Response to Security Audit: Pessimistic Proof Product

This document outlines the actions that will be taken in response to the security assessment conducted by Trail of Bits on the Pessimistic Proof crate.

Summary of Findings and Response

1. Exit tree hash does not commit to the tree size

https://github.com/agglayer/agglayer/issues/248
Severity: Medium
Difficulty: High
Type: Cryptography
Target: crates/pessimistic-proof/src/local_exit_tree/mod.rs,
crates/pessimistic-proof/src/local_exit_tree/mod.rs

Response
This was fixed in #463. Instead of committing to the LET size directly in the LER, which would cause a breaking change with previously computed LERs, we decided to include it in the pessimistic root. This is sufficient as the pessimistic root is unpacked in the PP and it's enforced that the new bridge exits are inserted at the correct location in the LET.

2. MultiBatchHeader signature does not include sending network or leaf index

https://github.com/agglayer/agglayer/issues/233
Severity: Medium
Difficulty: Medium
Type: Data Validation
Target: crates/pessimistic-proof/src/{local_state.rs,imported_bridge_exit.rs,bridge_exit.rs

Response
I think this is fixed given that the signer of the Certificate signs a commitment which commits also to the set of GlobalIndex that it wants to nullify in the nullifier tree (key: GlobalIndex, value: bool)

GlobalIndex (crates/pessimistic-proof/src/global_index.rs) uniquely identifies each possible imported bridge exits by committing to its sending network and leaf index within the local exit tree of the sending network

3. apply_batch_header mutates local state on error return

https://github.com/agglayer/agglayer/issues/239
Severity: Informational
Difficulty: Not Applicable
Type: Error Reporting
Target: crates/pessimistic-proof/src/local_state.rs

Response
Fixed in #247 by using the roll-back version.

4. get_inclusion_proof_zero overwrites nonzero entries

https://github.com/agglayer/agglayer/issues/238
Severity: Informational
Difficulty: Not Applicable
Type: Testing
Target: crates/pessimistic-proof/src/util/smt.rs

Response
Fixed in #245 by using `insert` instead of `update`.

5. SMT leaves and branches are not domain-separated

https://github.com/agglayer/agglayer/issues/235
Severity: Informational
Difficulty: Not Applicable
Type: Cryptography
Target: crates/pessimistic-proof/src/util/smt.rs

Response
Since the current SMT design is not susceptible to this issue, I don't think it's necessary to add domain separation for now.

6. keccak256_combine is not a safe generic hashing function

https://github.com/agglayer/agglayer/issues/234
Severity: Informational
Difficulty: Not Applicable
Type: Cryptography
Target: crates/pessimistic-proof/src/keccak.rs

Response
Added a comment regarding the safety of `keccak256_combine` in #256.

I don't think we'll ever need to hash variable length data in this crate, so I think it's fine to keep `keccak256_combine` as is and just make sure it's always used safely.

Addressing Recommendations

1. Testing and Validation

We recommend expanding both unit and integration testing of the pessimistic proofs implementation, especially focused on negative tests. The Polygon team should include tests that create transactions that trigger every possible rejection. For example, a test that tries to import bridge exits to the wrong destination network would have discovered the high-severity vulnerability immediately. Similarly, the low-severity issues found would be caught by including tests that attempted to tamper with the signature or the Merkle tree size.

Testing on the agglayer and pessimistic-proof has been done following the happy path scenario in order to build something fast, in the next iterations we'll add more tests, both in terms of units but also integrations and e2e.

2. Documentation Enhancements

We also recommend that the Polygon team create more extensive documentation for their data structures, to ensure that they do not get reused without correct precautions. For example, the sparse Merkle tree implementation in this crate would enable an attacker to create fraudulent lookup proofs that would be accepted by an implementation that verifies proofs with an incorrect total tree height, or that accepts a variable-length proof.

Same as the section for Testing. This will be addressed in the next iterations.