fq::zero();

    let mut builder = GateThreadBuilder::keygen();
    fp_mul_test(builder.main(0), k - 1, 88, 3, a, b);

    let config_params = builder.config(k, Some(10), Some(k - 1));
    let circuit = RangeCircuitBuilder::keygen(builder, config_params);
    halo2_proofs::dev::CircuitLayout::default().render(k as u32, &circuit,
&root).unwrap();
}
```

Figure 6.1: The `plot_fp` test
(`halo2-lib/halo2-ecc/src/fields/tests/fp/mod.rs#62–81`)

Attempting to run the test with `cargo test --features “dev-graph”` therefore results in a compilation error.

Recommendations

Short term, update the `plonk_fp` test to the new `VirtualRegionManager` model.

Long term, run tests using the `--all-features` flag before issuing new releases.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	Description
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels

Severity	Description
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels

Difficulty	Description
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	Description
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Documentation	The presence of comprehensive and readable codebase documentation
Memory Safety and Error Handling	The presence of memory safety and robust error-handling mechanisms
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Code Quality Findings

We identified the following code quality issues through manual and automatic code review.

- Due to errors in cross-package references, `cargo doc` fails to generate documentation for `snark-verifier` and produces several warnings for `halo2-lib`.

- The helper function `add_bytes_constraints` in the `SafeTypeChip` implementation adds constraints enforcing that each entry in a vector of values is a byte and that all entries together have a bit length of at most `bits`. However, this function assumes that the input variable `bits` is in the range $((n - 1) * 8, n * 8]$, which is not documented. Currently, this is not an issue, as the function is used only internally in a single place (and there is an assertion in the calling function to ensure the assumption), but it may be problematic if this function is used in more places in the future.

- The vanilla Keccak implementation contains outdated comments pertaining to RLCs.

- The calls to `to_vec` and `collect` in the following location are redundant:

```
.collect::<Vec<_>>()
.to_vec()
```

Figure C.1: The redundant call to `to_vec`
(`halo2-lib/hashes/zkevm/src/keccak/vanilla/witness.rs#411–412`)

- The following test case description has the vectors in the opposite order from the test logic:

```
#[test_case([4,5,6].map(Fr::from).to_vec(), [1,2,3].map(|x|
Constant(Fr::from(x))).to_vec() => (Fr::from(32), [4,5,6].map(Fr::from).to_vec());
"inner_product_left(): <[1,2,3],[4,5,6]> Constant b starts with 1")]
```

Figure C.2: The inner product test case is in reverse order.
(`halo2-lib/halo2-base/src/gates/tests/flex_gate.rs#102–103`)

D. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

From November 13 to November 15, 2023, Trail of Bits reviewed the fixes and mitigations implemented by the Axiom team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the six issues described in this report, Axiom has resolved five and has partially resolved one. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Solidity loader does not respect free memory pointer	Resolved
2	Libraries have unmaintained dependencies	Resolved
3	FpChip::range_check is unsound on CRTInteger values greater than 2^n*k	Resolved
4	Missing or inconsistent documentation	Partially Resolved
5	Keccak lacks tests against known test vectors	Resolved
6	Feature-gated test no longer compiles	Resolved

Detailed Fix Review Results

TOB-AXIOMv2-1: Solidity loader does not respect free memory pointer
Resolved. `EvmLoader::new` now begins allocations at `0x80`, and the verifier contract checks that `mload(0x40)` is equal to `0x80`.

TOB-AXIOMv2-2: Libraries have unmaintained dependencies
Resolved. `cargo audit` has been integrated into CI, `tui` has been replaced with `ratatui`, and `criterion` has been removed from `zkevm-hashes`.

TOB-AXIOMv2-3: FpChip::range_check is unsound on CRTInteger values greater than 2^n*k
Resolved. a is now `Into<ProperCrtUint<F>>` instead of `Into<CRTInteger<F>>`, which has the proper range-checking properties. However, unit tests were not added to check this fact.

TOB-AXIOMv2-4: Missing or inconsistent documentation
Partially resolved. The documentation has been vastly improved. However, there are still some mentions of the `GateStrategy` input, a type that no longer exists.

TOB-AXIOMv2-5: Keccak lacks tests against known test vectors
Resolved. Some known answer tests have been added to the Keccak test suite.

TOB-AXIOMv2-6: Feature-gated test no longer compiles
Resolved. The test no longer uses `GateThreadBuilder`; it uses `BaseCircuitBuilder` instead.

E. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	Description
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.