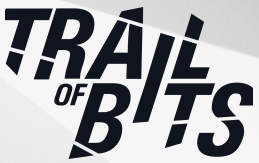

Scroll Euclid Upgrade Phase 2

Security Assessment

April 4, 2025

Prepared for:
Roy Lou
Scroll

Prepared by: **Filipe Casal, Tjaden Hess, and Anish Naik**

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Jeff Braswell, Project Manager
jeff.braswell@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:

Filipe Casal, Consultant
filipe.casal@trailofbits.com

Tjaden Hess, Consultant
tjaden.hess@trailofbits.com

Anish Naik, Consultant
anish.naik@trailofbits.com

Project Timeline

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

Date	Event
March 4, 2025	Pre-project kickoff call
March 14, 2025	Status update meeting #1
March 21, 2025	Delivery of report draft
March 24, 2025	Report readout meeting
April 4, 2025	Delivery of final comprehensive report

Executive Summary

Engagement Overview

Scroll engaged Trail of Bits to review the security of the changes introduced during phase 2 of the Euclid upgrade. These changes add enforced transactions and enforced batches, as well as restructuring block information from calldata to blobs, with associated changes to the zero-knowledge chunk and batch circuits.

A team of three consultants conducted the review from March 6 to March 21, 2025, for a total of four engineer-weeks of effort. Our testing efforts focused on assessing the soundness and correctness of the changes made to the chunk and batch circuits, the soundness of the enforced liveness mechanism, and whether the rollup contract correctly handles version 7 batches. With full access to source code and documentation, we performed static and dynamic testing of the codebase, using automated and manual processes.

Observations and Impact

We did not uncover any soundness or completeness issues in the changes made to the chunk, batch, and bundle circuits. However, we identified one potentially impactful but difficult-to-exploit supply-chain issue in the use of GitHub actions in the `zkvm-prover` repository, TOB-SCREUC2-4.

We did not uncover any issues in the smart contract logic that would adversely affect the confidentiality, integrity, or availability of the system. The updates have sufficient data validation, access controls, and testing to ensure liveness and support the processing of v7 batches.

Recommendations

Based on the findings identified during the security review, Trail of Bits 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 any refactor that may occur when addressing other recommendations.
- **Document updates to any system-level invariants.** The introduction of the enforced liveness mechanism and updates to the circuits lead to critical changes to the system's core invariants (see appendix D). Documenting these updates and ensuring that they are programmatically tested is essential to validating the system's new behavior.
- **Test the reversion of unfinalized v6 batches post-migration.** If the upgrade completes and a v6 batch must be reverted, the rollup contract must be

downgraded, the v6 batches must be reverted, and the rollup contract needs to be re-upgraded. This is currently not tested. Alternatively, finalize all v6 batches before migrating to v7 batches.

Finding Severities and Categories

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

EXPOSURE ANALYSIS

Severity	*Count*
High	0
Medium	0
Low	1
Informational	4
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Auditing and Logging	1
Configuration	1
Cryptography	1
Data Exposure	1
Data Validation	1

Project Goals

The engagement was scoped to provide a security assessment of the Scroll phase 2 Euclid upgrade. Specifically, we sought to answer the following non-exhaustive list of questions:

- Do the changes made to the chunk and batch circuits introduce any soundness or completeness issues?
- Is the KZG point opening verification functionality correctly implemented and used in the circuit?
- Does the new message queue implementation correctly update the rolling hash for incoming L1 messages?
- Does the rollup contract correctly handle v7 batches (encoding, commitment, finalization, and reversion)?
- Can an attacker bypass the enforced liveness mechanism or any access controls?
- Does the upgrade include any storage layer changes that could cause issues during the migration?
- Can the deprecated message queue (version 1) be used to cause undefined behavior?

Project Targets

The engagement involved a review and testing of the targets listed below.

zkvm-prover

Repository	https://github.com/scroll-tech/zkvm-prover/
Version	`c5781b608438b3dedccf468d696d99ecf27a110f`
Type	Rust, OpenVM

scroll-contracts

Repository	https://github.com/scroll-tech/scroll-contracts/
Version	`1a44cc4cd5bb29067051f59c09dd4407511f1a3a`
Type	Solidity

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:

- **Zkvm-prover.** We manually reviewed the chunk, batch, and bundle circuits, focusing on soundness and correctness issues. Specific areas of focus include the v7 payload decoding and validation, the KZG point opening verification routine, and its use in the context of EIP-4844.
- **Scroll-contracts.** We manually reviewed the new message queue (version 2), updates to the rollup contract, and any associated contract updates (e.g., the L1 cross-domain messenger). Specific areas of focus include soundness of the rolling message hash implementation, soundness of the enforced liveness mechanism, implications of EIP-7702 on address aliasing, and correctness of the encoding, commitment, finalization, and reversion of v7 batches.

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
Clippy	An open-source Rust linter used to catch common mistakes and unidiomatic Rust code	Appendix C
Dylint	A tool for running Rust lints from dynamic libraries	Appendix C
`cargo-audit`	An open-source tool for checking dependencies against the RustSec advisory database	Appendix C
`cargo-edit`	A tool for quickly identifying outdated crates	Appendix C
zizmor	A static analysis tool for GitHub Actions	Appendix C

Areas of Focus

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

- General code quality issues and unidiomatic code patterns
- Security of CI and GitHub Actions
- Identifying dependencies that are outdated or vulnerable to known attacks

Test Results

The results of this focused testing are detailed below.

cargo-audit

cargo-audit did not identify any dependency vulnerable to known security issues.

Clippy

Clippy reports warnings for three idiomatic and performance rules: manual assert, or_fun_call, and explicit_iter_loop, which we have included in our report for the Euclid

upgrade phase 1 engagement. We recommend adding these rules to your regular Clippy runs.

cargo-edit

Cargo-edit identified several outdated dependencies, including some that are, in a semantic versioning sense, compatible with the version currently in use, and others that are not. Consider integrating Dependabot to automatically update dependencies as soon as these are available.

```
$ cargo upgrade --incompatible --dry-run
    Checking virtual workspace's dependencies
name                    old req    compatible latest new req
====                    =======    ========== ====== =======
alloy-serde             0.8        0.8.3      0.12.5 0.12
bitcode                 0.6.3      0.6.5      0.6.5  0.6.5
bincode                 2.0.0-rc.3 2.0.1      2.0.1  2.0.1
halo2curves-axiom       0.5.3      0.5.3      0.7.0  0.7.0
metrics                 0.23.0     0.23.0     0.24.1 0.24.1
metrics-util            0.17       0.17.0     0.19.0 0.19
metrics-tracing-context 0.16.0     0.16.0     0.18.0 0.18.0
serde_with              3.11.0     3.12.0     3.12.0 3.12.0
toml                    0.8.14     0.8.20     0.8.20 0.8.20
    Checking scroll-zkvm-batch-circuit's dependencies
name  old req compatible latest new req
====  ======= ========== ====== =======
c-kzg 1.0     1.0.3      2.0.0  2.0
    Checking scroll-zkvm-build-guest's dependencies
name           old req compatible latest new req
====           ======= ========== ====== =======
cargo_metadata 0.19.1  0.19.2     0.19.2 0.19.2
    Checking scroll-zkvm-bundle-circuit's dependencies
    Checking scroll-zkvm-chunk-circuit's dependencies
    Checking scroll-zkvm-circuit-input-types's dependencies
    Checking scroll-zkvm-integration's dependencies
name      old req compatible latest new req
====      ======= ========== ====== =======
once_cell 1.20    1.21.1     1.21.1 1.21
    Checking scroll-zkvm-prover's dependencies
name        old req compatible latest new req
====        ======= ========== ====== =======
bincode     1.3     1.3.3      2.0.1  2.0
git-version 0.3.5   0.3.9      0.3.9  0.3.9
once_cell   1.20    1.21.1     1.21.1 1.21
revm        19.0    19.6.0     19.6.0 19.6
c-kzg       1.0     1.0.3      2.0.0  2.0
    Checking scroll-zkvm-verifier's dependencies
```

```
name    old req compatible latest new req
====    ======= ========== ====== =======
bincode 1.3     1.3.3      2.0.1  2.0
revm    19.0    19.6.0     19.6.0 19.6
```

Figure T.1: Results of running cargo-edit on the zkvm-prover codebase

Summary of Findings

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

ID	Title	Type	Severity
1	Polynomial evaluation does not consider roots of unity edge case	Data Validation	**Informational**
2	Transaction data length missing from the ChunkInfo PI hash	Cryptography	**Informational**
3	Risky use of GITHUB_ENV in GitHub action	Data Exposure	**Informational**
4	Unpinned external GitHub CI/CD action versions	Auditing and Logging	**Low**
5	Potential credential persistence in artifacts and stale GitHub action	Configuration	**Informational**

Detailed Findings

1. Polynomial evaluation does not consider roots of unity edge case

Severity: **Informational**	Difficulty: **N/A**
Type: Data Validation	Finding ID: TOB-SCREUC2-1

Target: `zkvm-prover/crates/circuits/batch-circuit/src/blob_consistency/openvm.rs`

Description

The polynomial evaluation does not check whether the evaluation point is one of the roots of unity. If this were to happen, the prover would panic when trying to invert zero. This is a highly unlikely scenario given that the evaluation point z is currently used only after deriving it using a hash function, but the EIP-4844 specification and the reference `c-kzg` implementation still perform this check.

```
for (i = 0; i < FIELD_ELEMENTS_PER_BLOB; i++) {
    /*
     * If the point to evaluate at is one of the evaluation points by which the
polynomial is
     * given, we can just return the result directly.  Note that special-casing this
is
     * necessary, as the formula below would divide by zero otherwise.
     */
    if (fr_equal(x, &brp_roots_of_unity[i])) {
        *out = poly[i];
        ret = C_KZG_OK;
        goto out;
    }
    blst_fr_sub(&inverses_in[i], x, &brp_roots_of_unity[i]);
}
```

Figure 1.1: Reference c-kzg implementation (`eip4844/eip4844.c#L207-L219`)

Figure 1.2 shows where, if passed a root of unity, the implementation would panic when trying to invert zero.

```
fn interpolate(z: &Scalar, coefficients: &[Scalar; BLOB_WIDTH]) -> Scalar {
    let blob_width = u64::try_from(BLOB_WIDTH).unwrap();
    (pow_bytes(z, &blob_width.to_be_bytes()) - <Scalar as IntMod>::ONE)
        * ROOTS_OF_UNITY
```

```
            .iter()
            .zip_eq(coefficients)
            .map(|(root, f)| f * root * (z.clone() - root).invert())
```

Figure 1.2: `circuits/batch-circuit/src/blob_consistency/openvm.rs#194–200`

Adding the check both guarantees that the implementation follows the specification and that if it is changed or reused in a context that allows an attacker to provide the evaluation point, it will not suffer from completeness issues.

Recommendations

Short term, add the check ensuring that the implementation does not throw a runtime error if the evaluation point is one of the roots of unity.

Long term, add tests exercising this particular code path.

2. Transaction data length missing from the ChunkInfo PI hash

Severity: **Informational**	Difficulty: **N/A**
Type: Cryptography	Finding ID: TOB-SCREUC2-2
Target: `zkvm-prover/crates/circuits/types/src/chunk/public_inputs.rs`	

Description

The public input hash function for the `ChunkInfo` structure does not include the `tx_data_length`. This means that two different `ChunkInfo` structures that differ only in the `tx_data_length` field would have the same public input hash.

```
pub struct ChunkInfo {
    /// The EIP-155 chain ID for all txs in the chunk.
    #[rkyv()]
    pub chain_id: u64,
    /// The state root before applying the chunk.
    #[rkyv()]
    pub prev_state_root: B256,
    /// The state root after applying the chunk.
    #[rkyv()]
    pub post_state_root: B256,
    /// The withdrawals root after applying the chunk.
    #[rkyv()]
    pub withdraw_root: B256,
    /// Digest of L2 tx data flattened over all L2 txs in the chunk.
    #[rkyv()]
    pub tx_data_digest: B256,
    /// The L1 msg queue hash at the end of the previous chunk.
    #[rkyv()]
    pub prev_msg_queue_hash: B256,
    /// The L1 msg queue hash at the end of the current chunk.
    #[rkyv()]
    pub post_msg_queue_hash: B256,
    /// The length of rlp encoded L2 tx bytes flattened over all L2 txs in the
chunk.
    #[rkyv()]
    pub tx_data_length: u64,
    /// The block number of the first block in the chunk.
    #[rkyv()]
    pub initial_block_number: u64,
    /// The block contexts of the blocks in the chunk.
    #[rkyv()]
    pub block_ctxs: Vec<BlockContextV2>,
}
```

Figure 2.1: `circuits/types/src/chunk/public_inputs.rs#90–121`

```rust
impl PublicInputs for ChunkInfo {
    /// Public input hash for a given chunk is defined as
    ///
    /// keccak(
    ///     chain id ||
    ///     prev state root ||
    ///     post state root ||
    ///     withdraw root ||
    ///     tx data digest ||
    ///     prev msg queue hash ||
    ///     post msg queue hash ||
    ///     initial block number ||
    ///     block_ctx for block_ctx in block_ctxs
    /// )
    fn pi_hash(&self) -> B256 {
        keccak256(
            std::iter::empty()
                .chain(&self.chain_id.to_be_bytes())
                .chain(self.prev_state_root.as_slice())
                .chain(self.post_state_root.as_slice())
                .chain(self.withdraw_root.as_slice())
                .chain(self.tx_data_digest.as_slice())
                .chain(self.prev_msg_queue_hash.as_slice())
                .chain(self.post_msg_queue_hash.as_slice())
                .chain(&self.initial_block_number.to_be_bytes())
                .chain(
                    self.block_ctxs
                        .iter()
                        .flat_map(|block_ctx| block_ctx.to_bytes())
                        .collect::<Vec<u8>>()
                        .as_slice(),
                )
                .cloned()
                .collect::<Vec<u8>>(),
        )
    }
```

Figure 2.2: circuits/types/src/chunk/public_inputs.rs#L144-L179

We could not identify a soundness issue resulting from the missing field in the hash calculation, given that the `tx_data_length` field is used to validate the `tx_data_digest` field in the `PayloadV7::validate` function.

Recommendations

Short term, add the missing `tx_data_length` field to the `keccak256` call of `ChunkInfo::pi_hash`.

Long term, consider using a derive macro to implement structured public input hashing.

3. Risky use of GITHUB_ENV in GitHub action

Severity: **Informational**	Difficulty: **N/A**
Type: Data Exposure	Finding ID: TOB-SCREUC2-3
Target: `zkvm-prover/.github/workflows/profile-guest.yml`	

Description

The `profile-guest` GitHub action uses the `GITHUB_ENV` variable to save the result of a command and use it in another step within the same job. However, due to the potential security consequences of using the `GITHUB_ENV` variable, we recommend using the `GITHUB_OUTPUT` variable instead.

Currently, the `profile-guest` action runs after the `build-guest` action has been completed and performs the following steps:

1. Checks out the default branch in the repository;
2. Downloads the cached artifact;
3. Runs `make profile-chunk` and gathers how many cycles were used, saving it to `$GITHUB_ENV`;
4. Updates the pull request with the saved information.

```
- name: Guest profiling to capture total_cycles
  id: guest-profiling
  run: |
    # Run the tests and capture the output
    output=$(make profile-chunk | grep "scroll-zkvm-integration(chunk-circuit):
total cycles = ")
    echo "total_cycles=$output" >> $GITHUB_ENV

- name: Update PR Description
  uses: actions/github-script@v6
  with:
    script: |
      const prNumber = context.payload.pull_request.number;
      const totalCycles = process.env.total_cycles;
      const currentBody = context.payload.pull_request.body;

      // Update the PR description with the total cycles
      const newBody = `### Total Cycles
(chunk-circuit)\n${totalCycles}\n\n${currentBody}`;
      await github.pulls.update({
```

```
        owner: context.repo.owner,
        repo: context.repo.repo,
        pull_number: prNumber,
        body: newBody,
      });
```

Figure 3.1: `.github/workflows/profile-guest.yml#L28-L50`

Currently, the action is not vulnerable because the branch that is checked out is the main branch and not the pull request branch, and because the value that is written to `$GITHUB_ENV` can only be a single `u64`. If, instead, the current action added the first line of `chunk-app-vmexe` to the `GITHUB_ENV`, an attacker could execute arbitrary code on the following step of the job by using the `LD_PRELOAD` environment variable.

Recommendations

Short term, prefer `$GITHUB_OUTPUT` to pass values between different job steps. Carefully review GitHub actions triggered on `workflow_run`.

Long term, add zizmor to the CI/CD pipeline of the codebase.

References

- Zizmor documentation on GITHUB_ENV use
- GitHub Actions exploitation: environment manipulation
- GHSL-2024-177: Environment Variable injection in an Actions workflow of Litestar

4. Unpinned external GitHub CI/CD action versions

Severity: **Low**	Difficulty: **High**
Type: Auditing and Logging	Finding ID: TOB-SCREUC2-4
Target: `zkvm-prover/.github/workflows/`	

Description

Several GitHub Actions workflows in the `zkvm-prover` repository use third-party actions pinned to a tag or branch name instead of a full commit SHA as recommended by GitHub. This configuration enables repository owners to silently modify the actions. A malicious actor could use this ability to tamper with an application release or leak secrets.

The following actions are owned by organizations or individuals that are not affiliated directly with Scroll:

- `nick-fields/assert-action@v1`
- `fkirc/skip-duplicate-actions@v5`
- `dtolnay/rust-toolchain@master`

```
- name: Sanity check for chunk-commitments (prover vs verifier)
  uses: nick-fields/assert-action@v1
  with:
    expected: ${{ steps.commitments-chunk-prover.outputs.commitments }}
    actual: ${{ steps.commitments-chunk-verifier.outputs.commitments }}

- name: Sanity check for chunk-commitments (prover vs circuits)
  uses: nick-fields/assert-action@v1
  with:
    expected: ${{ steps.commitments-chunk-prover.outputs.commitments }}
       actual: ${{ steps.commitments-chunk-circuits.outputs.commitments }}
```

Figure 4.1: `.github/workflows/build-guest.yml#112–122`

Exploit Scenario

An attacker gains unauthorized access to the account of a GitHub action owner. The attacker manipulates the action's code to steal GitHub credentials and compromise the repository.

Recommendations

Short term, pin each third-party action to a specific full-length commit SHA, as recommended by GitHub. Additionally, configure Dependabot to update the actions' commit SHAs after reviewing their available updates.

Long term, add `zizmor` to the CI/CD pipeline.

5. Potential credential persistence in artifacts

Severity: **Informational**	Difficulty: **N/A**
Type: Configuration	Finding ID: TOB-SCREUC2-5
Target: `.github/workflows/{profile-guest.yml, lint.yml, build_guest.yml}`	

Description

The default behavior of the `actions/checkout` GitHub action is to persist credentials, which means that the GitHub token is written to the local repository directory. This allows the action to execute Git commands that require authentication. The credentials are stored in `.git/config` files in the local repository directory, where they could be inadvertently included in workflow artifacts or accessed by subsequent workflow steps.

```
jobs:
  profile-guest:
    if: ${{ github.event.workflow_run.conclusion == 'success' }}
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Download artifact (chunk app.vmexe)
        uses: actions/download-artifact@v4
        with:
          name: chunk-app-vmexe
          path: ./crates/circuits/chunk-circuit/openvm/
```

Figure 5.1: `.github/workflows/profile-guest.yml#9–21`

Currently, the actions with the credential persistence do not appear to save them into artifacts that are publicly available, but we recommend adding this flag to prevent potential issues in case the actions are updated.

Recommendations

Short term, add `persist-credentials: false` to checkout actions that do not require privileged Git operations.

Long term, add `zizmor` to the CI/CD pipeline.

References

- ArtiPACKED: Hacking Giants Through a Race Condition in GitHub Actions Artifacts

- `zizmor` ArtiPACKED documentation

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 Quality Findings

We identified the following code quality issues through manual and automatic code review. These issues are not associated with any specific vulnerabilities. However, they will enhance code readability and may prevent the introduction of vulnerabilities in the future.

- **Blob byte format is not documented in the Rust code.** Although the `Blob` data format can be inferred from the implementation, we encourage adding documentation describing the format.

```
impl From<&[u8]> for EnvelopeV7 {
       fn from(blob_bytes: &[u8]) -> Self {
```

Figure B.1: `circuits/types/src/batch/payload/v7.rs#32–33`

- **Rust codebase includes duplicate and outdated files.** The Rust codebase has the `circuits/batch-circuit/src/payload` folder, which includes duplicate and outdated files included in `circuits/types/src/batch/payload`. The `v7.rs` file included within is outdated and uses the wrong constant for parsing block contexts.

```
// Deserialize block contexts depending on the number of blocks in the batch.
let mut block_contexts = Vec::with_capacity(num_blocks as usize);
for i in 0..num_blocks {
    let start = (i as usize) * SIZE_BLOCK_CTX + INDEX_NUM_BLOCKS;
    block_contexts.push(BlockContextV2::from(
        &payload_bytes[start..(start + SIZE_BLOCK_CTX)],
    ));
}
```

Figure B.2: `circuits/batch-circuit/src/payload/v7.rs#161–168`

```
// Deserialize block contexts depending on the number of blocks in the batch.
let mut block_contexts = Vec::with_capacity(num_blocks as usize);
for i in 0..num_blocks {
    let start = (i as usize) * SIZE_BLOCK_CTX + INDEX_BLOCK_CTX;
    block_contexts.push(BlockContextV2::from(
        &payload_bytes[start..(start + SIZE_BLOCK_CTX)],
    ));
}
```

Figure B.3: `crates/circuits/types/src/batch/payload/v7.rs#161–168`

- **The previous state root is redundantly fetched twice.** The `pre_state_root` and `prev_state_root` variables are assigned the same value, and the implementation could use just one of them.

```
let pre_state_root = witness.blocks[0].pre_state_root;
```

```
let chain = Chain::from_id(witness.blocks[0].chain_id());

// enable all forks
let hardforks = (*SCROLL_DEV_HARDFORKS).clone();

let inner = ChainSpec {
    chain,
    genesis_hash: Default::default(),
    genesis: Default::default(),
    genesis_header: Default::default(),
    paris_block_and_final_difficulty: Default::default(),
    hardforks,
    deposit_contract: Default::default(),
    base_fee_params: BaseFeeParamsKind::Constant(BaseFeeParams::ethereum()),
    prune_delete_limit: 20000,
    blob_params: Default::default(),
};
let config = ScrollChainConfig::mainnet();
let chain_spec = ScrollChainSpec { inner, config };

let (code_db, nodes_provider, block_hashes) =
make_providers(&witness.blocks);
let nodes_provider = manually_drop_on_zkvm!(nodes_provider);

let prev_state_root = witness.blocks[0].pre_state_root();
```

Figure B.4: `circuits/types/src/chunk/execute.rs#43–68`

- **Unresolved TODO comment in the code base.** The code comment mentions that the binary patching function should be removed once OpenVM reaches v1.0, which has already happened. Consider tracking such items on a centralized issue tracker like GitHub issues.

```
/// TODO: remove this after openvm v1.0
fn binary_patch(elf_bin: &[u8]) -> Vec<u8> {
```

Figure B.5: `build-guest/src/builder/mod.rs#49–50`

- **Multiple conversions when moving from the `halo2curves_axiom::bls12_381` type to the `openvm_pairing_guest::bls12_381` type.** The implementation first uses the `Bls12_381_G1::from_compressed_be` to deserialize the witness bytes, then converts this with `G1Affine::from_xy_unchecked`, and finally uses `G1Affine::from_xy_nonidentity` to check that the points are not the infinity point. Instead, consider doing the infinity check after the initial deserialization with the `Bls12_381_G1::is_identity` function; this will ensure that only one conversion is needed to convert to the OpenVM type.

```
// Verify KZG proof.
let proof_ok = {
    let commitment = Bls12_381_G1::from_compressed_be(kzg_commitment)
```

```
        .expect("kzg commitment")
        .convert();
    let proof = Bls12_381_G1::from_compressed_be(kzg_proof)
        .expect("kzg proof")
        .convert();
    verify_kzg_proof(challenge, evaluation, commitment, proof)
};
```

Figure B.6: circuits/batch-circuit/src/builder/v7.rs#51–60

```
impl EccToPairing for Bls12_381_G1 {
    type PairingType = G1Affine;

    fn convert(&self) -> Self::PairingType {
        G1Affine::from_xy_unchecked(self.x.convert(), self.y.convert())
    }
}

impl EccToPairing for Bls12_381_G2 {
    type PairingType = G2Affine;

    fn convert(&self) -> Self::PairingType {
        G2Affine::from_xy_unchecked(
            Fp2::new(self.x.c0.convert(), self.x.c1.convert()),
            Fp2::new(self.y.c0.convert(), self.y.c1.convert()),
        )
    }
}
```

Figure B.7: circuits/batch-circuit/src/blob_consistency/openvm.rs#99–116

```
/// Verify KZG `proof` that `P(z) == y` where `P` is the EIP-4844 blob
polynomial in its evaluation
/// form, and `commitment` is the KZG commitment to the polynomial `P`.
///
/// We use [`openvm_pairing_guest`] extension to implement this in guest
program.
pub fn verify_kzg_proof(z: Scalar, y: Scalar, commitment: G1Affine, proof:
G1Affine) -> bool {
    let proof_q = G1Affine::from_xy_nonidentity(proof.x().clone(),
proof.y().clone())
        .expect("kzg proof not G1 identity");
    let p_minus_y = G1Affine::from_xy_nonidentity(commitment.x().clone(),
commitment.y().clone())
        .expect("kzg commitment not G1 identity")
```

Figure B.8: circuits/batch-circuit/src/blob_consistency/openvm.rs#118–126

- **Fix the following inline comments.** In figure B.9, the highlighted snippet should be L1MessageQueueV2. In figure B.10, the highlighted snippet should be L1MessageQueueV1.

```
764     // Pop finalized and non-skipped message from L1MessageQueue.
```

Figure B.9: src/L1/rollup/ScrollChain.sol#L764

```
51     /// @dev The storage slot used as `L1MessageQueueV2` contract, which is
deprecated now.
```

Figure B.10: src/L1/gateways/EnforcedTxGateway.sol#L51

C. Automated Analysis Tool Configuration

We used the following tools to perform automated testing of the codebase.

C.1. Clippy

The Rust linter Clippy can be installed using `rustup` by running the command `rustup component add clippy`. Invoking `cargo clippy -- -W clippy::pedantic` in the root directory of the project runs the tool with the pedantic ruleset.

Clippy results from the regular set of rules should always be fixed.

```
# run clippy in regular and pedantic mode and output to SARIF
cargo clippy --message-format=json | clippy-sarif > clippy.sarif
cargo clippy --message-format=json -- -W clippy::pedantic -Wclippy::or_fun_call |
clippy-sarif > clippy_pedantic.sarif
```

Figure C.1: The invocation commands used to run Clippy in the codebase

Converting the output to the SARIF file format (e.g., with `clippy-sarif`) allows easy inspection of the results within an IDE (e.g., using VSCode's SARIF Explorer extension).

C.2. Dylint

Dylint is a tool for running Rust lints from dynamic libraries, similar to Clippy. We ran the general and supplementary rulesets against the codebase, finding no remarkable issues.

```
cargo dylint --no-deps --git https://github.com/trailofbits/dylint --pattern
examples/general -- --message-format=json | clippy-sarif > general.sarif

cargo dylint --no-deps --git https://github.com/trailofbits/dylint --pattern
examples/supplementary -- --message-format=json | clippy-sarif > supplementary.sarif
```

Figure C.2: The command used to run Dylint on the project

C.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.

C.4. cargo-audit

The `cargo-audit` Cargo plugin identifies known vulnerable dependencies in Rust projects. It can be installed using `cargo install cargo-audit`. To run the tool, run `cargo audit` in the crate root directory.

C.5. zizmor

Zizmor is a static analysis tool for GitHub Actions. It can be installed using `cargo install --locked zizmor`. We ran zizmor with the following commands:

```
zizmor . --format sarif > zizmor.sarif
zizmor --persona pedantic . --format sarif > zizmor_pedantic.sarif
```

Figure C.3: Commands used to run zizmor and output to SARIF

D. Smart Contract Invariants

Below is a non-exhaustive list of invariants that we identified during the course of the audit. This list specifically pertains to the changes made to the smart contracts for the new EuclidV2 upgrade.

- `firstCrossDomainMessageIndex <= nextUnfinalizedQueueIndex <= nextCrossDomainMessageIndex`
    - `firstCrossDomainMessageIndex` must not change once it is set.
- The `L1MessageQueueV1` contract should no longer be callable.
- A user cannot directly append a message to the `L1MessageQueueV2` queue (must go through the L1 scroll messenger or the enforced transaction gateway).
- The enforced liveness mechanism is activated if and only if:
    - An L1 message has not been finalized for more than `maxDelayMessageQueue` seconds.
    - A batch has not been finalized for more than `maxDelayEnterEnforcedMode` seconds.
- A v7 batch header must be 73 bytes long.
- All messages in the `L1MessageQueueV1` queue must be finalized before finalizing any messages in the `L1MessageQueueV2` queue.
- All batches that are not v7 must be finalized before finalizing any v7 batches.
- Messages in the `L1MessageQueueV2` queue cannot be skipped or dropped.
- The system configuration can be updated only by the owner.
- The rolling hash mechanism must be deterministic.
- All v6 batches must be finalized before the first v7 batch is finalized.
- Commitments of v6 and below batches must revert.
- No actor can directly enable the enforced liveness mechanism.
- The owner can directly disable the enforced liveness mechanism.
- No actor cannot directly update the `V1_MESSAGES_FINALIZED_OFFSET` flag.

E. 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.

On March 26, 2025, Trail of Bits 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.

In summary, of the five issues described in this report, Scroll has resolved three issues and has not resolved the remaining two issues. The unresolved issues are of informational severity and pose no security risk to the codebase. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Severity	Status
1	Polynomial evaluation does not consider roots of unity edge case	Informational	Unresolved
2	Transaction data length missing from the ChunkInfo PI hash	Informational	Unresolved
3	Risky use of GITHUB_ENV in GitHub action	Informational	Resolved
4	Unpinned external GitHub CI/CD action versions	Low	Resolved
5	Potential credential persistence in artifacts	Informational	Resolved

Detailed Fix Review Results

TOB-SCREUC2-1: Polynomial evaluation does not consider roots of unity edge case
Unresolved. The Scroll team provided the following context for this finding's fix status:

> *The polynomial evaluation function* should *in fact handle the case where the challenge could be a root of unity. Given how we have included this function in a standalone module, it comes across as available for a generic use-case. We are missing documentation/dev notes that we have skipped this explicit handling of root of unity case since in our case, the challenge is a keccak digest and the assumption is that the chances of that being a root of unity are negligible. It's worth adding such a note so that it does not get used for a generic purpose in the future.*

TOB-SCREUC2-2: Transaction data length missing from the ChunkInfo PI hash
Unresolved. The Scroll team provided the following context for this finding's fix status:

> *The tx_data_length is only required as a witness in the batch-circuit. We only need it to compute the tx_data_digest, i.e. the keccak digest of all L2 tx bytes chunk-by-chunk from the blob's payload. It essentially helps us in identifying chunk boundaries in the batch's data. It is essentially not required to be a part of the public-input from the chunk-circuit. However, given the way we setup the abstraction in the Circuit and AggCircuit traits, we thought its best to simply include the value tx_data_length in ChunkInfo and pass it on to the next layer (BatchCircuit). Since it's only an additional witness, we preferred passing it through this type instead of externally passing it through the BatchProvingTask or BatchWitness.*

TOB-SCREUC2-3: Risky use of GITHUB_ENV in GitHub action
Resolved in commits f616e and 7b1f7 by using the `GITHUB_OUTPUT` variable and sanitizing the template interpolation.

TOB-SCREUC2-4: Unpinned external GitHub CI/CD action versions
Resolved in commit 076b1 by pinning external actions to commit hashes.

TOB-SCREUC2-5: Potential credential persistence in artifacts
Resolved in commit 7d18c by disabling the default credential persistence option.

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.
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.

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 and 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 X 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

Trail of Bits performed all activities associated with this project in accordance with a statement of work and an agreed-upon project plan.

Security assessment projects are time-boxed and often rely on information 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 software controls and security properties. 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. A project's time and resource constraints also limit their use.