

Scroll l2geth (initial)

Security assessment by HashEye · prepared for Scroll

HASHEYE AUDITED

PROJECT	Scroll l2geth (initial)
CLIENT	Scroll
CATEGORY	Scroll
PUBLISHED	August 1, 2023
REPORT ID	research-scroll-l2geth-initial-2023-08-01-1mjtlv

This report was produced under HashEye's layered review process – **automated detection**, **pattern correlation**, and **senior manual verification** – with every finding signed off by a human reviewer. Full findings detail and on-chain attestation are available on the report page at hashey.io/audits/research-scroll-l2geth-initial-2023-08-01-1mjtlv.

ScrollzkEVMCircuits,Wave2 SecurityAssessment September8,2023 Preparedfor: HaichenShen Scroll
Preparedby:FilipeCasal,JoeDoyle,andMarcIlunga

About hashey Founded in 2012 and headquartered in New York, hashey 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/hashey-io/publications>, with links to papers, presentations, public audit reports, and podcast appearances. In recent years, hashey 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. hashey 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 hashey on Twitter and explore our public repositories at <https://github.com/hashey-io>. To engage us directly, visit our "Contact" page at <https://www.hashey.io/contact>, or email us at info@hashey.io. hashey, Inc. 228 Park Ave S #80688 New York, NY 10003 <https://www.hashey.io> info@hashey.io hashey
1 ScrollzkEVMCircuitsSecurityAssessment PUBLIC

Notices and Remarks Copyright and Distribution ©2023 by hashey, Inc. All rights reserved. hashey hereby asserts its right to be identified as the creator of this report in the United Kingdom. This report is considered by hashey to be public information; it is licensed to Scroll under the terms of the project statement of work and has been made public at Scroll's request. Material within this report may not be reproduced or distributed in part or in whole without the express written permission of hashey. This is the canonical source for hashey publications; the hashey Publications page. Reports accessed through any source other than that page may have been modified and should not be considered authentic. Test Coverage Disclaimer All activities undertaken by hashey 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. hashey 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. hashey
2 ScrollzkEVMCircuitsSecurityAssessment PUBLIC

Table of Contents About hashey 1 Notices and Remarks 2 Table of Contents 3 Executive Summary 4 Project Summary 6 Project Goals 7 Project Targets 8 Project Coverage 9 Automated Testing 10 Codebase Maturity Evaluation 12 Summary of Findings 14 Detailed Findings 15 1. Poseidon Lookup is not implemented 15
2. IsZeroGadget does not constrain the inverse witness when the value is zero 17
3. The MPT nonexistence proof gadget is missing constraint specified in the documentation 19
4. Discrepancies between the MPT circuits specification and implementation 21
5. Redundant lookups in the WordRLC circuit 24
6. The Nonce Changed configuration circuit does not constrain the new value nonce value 26
7. The Copy circuit does not totally enforce the tag values 28
8. The "invalid creation" error handling circuit is unconstrained 31
9. The OneHot primitive allows more than one value at once 33 10. Intermediate columns are not explicit 35
A. Vulnerability Categories 37 B. Code Maturity Categories 39 C. Code Quality Findings 40
D. Automated Analysis Tool Configuration 44 E. Fix Review Results 46 Detailed Fix Review Results 48
F. Fix Review Status Categories 50 hashey 3 ScrollzkEVMCircuitsSecurityAssessment PUBLIC

Executive Summary Engagement Overview

ScrollengagedhasheyetoreviewthesecurityofasetofzkEVMcircuits:theMerkle Patricia trie (MPT) circuit, the Copycircuit, word-addressable memory optimizations, and the Supercircuit. A team of three consultants conducted the review from July 17 to August 4, 2023, for a total of six engineer-weeks of effort. Our testing efforts focused on ensuring that the circuits are sound, complete, and faithful implementations of the specifications. With full access to the source code and documentation, we performed static and dynamic testing of the codebase, using automated and manual processes. Observations and Impact

This security review of the codebase uncovered three high-severity issues related to the soundness of the circuits: finding TOB-SCROLL2-6 affects the MPT circuit and allows attackers to bypass checks when creating or updating the MPT node containing the account nonce and code size; finding TOB-SCROLL2-8 allows attackers to hijack the EVM control flow after the CREATE opcode is called, allowing them to redirect a successful EVM execution to a halt with an error state; and finding TOB-SCROLL2-9 affects the OneHot primitive and may allow a malicious prover to skip constraints in the MPT update circuit.

We found several inconsistencies between the MPT circuit specification and its implementation (TOB-SCROLL2-3, TOB-SCROLL2-4, and TOB-SCROLL2-7). We also found some issues involving unresolved "TODO" comments and stub implementations used for testing, including the informational finding TOB-SCROLL2-1 and the high-severity finding TOB-SCROLL2-9. Recommendations

Based on the codebase maturity evaluation and findings identified during this security review, hashey recommends that Scroll take the following steps:

- Remediate the findings disclosed in this report. These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.

- Revise the circuit specifications. During the engagement, we found several inconsistencies between the specifications and the circuit implementations. The documentation also lacks high-level descriptions of how the circuits work. Higher-level descriptions would greatly aid developers and auditors in quickly understanding the inner workings of the circuits and allow them to more easily identify potential structural flaws. After the specifications are revised and clarified, ensure that the circuit implementations are faithful to the updated specifications.

- Implement branch/leaf domain separation in the Merkle tree redesign. The Merkle tree redesign implemented in the MPT circuit does not correctly include domain separation between branch and leaf nodes, which could enable Merkle proof forgery. We did not include this flaw as a finding in this report because we found it during our separate audit of the zktrie library, but the fix will need to be implemented in both zktrie and the MPT circuit.
- Address the "TODO" comments in the code and test with non-stub implementations. Although they are useful for rapid development and testing, "TODO" comments and testing with stub implementations allow flaws to remain in the codebase. Each "TODO" should be tracked and triaged in a centralized issue tracker (e.g., GitHub issues), not in code comments, and should be fixed if needed; each mock implementation should be associated with a full implementation that can be used in occasional full test suite runs.

- Consider refactoring the MPT update verification code to separate Merkle path-checking logic from proof-specific logic. The current circuit simultaneously checks Merkle paths and checks various proof-specific attributes of the leaves. This

design makes the circuit fairly complex, with two simultaneous state machines modeled within the table and many repeated checks, which we believe has contributed to copy-paste problems such as finding TOB-SCROLL2-6. Separating the

Merkle path checks from the specific checks used for specific update types would improve modularity and may make any future required modifications easier to do.

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

Severity	Count
High	3
Medium	0
Low	0
Informational	7
Undetermined	0

Category	Count
Cryptography	9
Testing	1

hashey 5 Scroll zkEVM Circuits Security Assessment PUBLIC

Project Summary Contact Information The following managers were associated with this project:

Dan Guido, Account Manager Brooke Langhorne, Project Manager dan@hashey.io brooke.langhorne@hashey.io The following engineers were associated with this project: Filipe Casal, Consultant Joe Doyle, Consultant filipe.casal@hashey.io joseph.doyle@hashey.io Marc Ilunga, Consultant marc.ilunga@hashey.io

Project Timeline The significant events and milestones of the project are listed below. Date Event

July 13, 2023	Pre-project kickoff call
July 21, 2023	Status update meeting #1
July 28, 2023	Status update meeting #2
August 7, 2023	Delivery of report draft
August 7, 2023	Report readout meeting
September 8, 2023	Delivery of comprehensive report

October10,2023Deliveryofcomprehensivereportwithfixreviewappendix hashey
6ScrollzkEVMCircuitsSecurityAssessment PUBLIC

ProjectGoals TheengagementwasscopedtoprovideasecurityassessmentofasubsetofScroll'szkEVM circuits. Specifically, wesoughttoanswerthefollowingnon-exhaustivelistofquestions: • DoesthecodebasefollowbestpracticesforRust? • Arethecircuitssoundandcomplete? • AretheMPTcircuitandword-addressablememoryoptimizationRust implementationsfaithfultotheirsSpecifications? • AretheMPTpath-checkingstatemachinescorrectlydesignedandimplemented? Arethecorrecttransitionsenforced? Isitpossibletoskiprequiredchecksby manipulatingthestatemachines? • Aretheword-addressableoptimizationsound? Has the associatedcoderefactoring introducedanyvulnerabilitiesintothecodebase? hashey 7ScrollzkEVMCircuitsSecurityAssessment PUBLIC

ProjectTargets Theengagementinvolvedareviewandtestingofthefollowingtargets. mpt-circuit Repository<https://github.com/scroll-tech/mpt-circuit/tree/v0.4> Version 7b56d0b323e92ac11e54213520f6e7db41941cd0 TypesRust,haLo2 PlatformNative copy-circuit Repository<https://github.com/scroll-tech/zkevm-circuits> Version fc6c8a2972870e62e96cde480b3aa48c0cc1303d TypesRust,haLo2 PlatformNative super_circuit.rs Repository<https://github.com/scroll-tech/zkevm-circuits> Version fc6c8a2972870e62e96cde480b3aa48c0cc1303d TypesRust,haLo2 PlatformNative hashey 8ScrollzkEVMCircuitsSecurityAssessment PUBLIC

ProjectCoverage Thissectionprovidesanoverviewoftheanalysiscoverageofthereview,asdeterminedby ourhigh-levelengagementgoals. Ourapproachesincludedthefollowing: • TheMPTcircuit:Wemanuallyreviewedthecircuitutilitiesandtheupdate-checking circuitin mpt_update.rs . Ifweidentifiedvulnerablecodepatternsorcodesmells, weusedSemgreptoperformvariantanalysisandtosearchforotherinstancesof thesamepatterns. Wecomparedtheimplementationwiththespecificationand reportedtheidentifieddifferences. • TheCopycircuitandword-addressablememoryoptimizations:Wereviewedthe memoryoptimizationspecificationforsoundnessandmanuallyreviewedtheCopy circuitandassociatedgadgetsagainsttheCopycircuitspecificationdocument. Additionally,wereviewedtherefactoringthatneededtobedoneonseveralzkEVM circuitsduetotheintroducedoptimizations. • TheSupercircuit:Wemanuallyreviewedthecircuitforincorrectcircuit initializationissues. hashey 9ScrollzkEVMCircuitsSecurityAssessment PUBLIC

AutomatedTesting hasheyusesautomatedtechniquesextensivelytestthesecuritypropertiesof software. Weusebothopen-sourcestaticanalysisandfuzzingutilities,alongwithtools developedinhouse,toperformautomatedtestingofsourcecodeandcompiledsoftware. TestHarnessConfiguration Weusedthefollowingtoolsintheautomatedtestingphaseofthisproject: ToolDescriptionPolicy SemgrepAnopen-sourcestaticanalysistoolforfindingbugsand enforcingcodestandardswheneditingorcommittingcode andduringbuildtime AppendixD.1 cargo-llvm -cov AtoolforgeneratingtestcoveragereportsinRustAppendixD.2 cargo-edit AtoolforquicklyidentifyingoutdatedcratesAppendixD.3 ClippyAnopen-sourceRustlinterusedtocatchcommonmistakes andunidiomaticRustcode AppendixD.4 AreasofFocus Ourautomatedtestingandverificationworkfocusedonidentifyingthefollowing: • Generalcodequalityissuesandunidiomaticcodepatterns • Untestedcoderegionswithcoveragereports • VariantsofmanuallyfoundvulnerableAPIpatterns TestResults Theresultsofthisfocusedtestingaredetailedbelow. Clippy MPTcircuit:RunningClippyinpedanticmodeontheMPTcircuitrevealsseveralcode qualityissuesandunidiomaticcodepatterns,includingthefollowing: semicolon_if_nothing_returned , match_same_arms , needless_pass_by_value , hashey 10ScrollzkEVMCircuitsSecurityAssessment PUBLIC inconsistent_struct_constructor , return_self_not_must_use ,and map_unwrap_or . Werecommendroutinely(e.g.,everyminorrelease)runningClippyinpedanticmode. If certainpatternsarecommonlyfound,consideraddingtheserulestothe defaultClippy runs. Semgrep TherulesthatwewrotetofindvariantsofmanuallyfoundvulnerableAPIpatternsare providedinappendixD. cargo-edit Running cargo-edit with cargoupgrade--incompatible--dry-run findsfive outdatedcratesusedbytheMPTcircuitcodebase: ethers-core , itertools , strum , strum_macros ,and rand_chacha . cargo-llvm-cov ThecoveragereportfortheMPTcircuitindicatesseveralcoveragelimitations. hashey 11ScrollzkEVMCircuitsSecurityAssessment PUBLIC

CodebaseMaturityEvaluation hasheyusesatraffic-lightprotocoltoprovideeachclientwithaclearunderstandingof theareasinwhichitscodebaseismature,immature,orunderdeveloped. Deficiencies identifiedhereoftenstemfromrootcauseswithinthesoftwaredevelopmentlifecylethat

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 found no issues related to the use of integer arithmetic in the code base. Satisfactory Complexity Management The general structure of the implementation is well organized and separated into clear modules. However, the state machine-based design of the MPT circuit is fairly complex, and the Merkle path verification code is tightly coupled with the code that validates the data stored within a given leaf. Two state machines govern Merkle path checking, each of whose state transitions are different depending on the proof type. Each proof type also has some repetitive but slightly changed constraints, such as the constraints enforcing the exact path through an account leaf. Moderate Cryptography and Key Management The MPT design implemented in this circuit is vulnerable to proof forgery attacks, as disclosed in a previous audit of the zktrie library. The implementation will need to be updated so that it has a secure design before it is suitable for deployment. Weak Documentation Both the MPT and the Copy circuit, as well as the word-addressable memory optimizations, have associated specification documents. However, we found several discrepancies between the specifications and implementations. We recommend that the specifications be revised and extended with high-level descriptions of the logic behind each circuit. Having the revised specifications will allow developers to have a clearer understanding of the circuits. Moderate has eye 12 Scroll zkEVM Circuits Security Assessment PUBLIC

Memory Safety and Error Handling The zkEVM circuits project uses no unsafe Rust code. Satisfactory Testing and Verification We identified some test-coverage limitations in the MPT circuit code base by using the cargo-llvm-cov tool. Several of the findings that we identified in this review could have been detected with a comprehensive positive and negative tests suite. Because we found high-severity findings related to circuit soundness, we believe it is necessary to develop an adversarial testing process, especially focused on malicious prover behavior. Formal methods to check for (e.g., circuit determinacy) should also be investigated and considered. Weak has eye 13 Scroll zkEVM Circuits Security Assessment PUBLIC

Summary of Findings The table below summarizes the findings of the review, including type and severity details. ID Title Type Severity 1 Poseidon Lookup is not implemented Testing Informational 2 Is Zero Gadget does not constrain the inverse witness when the value is zero Cryptography Informational 3 The MPT nonexistence proof gadget is missing constraint specified in the documentation Cryptography Informational 4 Discrepancies between the MPT circuit specification and implementation Cryptography Informational 5 Redundant lookups in the WordRLC circuit Cryptography Informational 6 The Nonce Changed configuration circuit does not constrain the new value nonce value Cryptography High 7 The Copy circuit does not totally enforce the tag values Cryptography Informational 8 The "invalid creation" error handling circuit is unconstrained Cryptography High 9 The One Hot primitive allows more than one value at once Cryptography High 10 Intermediate columns are not explicit Cryptography Informational has eye 14 Scroll zkEVM Circuits Security Assessment PUBLIC

Detailed Findings 1. Poseidon Lookup is not implemented Severity: Informational Difficulty: N/A Type: Testing Finding ID: TOB-SCROLL2-1 Target: src/gadgets/poseidon.rs Description Poseidon hashing is performed within the MPT circuit by performing lookups into a Poseidon table via the PoseidonLookup trait, shown in figure 1.1. // Lookup represents the poseidon table in zkvm circuit pub trait PoseidonLookup { fn lookup_columns(&self) -> (FixedColumn, [AdviceColumn; 5]); let (fixed, adv) = self.lookup_columns_generic(); (FixedColumn(fixed), adv.map(AdviceColumn)) } fn lookup_columns_generic(&self) -> (Column<Fixed>, [Column<Advice>; 5]); let (fixed, adv) = self.lookup_columns(); (fixed.0, adv.map(|col| col.0)) } } Figure 1.1: src/gadgets/poseidon.rs#11-21 This trait is not implemented by any type except the testing-only PoseidonTable shown in figure 1.2, which does not constrain its columns at all. #[cfg(test)] #[derive(Clone, Copy)] pub struct PoseidonTable { q_enable: FixedColumn, left: AdviceColumn, right: AdviceColumn, hash: AdviceColumn, control: AdviceColumn, head_mark: AdviceColumn, } #[cfg(test)] impl PoseidonTable { pub fn configure<F: FieldExt>(cs: &mut ConstraintSystem<F>) -> Self { has eye 15 Scroll zkEVM Circuits Security Assessment PUBLIC

let [hash, left, right, control, head_mark] = [0; 5].map(|_| AdviceColumn(cs.advice_column())); Self { left, right, hash, control, head_mark, q_enable: FixedColumn(cs.fixed_column()), } } Figure 1.2: src/gadgets/poseidon.rs#56-80 The rest of the code base treats this trait as a black-box implementation, so this does not seem to cause correctness problems elsewhere. However, it does limit one's ability to test some negative cases, and it makes the test coverage rely on the correctness of the PoseidonTable struct's witness generation. Recommendations Short term, create a concrete implementation of the PoseidonLookup trait to enable full testing of the MPT circuit. Long term, ensure that all parts of the MPT circuit are tested with both positive and negative tests. has eye 16 Scroll zkEVM Circuits Security Assessment PUBLIC

2. IsZeroGadget does not constrain the inverse witness when the value is zero

Severity: Informational Difficulty: N/A Type: Cryptography Finding ID: TOB-SCROLL2-2 Target: src/gadgets/is_zero.rs Description The IsZeroGadget implementation allows for an arbitrary inverse_or_zero witness value when the value parameter is 0. The gadget returns 1 when value is 0; otherwise, it returns 0. The implementation relies on the existence of an inverse for when value is non-zero and on correctly constraining that $value * (1 - value * inverse_or_zero) = 0$. However, when value is 0, the constraint is immediately satisfied, regardless of the value of the inverse_or_zero witness. This allows an arbitrary value to be provided for that witness value. `pub fn configure<F: FieldExt>(cs: &mut ConstraintSystem<F>, cb: &mut ConstraintBuilder<F>, value: AdviceColumn, // TOD0: make this a query once Query is cloneable/copyable....) -> Self { let inverse_or_zero = AdviceColumn(cs.advice_column()); cb.assert_zero("value is 0 or inverse_or_zero is inverse of value", value.current() * (Query::one() - value.current()) * inverse_or_zero.current(),); Self { value, inverse_or_zero, } } Figure 2.1: mpt-circuit/src/gadgets/is_zero.rs#48-62 Recommendations Short term, ensure that the circuit is deterministic by constraining inverse_or_zero to equal 0 when value is 0. hashey 17 ScrollzkEVMCircuitsSecurityAssessment PUBLIC`

Long term, document which circuits have non-deterministic witnesses; over time, constrain them so that all circuits have deterministic witnesses. `hashey 18 ScrollzkEVMCircuitsSecurityAssessment PUBLIC`

3. The MPT non-existence proof gadget is missing constraint specifications in the documentation

Severity: Informational Difficulty: N/A Type: Cryptography Finding ID: TOB-SCROLL2-3 Target: src/gadgets/mpt_update/nonexistence_proof.rs Description The gadget for checking the consistency of non-existence proofs is missing several constraints related to type 2 non-existence proofs. The circuit specification includes constraints for the non-existence of path proofs that are not included in the implementation. This causes the witness values to be unconstrained in some cases. For example, the following constraints are specified: • other_key_hash should equal 0 when key does not equal other_key. • other_leaf_data_hash should equal the hash of the empty node (pointer by other_key). Neither of these constraints is enforced in the implementation: this is because the implementation has no explicit constraints imposed for the type 2 non-existence proofs. Figure 3.1 shows that the circuit constrains these values only for type 1 proofs. `pub fn configure<F: FieldExt>(cb: &mut ConstraintBuilder<F>, value: SecondPhaseAdviceColumn, key: AdviceColumn, other_key: AdviceColumn, key_equals_other_key: IsZeroGadget, hash: AdviceColumn, hash_is_zero: IsZeroGadget, other_key_hash: AdviceColumn, other_leaf_data_hash: AdviceColumn, poseidon: &impl PoseidonLookup,) { cb.assert_zero("value is 0 for empty node", value.current()); cb.assert_equal("key_minus_other_key=key-otherkey", key_equals_other_key.value.current(), key.current() - other_key.current(),); cb.assert_equal("hashey 19 ScrollzkEVMCircuitsSecurityAssessment PUBLIC`

`"hash_is_zero input = hash", hash_is_zero.value.current(), hash.current(),); let is_type_1 = !key_equals_other_key.current(); let is_type_2 = hash_is_zero.current(); cb.assert_equal("Empty account is either type 1 or type 2", Query::one(), Query::from(is_type_1.clone()) + Query::from(is_type_2),); cb.condition(is_type_1, |cb| { cb.poseidon_lookup("other_key_hash = h(1, other_key)", [Query::one(), other_key.current(), other_key_hash.current()], poseidon,); cb.poseidon_lookup("hash = h(key_hash, other_leaf_data_hash)", [other_key_hash.current(), other_leaf_data_hash.current(), hash.current(),], poseidon,); }); Figure 3.1: mpt-circuit/src/gadgets/mpt_update/nonexistence_proof.rs#7-54`

The Scroll team has stated that this is a specification error and that the missing constraints do not impact the soundness of the circuit. Recommendations

Short term, update the specification to remove the description of these constraints; ensure that the documentation is kept updated.

Long term, add positive and negative tests for both types of non-existence proofs. `hashey 20 ScrollzkEVMCircuitsSecurityAssessment PUBLIC`

4. Discrepancies between the MPT circuit specification and implementation

Severity: Informational Difficulty: N/A Type: Cryptography Finding ID: TOB-SCROLL2-4 Target: Several files Description The MPT circuit implementation is not faithful to the circuit specification in many areas and does not contain comments for the constraints that are either missing from the implementation or that diverge from those in the specification. The allowed segment transitions depend on the proof type. For the NonceChanged proof type, the specification states that the Start segment type can transition to Start and that the AccountLeaf0 segment type also can transition to Start. However, neither of these paths is allowed in the implementation. `MPTProofType::NonceChanged | MPTProofType::BalanceChanged | MPTProofType::CodeSizeExists`

```
|MPTProofType::CodeHashExists=>[ ( SegmentType::Start, vec![
SegmentType::AccountTrie,//mptas>1account SegmentType::AccountLeaf0,//mptas≤1account ], ), (
SegmentType::AccountTrie, vec![ SegmentType::AccountTrie, SegmentType::AccountLeaf0,
SegmentType::Start,//emptyaccountproof ], ), (SegmentType::AccountLeaf0,vec!
[SegmentType::AccountLeaf1]), (SegmentType::AccountLeaf1,vec![SegmentType::AccountLeaf2]),
(SegmentType::AccountLeaf2,vec![SegmentType::AccountLeaf3]), (SegmentType::AccountLeaf3,vec!
[SegmentType::Start]), Figure4.1: mpt-circuit/src/gadgets/mpt_update/segment.rs#20-42 hashey
21ScrollzkEVMCircuitsSecurityAssessment PUBLIC
```

Figure4.2:PartoftheMPTspecification(spec/mpt-proof.md#L318-L328) Thetransitionsallowedforthe PoseidonCodeHashExists prooftypealsodonotmatch:

thespecificationstatesthatithasthesametransitionsasthe NonceChanged prooftype, buttheimplementationhasdifferenttransitions.

Thekeydepthdirectionchecksalsodonotmatchthespecification.Thespecificationstates thatthe depth parametersshouldbeusedbuttheimplementationuses depth-1 . cb.condition(is_trie.clone(),|cb|{ cb.add_lookup("directionisincorrectforkeyanddepth",

```
[key.current(),depth.current()-1,direction.current()], key_bit.lookup(), ); cb.assert_equal(
"depthincreasesby1intriesegments", depth.current(), depth.previous()+1, );
```

```
cb.condition(path_type.current_matches(&[PathType::Common]),|cb|{ cb.add_lookup(
"directionisincorrectforother_keyanddepth", [ other_key.current(), depth.current()-1, Figure4.3: mpt-
circuit/src/gadgets/mpt_update.rs#188-205 hashey 22ScrollzkEVMCircuitsSecurityAssessment PUBLIC
```

Figure4.4:PartoftheMPTspecification(spec/mpt-proof.md#L279-L282)

Finally,thespecificationstatesthatwhenasegmenttypeisanon-trietype,thevalueof key shouldbeconstrainedto 0 ,butthisconstraintisomittedfromtheimplementation.

```
cb.condition(!is_trie,|cb|{ cb.assert_zero("depthis0innon-triesegments",depth.current()); });
```

Figure4.5: mpt-circuit/src/gadgets/mpt_update.rs#212-214 Figure4.6:PartoftheMPTspecification(spec/mpt-proof.md#L284-L286) Recommendations

Shortterm,reviewthespecificationandensureitsconsistency.Matchtheimplementation withthespecification,anddocumentpossibleoptimizationsthatremoveconstraints, detailingwhytheydonotcausesoundnessissues.

Longterm,includebothpositiveandnegativetestsforalledgecasesinthespecification. hashey 23ScrollzkEVMCircuitsSecurityAssessment PUBLIC

5.RedundantlookupsintheWordRLCircuit Severity:InformationalDifficulty:N/A

Type:CryptographyFindingID:TOB-SCROLL2-5 Target: src/gadgets/mpt_update/word_rlc.rs Description TheWordRLCircuit hastworedundantlookupsintothe BytesLookup table.

TheWordRLCircuit combinestherandomlinearcombination(RLC)forthe lowerand upper16bytesofawordintoasinglerlcvalue.Forthis,itchecksthatthelowerand upperwordsegmentsare16bytesbylookingintothe BytesLookup table,anditchecks

thattheirRLCsarecorrectlycomputedbylookingintothe RlcLookup table.However,the lookupintothe RlcLookup tablewillalsoensurethattheloweranduppersegmentsofthe

wordhavethecorrect16bytes,makingthefirsttwolookupsredundant. pubfnconfigure<F:FieldExt>(

```
cb:&mutConstraintBuilder<F>, [word_hash,high,low]:[AdviceColumn;3], [rlc_word,rlc_high,rlc_low]:
```

```
[SecondPhaseAdviceColumn;3], poseidon:&implPoseidonLookup, bytes:&implBytesLookup,
```

```
rlc:&implRlcLookup, randomness:Query<F>, ){ cb.add_lookup( "old_highis16bytes",
```

```
[high.current(),Query::from(15)], bytes.lookup(), ); cb.add_lookup( "old_lowis16bytes",
```

```
[low.current(),Query::from(15)], bytes.lookup(), ); cb.poseidon_lookup(
```

```
"word_hash=poseidon(high,low)", [high.current(),low.current(),word_hash.current()], poseidon, );
```

```
cb.add_lookup( "rlc_high=rlc(high)andhighis16bytes",
```

```
[high.current(),Query::from(15),rlc_high.current()], hashey
```

```
24ScrollzkEVMCircuitsSecurityAssessment PUBLIC
```

```
rlc.lookup(), ); cb.add_lookup( "rlc_low=rlc(low)andlowis16bytes",
```

```
[low.current(),Query::from(15),rlc_low.current()], rlc.lookup(), Figure5.1: mpt-
```

```
circuit/src/gadgets/mpt_update/word_rlc.rs#16-49 Althoughthe WordRlc::configure
```

```
functionreceivestwodifferentlookupobjects, bytes and rlc
```

```
,theyareinstantiatedwiththesameconcretelookup: letmpt_update=MptUpdateConfig::configure( cs,
```

```
&mutcb, poseidon, &key_bit, &byte_representation, &byte_representation, &rlc_randomness,
```

```
&canonical_representation, ); Figure5.2: mpt-circuit/src/mpt.rs#60-69
```

Wealsonotethatthelabelsrefertotheupperandlowerbytesas old_high and old_low insteadofjust high and low . Recommendations Shortterm,determinewhetherboththe BytesLookup and RlcLookup tablesareneeded forthiscircuit,andrefactorthecircuitaccordingly,removingtheredundantconstraints.

Longterm,reviewthecodebaseforduplicatedorredundantconstraintsusingmanualand automatedmethods.

hashey 25ScrollzkEVMCircuitsSecurityAssessment PUBLIC

6. The NonceChanged configuration circuit does not constrain the new value. Severity: High Difficulty: Low Type: Cryptography Finding ID: TOB-SCROLL2-6 Target: src/gadgets/mpt_update.rs Description: The NonceChanged configuration circuit does not constrain the config.new_value parameter to be 8 bytes. Instead, there is a duplicated constraint for config.old_value: SegmentType::AccountLeaf3 => { cb.assert_zero("direction is 0", config.direction.current()); let old_code_size = (config.old_hash.current() - config.old_value.current()); *Query::Constant(F::from(1 << 32).square().invert().unwrap()); let new_code_size = (config.new_hash.current() - config.new_value.current()); *Query::Constant(F::from(1 << 32).square().invert().unwrap()); cb.condition(config.path_type.current_matches(&[PathType::Common]), |cb| { cb.add_lookup("old nonce is 8 bytes", [config.old_value.current(), Query::from(7)], bytes.lookup(),); cb.add_lookup("new nonce is 8 bytes", [config.old_value.current(), Query::from(7)], bytes.lookup(),); Figure 6.1: mpt-circuit/src/gadgets/mpt_update.rs#1209-1228 This means that a malicious prover could update the Account node with a value of arbitrary length for the Nonce and Code size parameters. The same constraint (with a correct label but an incorrect value) is used in the ExtensionNew path type: cb.condition(config.path_type.current_matches(&[PathType::ExtensionNew]), |cb| { cb.add_lookup("new nonce is 8 bytes", [config.old_value.current(), Query::from(7)], bytes.lookup(),); Figure 6.2: mpt-circuit/src/gadgets/mpt_update.rs#1241-1248 Exploit Scenario: A malicious prover uses the NonceChanged proof to update the nonce with a larger than expected value. Recommendations: Short term, enforce the constraint for the config.new_value witness. Long term, add positive and negative testing of the edge cases present in the specification. For both the Common and ExtensionNew path types, there should be a negative test that fails because it changes the new nonce to a value larger than 8 bytes. Use automated testing tools like Semgrep to find redundant and duplicate constraints, as these could indicate that a constraint is incorrect. hashey 26 ScrollzkEVM Circuits Security Assessment PUBLIC

7. The Copy circuit does not totally enforce the tag values. Severity: Informational Difficulty: N/A Type: Cryptography Finding ID: TOB-SCROLL2-7 Target: src/copy_circuit/copy_gadgets.rs Description: The Copy table includes a tag column that indicates the type of data for that particular row. However, the Copy circuit tag validation function does not totally ensure that the tag matches one of the predefined tag values. The implementation uses the copy_gadgets::constrain_tag function to bind the is_precompiled, is_tx_calldata, is_bytecode, is_memory, and is_tx_log witnesses to the actual tag value. However, the code does not ensure that exactly one of these Boolean values is true. #[allow(clippy::too_many_arguments)] pub fn constrain_tag<F: Field>(meta: &mut ConstraintSystem<F>, q_enable: Column<Fixed>, tag: BinaryNumberConfig<CopyDataType, 4>, is_precompiled: Column<Advice>, is_tx_calldata: Column<Advice>, is_bytecode: Column<Advice>, is_memory: Column<Advice>, is_tx_log: Column<Advice>,){ meta.create_gate("decode tag", |meta| { let enabled = meta.query_fixed(q_enable, CURRENT); let is_precompile = meta.query_advice(is_precompiled, CURRENT); let is_tx_calldata = meta.query_advice(is_tx_calldata, CURRENT); let is_bytecode = meta.query_advice(is_bytecode, CURRENT); let is_memory = meta.query_advice(is_memory, CURRENT); let is_tx_log = meta.query_advice(is_tx_log, CURRENT); let precompiles = sum::expr([tag.value_equals(CopyDataType::Precompile(PrecompileCalls::Erecover), CURRENT,)(meta), tag.value_equals(CopyDataType::Precompile(PrecompileCalls::Sha256), CURRENT,)(meta), tag.value_equals(CopyDataType::Precompile(PrecompileCalls::Ripemd160), CURRENT, hashey 28 ScrollzkEVM Circuits Security Assessment PUBLIC

)(meta), tag.value_equals(CopyDataType::Precompile(PrecompileCalls::Identity), CURRENT,)(meta), tag.value_equals(CopyDataType::Precompile(PrecompileCalls::Modexp), CURRENT,)(meta), tag.value_equals(CopyDataType::Precompile(PrecompileCalls::Bn128Add), CURRENT,)(meta), tag.value_equals(CopyDataType::Precompile(PrecompileCalls::Bn128Mul), CURRENT,)(meta), tag.value_equals(CopyDataType::Precompile(PrecompileCalls::Bn128Pairing), CURRENT,)(meta), tag.value_equals(CopyDataType::Precompile(PrecompileCalls::Blake2F), CURRENT,)(meta),]); vec![// Match boolean indicators to their respective tag values. enabled.expr() * (is_precompile - precompiles), enabled.expr() * (is_tx_calldata - tag.value_equals(CopyDataType::TxCalldata, CURRENT,)(meta)), enabled.expr() * (is_bytecode - tag.value_equals(CopyDataType::Bytecode, CURRENT,)(meta)), enabled.expr() * (is_memory - tag.value_equals(CopyDataType::Memory, CURRENT,)(meta)), enabled.expr() * (is_tx_log - tag.value_equals(CopyDataType::TxLog, CURRENT,)(meta)),]; } Figure 7.1: copy_circuit/copy_gadgets.rs#13-62 In fact, the tag value could equal CopyDataType::RlcAcc, as in the SHA3 gadget. The CopyDataType::Padding value is also not currently matched. In the current state of the codebase, this issue does not appear to cause any soundness issues because the lookups into the Copy table either use a statically set source and destination tag, as in the case of precompiles, the value is correctly bounded and does

notposeanavenueofattackforamaliciousprover.

WealsoobservethattheCopycircuitspecificationmentionsawitnessvalueforthe `is_rlc_acc` case, but this is not reflected in the code. Recommendations

Short term, ensure that the tag column is fully constrained. Review the circuit specification and match the implementation with the specification, documenting possible optimizations that remove constraints and detailing why they do not cause soundness issues. hashey
29 ScrollzkEVMCircuitsSecurityAssessment PUBLIC

Long term, include negative tests for an unintended tag value. hashey
30 ScrollzkEVMCircuitsSecurityAssessment PUBLIC

8. The "invalid creation" error handling circuit is unconstrained Severity: High Difficulty: Medium
Type: Cryptography Finding ID: TOB-SCROLL2-8 Target:

`evm_circuit/execution/error_invalid_creation_code.rs` Description

The "invalid creation" error handling circuit does not constrain the first byte of the actual memory to be `0xef` as intended. This allows a malicious prover to redirect the EVM execution to a halt after the `CREATE` opcode is called, regardless of the memory value. The `ErrorInvalidCreationCodeGadget` circuit was updated to accommodate the memory addressing optimizations. However, in doing so, the `first_byte` witness value that was bound to the memory's first byte is no longer bound to it. Therefore, a malicious prover can always satisfy the circuit constraints, even if they are not in an error state after the `CREATE` opcode is called. `fn configure(cb: &mut EVMConstraintBuilder<F>) -> Self { let opcode = cb.query_cell(); let first_byte = cb.query_cell(); // let address = cb.query_word_rlc(); let offset = cb.query_word_rlc(); let length = cb.query_word_rlc(); let value_left = cb.query_word_rlc(); cb.stack_pop(offset.expr()); cb.stack_pop(length.expr()); cb.require_true("is_create is true", cb.curr.state.is_create.expr()); let address_word = MemoryWordAddress::construct(cb, offset.clone()); // lookup memory for first word cb.memory_lookup(0.expr(), address_word.addr_left(), value_left.expr(), value_left.expr(), None,); // let first_byte = value_left.cells[address_word.shift()]; // constrain first byte is 0xef let is_first_byte_invalid = IsEqualGadget::construct(cb, first_byte.expr(), hashey
31 ScrollzkEVMCircuitsSecurityAssessment PUBLIC`

`0xef.expr()); cb.require_true("is_first_byte_invalid is true", is_first_byte_invalid.expr(),);`

Figure 8.1: `evm_circuit/execution/error_invalid_creation_code.rs#36-67` Exploit Scenario

A malicious prover generates two different proofs for the same transaction, one leading to the error state, and the other successfully executing the `CREATE` opcode. Distributing these proofs to two ends of a bridge leads to a stated divergence and a loss of funds. Recommendations Short term, bind the `first_byte` witness value to the memory value; ensure that the successful `CREATE` end state checks that the first byte is different from `0xef`.

Long term, investigate ways to generate malicious traces that could be added to the test suite; every time a new soundness issue is found, create such a malicious trace and add it to the test suite. hashey
32 ScrollzkEVMCircuitsSecurityAssessment PUBLIC

9. The `OneHot` primitive allows more than one value at once Severity: High Difficulty: Low

Type: Cryptography Finding ID: TOB-SCROLL2-9 Target: `constraint_builder/binary_column.rs` Description

The `OneHot` primitive uses `BinaryQuery` values as witness values. However, despite their name, these values are not constrained to be Boolean values, allowing a malicious prover to choose more than one "hot" value in the data structure. `impl<T: IntoEnumIterator+Hash+Eq> OneHot<T> { pub fn configure<F: FieldExt>(cs: &mut ConstraintSystem<F>, cb: &mut ConstraintBuilder<F>,) -> Self { let mut columns = HashMap::new(); for variant in Self::nonfirst_variants() { columns.insert(variant, cb.binary_columns::<1>(cs)[0]); } let config = Self { columns }; cb.assert("sum of binary columns in OneHot is 0 or 1", config.sum(0).or(!config.sum(0)),); config } Figure 9.1: mpt-circuit/src/gadgets/one_hot.rs#14-30 Thereason the BinaryQuery values are not constrained to be Boolean is because the BinaryColumn configuration does not constrain the advice value to be Boolean, and the configuration is simply a type wrapper around the Column<Advice> type. This provides no guarantee to the users of this API, whom might assume that these values are guaranteed to be Boolean. pub fn configure<F: FieldExt>(cs: &mut ConstraintSystem<F>, _cb: &mut ConstraintBuilder<F>,) -> Self { let advice_column = cs.advice_column(); // TODO: constraint to be binary here ... // cb.add_constraint() Self(advice_column) hashey
33 ScrollzkEVMCircuitsSecurityAssessment PUBLIC`

} Figure 9.2: `mpt-circuit/src/constraint_builder/binary_column.rs#29-37` The `OneHot`

primitive is used to implement the Merkle path-checking state machine,

including critical properties such as requiring the `key` and `other_key` columns to remain

unchanged along a given Merkle path calculation, as shown in figure 9.3. `cb.condition(`

`!segment_type.current_matches(&[SegmentType::Start, SegmentType::AccountLeaf3]), |cb| { cb.assert_equal("key can only change on Start or AccountLeaf3 rows", key.current(), key.previous(),); cb.assert_equal("other_key can only change on Start or AccountLeaf3 rows", other_key.current(), other_key.previous(),); },); Figure 9.3: mpt-circuit/src/gadgets/mpt_update.rs#170-184`

Wedidnotdevelopaproof-of-conceptexploitforthepath-checkingtable,soitmaybethecasethattheconstraintinfigure9.3isnotexploitableduetootherconstraints.However,ifatanypointitispossibletomatchboth SegmentType::Start andsomeothersegment type(suchasbysettingone OneHot cellto 1 andanother to -1),amaliciousproverwould beabletochangethekeypartwaythroughandforgeMerkleupdates. ExploitScenario Amaliciousproverusethes OneHot soundnessissuetobypassconstraints,ensuringthat the key and other_key columnsremainunchangedalongagivenMerklepathcalculation. ThisallowstheattackertosuccessfullyforgeMPTupdateproofsthatupdateanarbitrary key. Recommendations Shortterm,addconstraintsthatensurethattheadvicevaluesfromthesecolumnsare Boolean. Longterm,addpositiveandnegativetestsensuringthattheseconstraintbuildersoperate accordingtotheirexpectations. hashey 34ScrollzkEVMCircuitsSecurityAssessment PUBLIC

10. Intermediatecolumnsarenotexplicit Severity:InformationalDifficulty:N/A
Type:CryptographyFindingID:TOB-SCROLL2-10 Target: src/mpt_update.rs Description
TheMPTupdatecircuitincludestwoarraysof"intermediatevalue"columns,asshownin figure10.1.
intermediate_values:[AdviceColumn;10],//canbe4? second_phase_intermediate_values:
[SecondPhaseAdviceColumn;10],//4? Figure10.1: mpt-circuit/src/gadgets/mpt_update.rs#65-66
Thesecolumnsareusedasgeneral-usecellsforvalueshatareonlyconditionallyneeded
inagivenrow, reducingthetotalnumberofcolumnsneeded. Forexample, figure10.2 showsthat
intermediate_values[0] isusedforthe address valueinrowsthatmatch SegmentType::Start
,butasshowninfigure10.3,rowsrepresentingthe SegmentType::AccountLeaf3 stateofaKeccakcode-
hashproofusethatsameslotfor the old_high value.
letaddress=self.intermediate_values[0].current()*is_start(); Figure10.2: mpt-
circuit/src/gadgets/mpt_update.rs#78 SegmentType::AccountLeaf3=>{
cb.assert_equal("directionis1",config.direction.current(),Query::one());
let[old_high,old_low,new_high,new_low,..]=config.intermediate_values; Figure10.3: mpt-
circuit/src/gadgets/mpt_update.rs#1632-1635 Insomecases,cellsof intermediate_values
areusedstartingfromtheendofthe intermediate_values column,suchasthe other_key_hash and
other_leaf_data_hash valuesin PathType::Extension0ld rows,asillustratedin figure10.4.
let[..,key_equals_other_key,new_hash_is_zero]=config.is_zero_gadgets;
let[..,other_key_hash,other_leaf_data_hash]=config.intermediate_values;
nonexistence_proof::configure(cb, hashey 35ScrollzkEVMCircuitsSecurityAssessment PUBLIC
config.new_value, config.key, config.other_key, key_equals_other_key, config.new_hash,
new_hash_is_zero, other_key_hash, other_leaf_data_hash, poseidon,); Figure10.4: mpt-
circuit/src/gadgets/mpt_update.rs#1036-1049
Althoughwedidnotfindanymistakesuchasmisusedcolumns,thispatternisadhocand error-
prone,andevaluatingthecorrectnessofthispatternrequirescheckingevery individualuseof
intermediate_values . Recommendations Shortterm,documenttheassignmenttofall intermediate_values
columnsineach relevantcase. Longterm,considerusingRusttypestoexpressthedifferentusesofthevarious
intermediate_values columns. Forexample,onecoulddefinean IntermediateValues enum,withcaseslike
StartRow{address:&AdviceColumn} and Extension0ld{other_key_hash:&AdviceColumn,other_leaf_data_hash:
&AdviceColumn} ,andasinglefunction fn parse_intermediate_values(segment_type:SegmentType,path_type:
PathType,columns:&[AdviceColumn;10])>IntermediateValues .Then,thecorrectassignmentanduseof
intermediate_values columnscanbeauditedonlyby checking parse_intermediate_values . hashey
36ScrollzkEVMCircuitsSecurityAssessment PUBLIC

A. VulnerabilityCategories

Thefollowingtablesdescribethevulnerabilitycategories,severitylevels,anddifficulty
levelsinthisdocument. VulnerabilityCategories CategoryDescription
AccessControlsInsufficientauthorizationorassessmentofrights
AuditingandLoggingInsufficientauditingofactionsorloggingofproblems
AuthenticationImproperidentificationofusers
ConfigurationMisconfiguredservers,devices,orsoftwarecomponents
CryptographyAbreachofsystemconfidentialityorintegrity DataExposureExposureofsensitiveinformation
DataValidationImproperrelianceonthestructureorvaluesofdata
DenialofServiceAsystemfailurewithanavailabilityimpact
ErrorReportingInsecureorinsufficientreportingoferrorconditions
PatchingUseofanoutdatedsoftwarepackageorlibrary
SessionManagementImproperidentificationofauthenticatedusers
TestingInsufficienttestmethodologyortestcoverage TimingRaceconditionsorotherorder-of-
operationsflaws UndefinedBehaviorUndefinedbehaviortriggeredwithinthelibrary hashey
37ScrollzkEVMCircuitsSecurityAssessment PUBLIC

SeverityLevels SeverityDescription
InformationalTheissuedoesnotposeanimmateriskbutisrelevanttosecuritybest practices.

UndeterminedThe extent of the risk was not determined during this engagement.
LowThe risk is small or is not one the client has indicated is important.
MediumUser information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
HighThe flaw could affect numerous users and have serious reputational, legal, or financial implications.

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

38 ScrollzkEVMCircuitsSecurityAssessment PUBLIC

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 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 code based 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	Are 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. hashey

39 ScrollzkEVMCircuitsSecurityAssessment PUBLIC

C. Code Quality Findings We identified the following code quality issues through manual and automatic code review.

- Allow the `dead_code` lint and fix all issues. The `dead_code` lint is currently disabled; it should be enabled to allow developers to quickly detect unused functions and variables. •
- Several constraints have meaningless labels. Constraint labels are useful for explaining the intention behind constraints; using meaningless labels hinders code readability. region
`.assign_fixed(||"asdfasdfawe", self.0, FigureC.1: mpt-circuit/src/constraint_builder/column.rs#52-55
cb.assert_equal("???????", rlc.current(), byte.current()); });
cb.condition(!index_is_zero.current(), |cb| { cb.assert_equal("value can only change when index=0",
value.current(), value.previous(),); cb.assert_equal("differences are zero so far=difference==0&&
differences_are_zero_so_far.previous() when index≠0", differences_are_zero_so_far.current().into(),
differences_are_zero_so_far .previous() .and(difference_is_zero.previous()) .into(),);
cb.assert_equal("???", FigureC.2: mpt-circuit/src/gadgets/canonical_representation.rs#72-89`
- There are duplicate and unused functions in the code base. The `types.rs` and `util.rs` files have several duplicate functions: `fr`, `hash`, `storage_key_hash`, `split_word`, `hi_lo`, and `Bit`. •
- The following constraint label was incorrectly copy-pasted. `cb.assert_equal(hashey`

40 ScrollzkEVMCircuitsSecurityAssessment PUBLIC

`"old_value does not change", new_value.current(), new_value.previous(),); FigureC.3: mpt-circuit/src/gadgets/mpt_update.rs#163-167` •

There are redundant constraints in empty storage/account constraints. The `configure_empty_storage` and `configure_empty_account` functions require the `new_value` and `old_value` fields to be 0 but also constrain them to be equal. `cb.assert_zero("old value is 0 for empty storage",
config.old_value.current(),); cb.assert_zero("new value is 0 for empty storage",
config.new_value.current(),); ... cb.assert_equal("old value = new value for empty account proof",
config.old_value.current(), config.new_value.current(),); FigureC.4: mpt-circuit/src/gadgets/mpt_update.rs#1785-1811`

`cb.assert_zero("old value is 0", config.old_value.current());
cb.assert_zero("new value is 0", config.new_value.current()); ... cb.assert_equal(
"old value = new value for empty account proof", config.old_value.current(), config.new_value.current(),);
FigureC.5: mpt-circuit/src/gadgets/mpt_update.rs#1869-1893` •

The following constraint label is imprecise. The label should read `rlc_inc_left[1]==rlc_inc_left[0]-rlc_diff`, or 0 at the end to match the code.

```

//Decrement rwc_inc_left for the next row, when an RW operation happens.
let rwc_diff = is_rw_type.expr()*is_word_end.expr();
let new_value = meta.query_advice(rwc_inc_left, CURRENT) - rwc_diff; //At the end, it must reach 0.
let update_or_finish = select::expr( not::expr(is_last.expr()),
meta.query_advice(rwc_inc_left, NEXT_ROW), 0.expr(), hashey 41 ScrollzkEVM Circuits Security Assessment
PUBLIC

); cb.require_equal( "rwc_inc_left[2] = rwc_inc_left[0] - rwc_diff, or 0 at the end", new_value,
update_or_finish, ); Figure C.6: src/copy_circuit/copy_gadgets.rs#524-537 • The IsZeroGadget assign
function does not assign the witness value. pub fn assign<F: FieldExt, T: Copy+TryInto<F>>( &self,
region: &mut Region<'_, F>, offset: usize, value: T, ) where <T as TryInto<F>>::Error: Debug, {
self.inverse_or_zero.assign( region, offset,
value.try_into().unwrap().invert().unwrap_or(F::zero()), ); }
//TODO: get rid of assign method in favor of fit. pub fn assign_value_and_inverse<F: FieldExt, T: Copy+TryInto<F>>
( &self, region: &mut Region<'_, F>, offset: usize, value: T, ) where <T as TryInto<F>>::Error: Debug, {
self.value.assign(region, offset, value); self.assign(region, offset, value); } Figure C.7: mpt-
circuit/src/gadgets/is_zero.rs#20-46 • The OneHot assign functions should ensure that no more than one item is
assigned. pub fn assign<F: FieldExt>( &self, region: &mut Region<'_, F>, offset: usize, value: T ) {
if let Some(c) = self.columns.get(&value) { c.assign(region, offset, true) } } Figure C.8: mpt-
circuit/src/gadgets/one_hot.rs#31-36 hashey 42 ScrollzkEVM Circuits Security Assessment PUBLIC

• There is a redundant condition in the configure_empty_account function. The function could just match
SegmentType::Start . SegmentType::Start | SegmentType::AccountTrie => { let is_final_segment =
config.segment_type.next_matches(&[SegmentType::Start]); cb.condition(is_final_segment, |cb| {
Figure C.9: mpt-circuit/src/gadgets/mpt_update.rs#1878-1880 • There is a redundant .and() call in the
MptUpdateConfig configure function. The cb.every_row_selector()
function returns the first condition in the condition stack, so this condition is equivalent to is_start .
cb.condition(is_start.clone().and(cb.every_row_selector()), |cb| { Figure C.10: mpt-
circuit/src/gadgets/mpt_update.rs#124 • The rw_counter constraints could be consolidated. Constraining
rw_counter requires constraining the tag to Memory or TxLog and constraining the Padding to 0
. These constraints are implemented in different locations in the codebase, making the code hard to understand.
let is_rw_type = meta.query_advice(is_memory, CURRENT) + is_tx_log.expr(); Figure C.11:
src/copy_circuit.rs#340-341 //Decrement rwc_inc_left for the next row, when an RW operation happens.
let rwc_diff = is_rw_type.expr()*is_word_end.expr(); Figure C.12: copy_circuit/copy_gadgets.rs#L525 •
The following constraint label is imprecise. The label should read assign_real_bytes_left { } .
//real_bytes_left region.assign_advice( || format!("assign_bytes_left{}", *offset),
self.copy_table.real_bytes_left, *offset, || Value::known(F::zero()), ); Figure C.13:
src/copy_circuit.rs#776-782 hashey 43 ScrollzkEVM Circuits Security Assessment PUBLIC

```

D. Automated Analysis Tool Configuration

As part of this assessment, we used the tools described below to perform automated testing of the codebase.

D.1. Semgrep We used the static analyzer Semgrep to search for risky API patterns and weaknesses in the source code repository. For this purpose, we wrote rules specifically targeting the ConstraintBuilder APIs and the ExecutionGadget trait. semgrep --metrics=off --sarif --config=custom_rule_path.yml
Figure D.1: The invocation command used to run Semgrep for each custom rule Duplicate Constraints
The presence of duplicate constraints, with potentially different labels, indicates either a redundant constraint that can be removed or an intended constraint that was not correctly updated. This pattern was written to find variants of finding TOB-SCROLL2-6, but no other instances of it were found in the codebase. However, this pattern should be added as a CI/CD Semgrep rule to prevent a similar issue from recurring in the codebase. rules: -id:repeated-constraints message:"Found redundant or incorrectly updated constraint" languages:[rust] severity:ERROR patterns: -pattern: | cb.\$FUNC(\$LABEL1,\$LEFT,\$RIGHT); ... cb.\$FUNC(\$LABEL2,\$LEFT,\$RIGHT); Figure D.2: The repeated-constraints Semgrep rule Constraints with Repeated Labels
The presence of a repeated label could indicate a copy-pasted label that should be updated. rules: -id:constraints-with-repeated-labels message:"Found constraints with the same label" languages:[rust] severity:ERROR patterns: -pattern: | hashey 44 ScrollzkEVM Circuits Security Assessment PUBLIC

cb.\$FUNC("\$LABEL", ...); ... cb.\$FUNC("\$LABEL", ...); Figure D.3: The repeated-labels Semgrep rule

D.2. cargo-llvm-cov cargo-llvm-cov generates Rust code coverage reports. We used the cargo-llvm-cov --open command in the MPT codebase to generate the coverage report presented in the Automated Testing section.

D.3. cargo-edit cargo-edit allows developers to quickly find outdated Rust crates. The tool can be installed with the cargo install cargo-edit command and the cargo upgrade --incompatible --dry-run command can be used to find outdated crates. D.4. Clippy The Rust linter Clippy can be installed using rustup by running the command rustup component add clippy . Invoking cargo clippy --workspace ---Wclippy::pedantic in the root directory of the project runs the tool with the pedantic ruleset. cargo clippy --workspace ---

E. Fix Review Results When undertaking a fix review, hashey 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 September 25 to September 29, 2023, hashey reviewed the fixes and mitigations implemented by the Scroll team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue. Scroll provided PRs with fixes for all high-severity findings and for the informational-severity finding TOB-SCROLL2-7. Scroll did not submit fixes for the remaining informational-severity findings. In summary, of the 10 issues described in this report, Scroll has resolved three issues and has partially resolved one issue. Scroll indicated that it does not intend to fix finding TOB-SCROLL2-2, so its status is unresolved. No fix PRs were provided for the remaining five issues, so their fix statuses are undetermined. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Poseidon Lookup is not implemented	Undetermined
2	IsZeroGadget does not constrain the inverse witness when the value is zero	Unresolved
3	The MPT nonexistence proof gadget is missing constraint specified in the documentation	Undetermined
4	Discrepancies between the MPT circuit specification and implementation	Undetermined
5	Redundant lookups in the WordRLC circuit	Undetermined
6	The Nonce Changed configuration circuit does not constrain the new value nonce value	Resolved

46 ScrollzkEVMCircuitsSecurityAssessment PUBLIC

7	The Copy circuit does not totally enforce the tag values	Partially Resolved
8	The "invalid creation" error handling circuit is unconstrained	Resolved
9	The OneHot primitive allows more than one value at once	Resolved
10	Intermediate columns are not explicit	Undetermined

hashey 47 ScrollzkEVMCircuitsSecurityAssessment PUBLIC

Detailed Fix Review Results TOB-SCROLL2-1: Poseidon Lookup is not implemented Undetermined. No fix was provided for this issue, so we do not know whether this issue has been addressed. TOB-SCROLL2-2: IsZeroGadget does not constrain the inverse witness when the value is zero Unresolved. The Scroll team has indicated that it does not intend to fix this issue. TOB-SCROLL2-3: The MPT nonexistence proof gadget is missing constraint specified in the documentation Undetermined. No fix was provided for this issue, so we do not know whether this issue has been addressed. TOB-SCROLL2-4: Discrepancies between the MPT circuit specification and implementation Undetermined. No fix was provided for this issue, so we do not know whether this issue has been addressed. TOB-SCROLL2-5: Redundant lookups in the WordRLC circuit Undetermined. No fix was provided for this issue, so we do not know whether this issue has been addressed. TOB-SCROLL2-6: The Nonce Changed configuration circuit does not constrain the new value nonce value Resolved in PR#73. The two conditional constraints that referred to `old_value` instead of `new_value` have been factored out into a single unconditional constraint referring to `new_value`, and the `Extension0` case has been removed entirely in favor of a blanket assertion forbidding that case of `configure_nonce`. TOB-SCROLL2-7: The Copy circuit does not totally enforce the tag values Partially resolved in PR#809. Several cases of the `CopyDataType` tag have been removed, and an assertion has been added to document that the `Padding` tag value is internal only. We did not fully evaluate whether this prevents tag values outside the expected range. TOB-SCROLL2-8: The "invalid creation" error handling circuit is unconstrained Resolved in PR#751. The `MemoryMask` gadget is used to extract the correct byte from the `word` at `offset` and to constrain it to equal `0xef`. TOB-SCROLL2-9: The OneHot primitive allows more than one value at once Resolved in PR#69. Each binary column value `v` is constrained by enforcing $1 - v \text{ or } (!v) = 0$. It is not immediately obvious that this constraint suffices, but it is equivalent to `by` the following reasoning: $(1 - 0) = 0$ hashey 48 ScrollzkEVMCircuitsSecurityAssessment PUBLIC

$1 - v \text{ or } (!v) = 1 - (!(v) \text{ and } (!!v)) = 1 - (1 - ((!v) * (!!v))) \dots = ((1 - v) * (1 - (1 - v))) = (1 - v) * v$

In addition, another related issue with the `OneHot` primitive, which was not discovered during the initial review engagement, was fixed in PR#68. The related problem was a typo causing `OneHot::previous` to return the result of the current row rather than the previous row. It exploits a scenario would be effectively the same as the one described in finding TOB-SCROLL2-9. The Scroll team fixed this by replacing the value `BinaryColumn::current` with `BinaryColumn::previous`.

TOB-SCROLL2-10: Intermediate columns are not explicit	Undetermined
---	--------------

No fix was provided for this issue, so we do not know whether this issue has been addressed. hashey 49 ScrollzkEVMCircuitsSecurityAssessment PUBLIC

F. 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.

