Axiom Circuits Security Review

Auditors

CPerezz, Lead Security Researcher

Eduard Sanou, Lead Security Researcher

Kyle Charbonnet, Security Researcher

Report prepared by: Lucas Goiriz

January 20, 2024

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Axiom gives smart contracts trustless access to the entire history of Ethereum and arbitrary ZK-verified compute over it. Developers can send on-chain queries into Axiom, which are trustlessly fulfilled with ZK-verified results sent in a callback to the developer's smart contract. This allows developers to build rich on-chain applications without additional trust assumptions.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Axiom according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 19 days in total, Axiom engaged with Spearbit to review the halo2-lib, axiom-eth-working and axiom-sdk-client protocols. In this period of time a total of **12** issues were found.

Summary

Project Name	Axiom
Repositories	halo2-lib, axiom-eth-working, axiom-sdk-client
Commits	267123...810f3d, aae4ae...c3f6ed, bee86e...97d473, af148c...b1f68a, 49d136...427653, PR200
Type of Project	ZK, Cryptography
Audit Timeline	Oct 23 to Nov 11
Two week fix period	Nov 11 - Nov 25

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	4	4	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	0	0	0
Gas Optimizations	1	1	0
Informational	7	5	2
Total	**12**	**10**	**2**

5 Findings

5.1 Critical Risk

5.1.1 Missing fixed column in dynamic lookup table

Severity: Critical Risk

Context: halo2-base/src/virtual_region/lookups/basic.rs#L39

Description: The dynamic lookup table functionality provided by `halo2-lib` does not include any fixed/selector columns for the table columns. So all table columns are advice columns. This can easily lead to bugs when all rows of these advice columns are not constrained. The unconstrained rows can have any input value by the attacker, allowing them to include a false lookup. This bug was originally seen in the Scroll/PSE zkevm circuits.

Note that this was already reported in the `halo2-lib` repository by one of the reviewers. The issue has been copied here to include it in the audit report.

Recommendation: Adding a fixed/selector column to the lookup table allows lookups to succeed only for rows where the selector is 1. This selector should be enabled only on rows where the lookup table's advice values are set.

Axiom: Resolved in PR 206.

Spearbit: Fixed.

5.1.2 Missing check on the Header snark keccak promise in `SubqueryAggregation`

Severity: Critical Risk

Context: axiom-query/src/subquery_aggregation/circuit.rs#L140

Description: In the `SubqueryAggregation` circuit there are copy constraints to check for all the subquery snarks that depend on other subquery, that their promises match the commitments, where each promise is `hash(promise_keccak | promise_subquery)`. But the header subquery doesn't depend on other subquery, yet it still promises some keccaks. The promise that the header snark gives is `hash(promise_keccak)`, but that promise is not checked to be correct.

A copy constraint that verifies the following check is missing: `public_inputs_header.promise_result_commit == hash(promise_keccak)`. This implies that the keccaks used in the Header circuit are not verified. Without these keccak's verifications, a malicious prover could generate a header snark proof with arbitrary block headers (which would not be required to match the real block headers hashes) and sill build a final passing proof; with this, the malicious prover could prove arbitrary (invalid) subquery results.

Recommendation: In the `SubqueryAggregation` circuit introduce a copy constraint that performs the following logic:

```
public_inputs_header.promise_result_commit == hash(promise_keccak)
```

Axiom: Fixed in PR 191.

Spearbit: Fixed.

5.1.3 Missing copy constraints in all dynamic lookup tables

Severity: Critical Risk

Context: halo2-base/src/virtual_region/lookups/basic.rs#L154

Description: In the dynamic table implementation from `halo2-lib`, the `raw_synthesize_phase1` is supposed to bind a list of cells corresponding to the lookup source `to_lookup` to the "source" halo2 lookup column, and a list of cells corresponding to the lookup table `lookup_table` to the "dest" halo2 lookup column. The first binding is handled via `BasicDynLookupConfig::assign_virtual_to_lookup_to_raw` which applies copy constraints. The second binding is handled via `BasicDynLookupConfig::assign_virtual_table_to_raw` but doesn't apply copy constraints, it only assigns values to the "dest" halo2 lookup column. This means that the values of the lookup column are disconnected from the actual cells that constrain the table data.

`assign_virtual_to_lookup_to_raw` works correctly because it uses the function `constrain_virtual_equals_external`, but `assign_virtual_table_to_raw` only uses `assign_virtual_to_raw` which doesn't create any copy constraint. This issue affects all circuits that do dynamic lookups (which is most of them, as they do lookups with the `PromiseLoader`). This issue would allow a malicious prover to generate valid proofs with invalid lookup key-values (for example keccaks or subquery results)

Note that this issue is not obvious by looking at the code. I found it by running a custom analysis tool looking for cells only used in 1 constraint. A quick way to verify this issue is to dump all the copy constraints that include the lookup "dest" halo2 column: there will be none.

Recommendation: The `halo2-lib` function `assign_virtual_to_raw` seems a bit misleading. It receives a virtual cell and assigns it's value to a halo2 cell, and then stores the mapping between the virtual and halo2 cell in the `copy_manager`. This mapping is only used in case copy constraints are defined on top of this virtual cell. If the virtual cell was used in a gate, it will be later assigned to a different halo2 cell with the constraints of the gate; and we end up with two different halo2 cells that are disconnected. I would recommend removing the function `assign_virtual_to_raw`, and perhaps consider adding a similar function that assigns a value to a halo2 cell, and returns a new virtual cell corresponding to it. Then to solve the dynamic lookup issue, I would implement `assign_virtual_table_to_raw_from_offset` using the same code as `assign_virtual_to_lookup_to_raw_from_offset`, that is, calling `raw_assign_advice` and `constrain_virtual_equals_external`.

Axiom: Fixed in PR 224.

Spearbit: Fix looks good. I've run the analysis applying the patch in PR 224 with the `ResultsRoot` circuit and now I see the copy constraints being applied to the "dest" halo2 lookup column, and I don't see any "dangling" cell.

5.1.4 Missing `raw_synthesize_phase1` for second `PromiseBuilder` in `PromiseBuilderCombo`

Severity: Critical Risk

Context: axiom-eth/src/utils/component/promise_loader/combo.rs#L88

Description: The `PromiseBuilderCombo` doesn't execute the `raw_synthesize_phase1` for the second `PromiseBuilder`. Instead it runs `raw_synthesize_phase1` twice for the first `PromiseBuilder`. The affected circuits are: `ResultsRoot`, Subquery circuits `Transaction`, `Account`, `Storage`, `Receipt`, `Solidity`. As a result of this mistake, the copy constraints of the source-dest values that should be enforced via a lookup are not enabled; this means that a malicious prover could generate proofs of invalid subqueries successfully.

Recommendation: Apply the following patch:

```diff
--- a/axiom-eth/src/utils/component/promise_loader/combo.rs
+++ b/axiom-eth/src/utils/component/promise_loader/combo.rs
@@ -85,6 +85,6 @@ impl<F: Field, FIRST: PromiseBuilder<F>, SECOND: PromiseBuilder<F>> PromiseBuild
     }
     fn raw_synthesize_phase1(&mut self, config: &Self::Config, layouter: &mut impl Layouter<F>) {
         self.to_combine.0.raw_synthesize_phase1(&config.0, layouter);
-        self.to_combine.0.raw_synthesize_phase1(&config.0, layouter);
+        self.to_combine.1.raw_synthesize_phase1(&config.1, layouter);
     }
 }
```

Axiom: Fixed in PR 190.

Spearbit: Fixed.

5.2 Gas Optimization

5.2.1 Possible gas optimization saving RLC column and challenge for `CoreIntentRoot` circuit keygen

Severity: Gas Optimization

Context: [axiom-core/src/keygen/mod.rs#L232](`https://github.com/axiom-crypto/axiom-eth-working/blob/f2c757b76a20f19f8180fc64622a2bdda49e9a65/axiom-core/src/keygen/mod.rs#L232`

Description: In the `KeygenAggregationCircuitIntent` trait impl for `CoreIntentRoot`, specifically in the `build_keygen_circuit_from_snarks`, the rlc-bits value is set to `1` but in integration tests this is set to `0`.

Is it a big deal? If not, can we simplify/clarify this?

This should be a `0` (ideally) such that we don't waste a Phase1 column and challenge computation unnecessarily. This potentially would save gas even as `RootCircuit` is verified in the EVM.

Recommendation: Review this and set to 0 if possible.

Axiom: Addressed in PR 248.

Spearbit: Fixed.

5.3 Informational

5.3.1 Simplify `CoreIntent` structures

Severity: Informational

Context: PR 200

Description: `CoreIntent`-related structures all hold `kzg_params: Arc<ParamsKZG<Bn256>>` inside them. There's a comment (which is indeed correct saying that only one G1 point of the entire struct is actually used.), see axiom-core/src/keygen/mod.rs#L89-L90.

Recommendation: Why not loading the `DummyParams`, in case you don't want to do a PR to the sdk repo such that your Params in memory are much smaller and at the same time, you place into them or wherever you want the G1 point that you indeed need?

Axiom: Acknowledged. We will keep as is. The `CoreIntent` structures mentioned are only used internally, and passing the `Arc<Params>` around is not a memory concern. We prefer for proving key generation to avoid needing to do any tricks around making a dummy kzg params.

Longer term, we will change the `snark-verifier` API so it does not require `kzg_params`.

Spearbit: Acknowledged.

5.3.2 Leftovers and unused code

Severity: Informational

Context: PR 200

Description: There are a couple places where there is a chance to remove lines of code. Particularly in:

- axiom-query/src/components/results/results_root.rs#L85: there is a possible leftover.
- axiom-query/src/subquery_aggregation/types.rs#L115-L117: we have claimed unused things which we can get rid off.

Recommendation: Analyze if we can indeed get rid of these things. And if so, do it.

Axiom: Acknowledged. See the details below:

- axiom-query/src/components/results/results_root.rs#L85: there is a possible leftover.

This was just a comment to explain the next line of code.

- axiom-query/src/subquery_aggregation/types.rs#L115-L117: we have claimed unused things which we can get rid off.

It is unused, but implements a trait that could be useful later, so we will keep the line.

Spearbit: Acknowledged.

5.3.3 `OnchainVerifyingKey` should have an `impl` block

Severity: Informational

Context: axiom-query/src/utils/client_circuit/vkey.rs#L33-L76

Description: The two highlighted methods `read_onchain_vkey` and `transform_onchain_vkey_to_plonk_protocol` could be inside of an `impl` block instead of having standalone functions. This reduces imports, verbosity and will clarify and simplify the code structure around this struct.

Recommendation: Add both functions within an `impl` block.

Axiom: Fixed in PR 259.

Spearbit: Acknowledged.

5.3.4 Unused user results and subqueries are unconstrained in `axiom-sdk-client`

Severity: Informational

Context: rust/src/scaffold.rs#L210, rust/src/scaffold.rs#L215, rust/src/subquery/caller.rs#L102

Description: Unused outputs of the userCompute circuit generated via the `axiom-sdk-client`, both for user results and subqueries, are assigned 0 values but they are free witnesses. The parameter `result_len` will determine the number of user results that are used; and from the `data_query` we can infer the number of subqueries that are used. Also, the subquery data in the public input has a fixed max size, which some subquery types don't use fully, leading to unused elements which are also assigned 0 but unconstrained (but they will be skipped when parsing the subquery).

This means that the values of unused results and subqueries don't matter in practice, but the client can set any value there, which could lead to some confusion if these values are not set to 0.

Recommendation: Fixing the unused user results and subqueries, as well as the padding elements in the subqueries in the public inputs to be 0 with `load_constant(F::ZERO)` instead of letting them be free with `load_witness(F::ZERO)`. This will force the unused results and subqueries to be 0, and remove this possible source of randomness from unused public inputs, without affecting the proving cost.

Axiom: Implemented the suggestion in commit a05a1289.

Spearbit: Fixed.

5.3.5 Correction on expected range for `num_blocks` field comment

Severity: Informational

Context: axiom-eth-working/axiom-core/src/header_chain.rs#L38

Description: The commented range, `[0, 2**max_depth)` for the field `num_blocks` is slightly off.

Recommendation: The range should be changed to the actual expression: `[1, 2**max_depth]`.

Axiom: Fixed in PR 209.

Spearbit: Fixed.

5.3.6 Improvable doc comment over decorator keccak padding

Severity: Informational

Context: axiom-crypto/axiom-eth-working/axiom-eth/src/utils/keccak/decorator.rs

Description: The code comment here is:

```
I don't think ensure_0_padding is necessary, but including for safety
```

This doesn't need to be added "*Just In Case*". But rather, should always be added to ensure that `packed_input` after variable length `bytes.len()` corresponds to `format_input` of `[]` empty bytes

Recommendation: Update the comment with the aforementioned details.

Axiom: Fixed in PR 159.

Spearbit: Fixed.

5.3.7 Remove unused `dyn_lookup` in `RlcKeccakConfig`

Severity: Informational

Context: axiom-crypto/axiom-eth-working/axiom-eth/src/utils/keccak/decorator.rs

Description: Within the aforementioned file, the `dyn_lookup` attribute is created but dropped after without any usage of it done. If it's not going to be used, this can save some time/performance as well as other implications it might have internally at config time.

Recommendation: Simply remove it.

Axiom: Fixed in PR 158.

Spearbit: Fixed.