HashCloak

Code Review and Security Assessment For *Light Protocol*

Initial Delivery: December 20, 2024

Updated Report: March 7, 2025

Prepared For:

Jorrit Palfner	| *GLAMOUR PROPHECY LDA*
Swen Schäferjohann	| *GLAMOUR PROPHECY LDA*

Prepared by:

Alex Yu-Chen Liao | *HashCloak Inc.*

Manish Kumar | *HashCloak Inc.*

Table Of Contents

Executive Summary **2**

Scope 3

Overview **3**

Methodology **3**

Overview of Evaluated Components **4**

Findings **7**

- LPT-1: No checks on batch size, `zkp_batch size`, and bloom filter capacity while creating a new batch 7
- LPT-2: Constraints on the size of the root_history relative to the queue size 7
- LPT-3: The function `read_ptr_at` and `read_array_like_ptr_at` return a mutable pointer from an immutable byte slice 8
- LPT-4: Function i`s_post_epoch` can return `Ok(())` even if the epoch is still in `ReportWork` state for the slot 9
- LPT-5: Consider moving `finalize_insert()` inside `add_to_hash_chain()` since `finalize_insert()` must be called after `add_to_hash_chain()` 9
- LPT-6: The function `get_num_zkp_batches` does not validate exact division of self.batch_size / self.zkp_batch_size 10
- LPT-7: Use of `unwrap()` for the `Option<> type` 11
- LPT-8: Unsafe assumptions in the bounded-vec implementation 11
- LPT-9: Redundant code blocks 12
- LPT-10: The function `calculate_optimal_hash_functions` is a dead code 13
- LPT-11: The instruction containing `noop_program`, `account_compression_authority`, and `account_compression_program` are unchecked accounts 13
- LPT-12: Comment for `hashv_to_bn254_field_size_be` doesn't match its functionality, and is duplicated with `hash_to_bn254_field_size_be` 14
- LPT-13: In the function `add_token_pool`, `token_pool_index.saturating_sub(1)` will be same for index 0 and 1 15

References **15**

Appendix **16**

Executive Summary

Light Protocol engaged *HashCloak Inc*. to perform code review and security assessment for Light Protocol, a ZK Compression Protocol for Solana.

During the audit period, we began familiarizing ourselves with the overall concepts of ZK compression, its usage in Solana, and its core components relevant to the audit scope by reviewing the code and the provided documentation. We also assessed some security vulnerabilities commonly found in Solana projects. Subsequently, we examined the code for errors, primarily focusing on logical correctness and identifying potential issues that could lead to double spending, loss of funds, or other consensus-related issues.

Overall, we found the source code is of good quality in representing the entities in the specification. However, we observed that some parts of the documentation are outdated, making certain sections of the code within the audit scope more challenging to assess. Additionally, we identified sections where the ideas, intuitions, and rationale are difficult to follow due to their conceptual complexity. To improve understandability, we recommend enriching the code with inline comments and updating the documentation to align with the current implementation. We also noted the presence of some TODOs that are yet to be added, as well as some other components such as `relay_fees` remain unimplemented.

Overall, we found the issues range from Medium to informational:

Severity	**Number of Findings**
Critical	0
High	0
Medium	4
Low	4
Informational	5

Scope

For the audit, we considered the repository https://github.com/Lightprotocol/light-protocol at commit d1cb64ce81e2341fc4f9d8a10e9bbbcb2d517333, 9269d6c328b499809ca60391abf16d04fcdb2801, 4f822f17e507d9f289432f161e535eaa9b8cd2b2, and 472feca74b1674fbfc778905611cfd184a84d974.

Overview

Light Protocol is a ZK Compression Protocol for Solana that enables secure scaling directly on Layer 1. It provides a hyperscale state, allowing developers and users to compress their on-chain state, reducing state costs by orders of magnitude while preserving the security, performance, and composability of the Solana L1. Furthermore, In addition to classic on-chain compute, compressed state via Light natively supports custom ZK compute, enabling developers to build previously unattainable computation designs on Solana.

Methodology

The audit was conducted through a combination of manual verification, writing test cases for specific checks, and group audit sessions. The primary focus was to map the available specifications to the actual implementation. We also assessed the code for sufficient code coverage and various types of vulnerabilities, including:

- Functions' logic matches the intended idea
- Look for any dead code
- Any attack that can impact user funds
- Any undefined behavior
- Usage of dependencies
- Data leaking and information integrity
- Any transaction replay or double spending possibility
- Check for transaction malleability
- Issues related to signature validation
- Validation and access control of the authority

- account checks
- tree and queue association
- Batched Merkle tree operations including append, nullify, initialize, and rollover

Overview of Evaluated Components

Bloom filter

- Correctness of the algorithm for calculating bloom filter size and number of hashes
- Reasonable parameters are used when creating a bloom filter
- The hashing algorithm used
- The implementation provides the correct functionalities of a bloom filter
- The possibility to insert an element multiple times

Account Compression

- **Instructions**
  - Correct logic is implemented append and nullify leaves
  - Different versions of append functions are called with respect to the account's own version
  - Validation of the signer's authority
  - The default parameters are reasonably set and validated
  - Correct function of bundle processing is called based on the queue's type and version
  - Rollover fee is transferred from and to the right party
  - Tree and queue association while batch updating and rollover of state
  - Only the desired signer(registered or authority) can append, nullify, initialize, or update the batched tree(state and account)
  - Correct logic for batch append, nullify, update, initialization, and rollover tree.
- **Processor**
  - Parameters are correctly set for initialization
  - Rollover fee is computed correctly for different types of account
- **State**
  - Correct rotation and control of the batch state
  - Elements cannot be double-spent by inserting them into the input queue

    - Access control of the Merkle tree
    - The root in the root history array must be cleared after a batch has been cleared
    - Non-inclusion check is correctly performed with the bloom filter
    - Default parameters are reasonably set
    - Parameters for batches, queues, and Merkle trees are validated upon initialization
    - Fees are correctly calculated and charged
    - The correctness of calculating account size
    - Any potential issues that can lead to the failure of transaction execution
    - Correctness of the deserialization for raw account data

Registry

- Verification of forester
- Overall correctness of the accounts structs' field
- Validation of the authority
- Conditions on creating an Epoch account and Forester Epoch Account are validated correctly
- State of an epoch is correctly set and rotated
- The protocol config is correctly checked
- Verification of wrapper instructions for batched Merkle tree

System

- Overflow/underflow and sum-check logic to ensure no new lamports being created
- Improper Account Validation
- Signer check
- Ownership validation
- Bump Seed Canonicalization
- Nullifier logic to avoid double spends
- Address collision while creating new addresses
- Compression/Decompression Logic
- Cross Program Invocations
- Lack of PDA validation

Bounded-vec

- alignment for the type T (responsibility of caller functions)

- Bound check for undefined behavior or buffer overflow (responsibility of caller functions)
- availability of the data at the pointer (ptr::read assumes the memory at the pointer is properly initialized)
- make sure that those functions don't use heap-backed structures (e.g., Vec, slice)
- double initialization or double drop of pointers

light-account-checks

- Account mutability
- Ownership
- Account Size
- Rent Exemption

light-compressed-account

- Zero-copy deserialization
- Hashing scheme for compressed account
- Address derivation

light-verifier

- Correct logic for batch verification for different batch size

Findings

LPT-1: No checks on batch size, `zkp_batch` size, and bloom filter capacity while creating a new batch

Type: Medium

Files affected:

- program/account_compression/src/state/batch.rs
- program-libs/batched-merkle-tree/src/batch.rs in 472feca74b1674fbfc778905611cfd184a84d974

Description: The function `Batch::new()` takes `num_iters`, `bloom_filter_capacity`, `batch_size,` and `zkp_batch_size` as arguments, but it lacks validation checks for these inputs. Without default sizes of `bloom_filter_capacity`, `batch_size`, and `zkp_batch_size` maintained throughout the codebase, it is encouraged to check the validation of those inputs to avoid significant false positive in the case of bloom filter and any other state inconsistency because of batch size and zkp batch size.

Impact: Significant false positives can cause the system to incorrectly identify an element as part of the set when it is not. If the `batch_size` and `zkp_batch_size` are not correctly defined, the proof verification may fail for the last zkp batch.

Recommendation: Add default parameters and check the parameters when creating structs.

Status: Pending

LPT-2: Constraints on the size of the `root_history` relative to the queue size

Type: Medium

Files affected:
- program/account_compression/src/state/batched_merkle_tree.rs
- program-libs/batched-merkle-tree/src/merkle_tree.rs in 472feca74b1674fbfc778905611cfd184a84d974

Description: The `root_history` is a `CyclicBoundedVec` that will overwrite existing data once full. If the `root_history` size is too small in comparison to the input or output queue, some old roots in the `root_history` might be overwritten while still needed by an element inside the input or output queue for its ZK proof.

Impact: The proof for the input or output queue may fail.

Recommendation: Add constraints for the size of `root_history`.

Status: Pending

LPT-3: The function `read_ptr_at` and `read_array_like_ptr_at` return a mutable pointer from an immutable byte slice

Type: Medium

Files affected:
- merkle-tree/bounded-vec/src/offset/zero_copy.rs

Description: The mentioned functions accept an immutable bytes slice (`&[u8]`) as parameter, and will return a mutable pointer (`*mut T`) from it that can break the mutability assumption.

Impact: Callers may falsely assume the bytes slice to be immutable based on the immutable parameter, which can lead to unexpected modifications to the data for which the pointer points are.

Recommendation: Change the parameter type from `&[u8]` to `&mut [u8]`.

Status: The issue has been resolved in commit [39bdaf89eb13d10aa54c310b7723bd247745d091]().

LPT-4: Function `is_post_epoch` can return `Ok(())` even if the epoch is still in `ReportWork` state for the slot

Type: Medium

Files affected:

- programs/registry/src/protocol_config/state.rs

Description: The function `is_post_epoch(&self, slot: u64, epoch: u64)` determines whether the epoch is in Post state by checking `self.get_current_active_epoch(slot)? <= epoch`. However, even if the next epoch is active, the previous epoch can still be in `ReportWork` state and has not yet reached the `Post` state.

Impact: Callers may mistakenly assume the epoch has reached `Post` state while it is still in the `ReportWork` state.

Recommendation: Add an additional check to confirm whether the input epoch is still in the `ReportWork` state when the active epoch is the subsequent epoch of the input epoch.

Status: Pending

LPT-5: Consider moving `finalize_insert()` inside `add_to_hash_chain()` since `finalize_insert()` must be called after `add_to_hash_chain()`

Type: Low

Files affected:
- program/account_compression/src/state/batch.rs

Description: If the function `finalize_insert()` is not called after `add_to_hash_chain()`, the value of `self.num_inserted` will not increase and may cause errors in `finalize_insert()`.

Impact: Failing to call `finalize_insert()` results in `self.num_inserted` always being zero.

Recommendation: Incorporate the logic of `finalize_insert()` directly into `add_to_hash_chain()`.

Status: The issue has been resolved at commit 9269d6c328b499809ca60391abf16d04fcdb2801.

LPT-6: The function `get_num_zkp_batches` does not validate exact division of `self.batch_size / self.zkp_batch_size`

Type: Low

Files affected:
- program/account_compression/src/state/batch.rs
- program-libs/batched-merkle-tree/src/batch.rs in 472feca74b1674fbfc778905611cfd184a84d974

Description: If there are no default values of the batch sizes and zkp batch sizes, then it is encouraged to check that the division of the `batch_size` by `zkp_batch_size` is the exact ie, `self.batch_size % self.zkp_batch_size == 0`.

Impact: If the division is not exact, the ZK proof for the last `zkp_batch` in a batch may fail.

Recommendation: Add checks for the remainder of the division.

Status: Pending

LPT-7: Use of `unwrap()` for the `Option<>` type

Type: Low

Files affected:

- programs/registry/src/protocol_config/state.rs

Description: The function `get_current_active_epoch_progress()` uses `unwrap()` for the `checked_sub()` function.

Impact: Undesired panic may occur when using the function.

Recommendation: Consider using `ok_or(RegistryError::ArithmeticUnderflow)?` for consistency with similar functions in the file.

Status: Pending

LPT-8: Unsafe assumptions in the bounded-vec implementation

Type: Informational

Files affected:

- merkle-tree/bounded-vec/src/libs.rs

Description: The bounded-vec implementation relies on unsafe assumptions, including the alignment of the memory, and the correctness of the buffer, ensuring that the length does not exceed the capacity(`length <= capacity`) and allocations must not exceed isize::MAX bytes. It delegates the responsibility of these checks to the caller, the checks are missing and are assumed to be taken care of by the caller.

Impact: If the caller fails to verify unsafe assumptions like alignment or capacity before calling the unsafe functions, some memory safety vulnerabilities may occur.

Recommendation: If possible, all checks such as alignment of type, correctness of buffer, and capacity checks should be performed in the bounded-vec functions.

Status: Pending

LPT-9: Redundant code blocks

Type: Informational

Files affected:

- program/account_compression/src/state/batch.rs
- program/account_compression/src/instructions/nullify_leaves.rs

Description:

```
if self.num_inserted == self.zkp_batch_size || self.num_inserted == 0
{
   self.num_inserted = 0;
}
```

In the code snippet above, the second check is redundant in the function `store_value` within the `batch.rs` file and should be removed.

Similarly, in the `process_nullify_leaves` function in the `nullify_leaves.rs file`, the following code block is also redundant.

```
if proofs.len() > 1 && proofs[0].len() != proofs[1].len() {
        msg!(
            "Proofs length mismatch {} {}",
            proofs[0].len(),
            proofs[1].len()
        );
        return
Err(AccountCompressionErrorCode::ProofLengthMismatch.into());
    }
```

Recommendation: Remove the redundant check.

Status: The first issue is fixed in 472feca74b1674fbfc778905611cfd184a84d974.

LPT-10: The function `calculate_optimal_hash_functions` is a dead code

Type: Informational

Files affected:
- merkle-tree/bloom-filter/src/lib.rs
- program-libs/bloom-filter/src/lib.rs in 472feca74b1674fbfc778905611cfd184a84d974

Description: The indicated function is not called anywhere in the codebase and lacks associated tests. If there is no planned use for this function in the future, it should be removed.

Recommendation: Remove the function if it is not used.

Status: Pending

LPT-11: The instruction containing `noop_program`, `account_compression_authority`, and `account_compression_program` are unchecked accounts

Type: Informational

Files affected:

- programs/system/src/invoke/instruction.rs
- programs/system/src/invoke_cpi/instruction.rs

Description: In both `InvokeCpiInstruction` and `InvokeInstruction`, the `noop_program`, `account_compression_authority,` and `invoking_program` are unchecked accounts. While manual checks are implemented in the codebase, it is considered the best practice to explicitly define these accounts to enhance safety and reduce the risk of runtime issues.

Impact: The program loses the ability to enforce certain safety and correctness checks during runtime.

Recommendation: Consider using specific Account Types instead of unchecked accounts.

Status: Pending

LPT-12: Comment for `hashv_to_bn254_field_size_be` doesn't match its functionality, and is duplicated with `hash_to_bn254_field_size_be`

Type: Informational

Files affected:

- program-libs/compressed-account/src/lib.rs

Description: `hashv_to_bn254_field_size_be` does not use bump seeds despite being mentioned in the function comment. The function also has duplicated functionality with `hash_to_bn254_field_size_be`.

Impact: The program loses the ability to enforce certain safety and correctness checks during runtime.

Recommendation: Update the comment and combine the two functions.

Status: Pending

LPT-13: In the function `add_token_pool`, `token_pool_index.saturating_sub(1)` will be same for index 0 and 1

Type: Low

Files affected:

- programs/compressed-token/src/lib.rs

Description: In function `add_token_pool`,

```
check_spl_token_pool_derivation_with_index(
            &ctx.accounts.mint.key().to_bytes(),
            &ctx.accounts.existing_token_pool_pda.key(),
            &[token_pool_index.saturating_sub(1)],
        )
```

`token_pool_index.saturating_sub(1)` is used to check whether a token pool account with a previous bump already exists. This will be the same for both index 0 and 1.

Impact: This may lead to Incorrect Token Pool Verification.

Recommendation: consider handling the edge cases separately.

Status: Pending

References

- Light Protocol Documentation
- ZK Compression Overview
- Light Protocol: Batched State Merkle Tree Updates
- SolSec GitHub Repository by Sanny Kim
- Solana Documentation
- Helius Blog: A Hitchhiker's Guide to Solana Program Security

Appendix