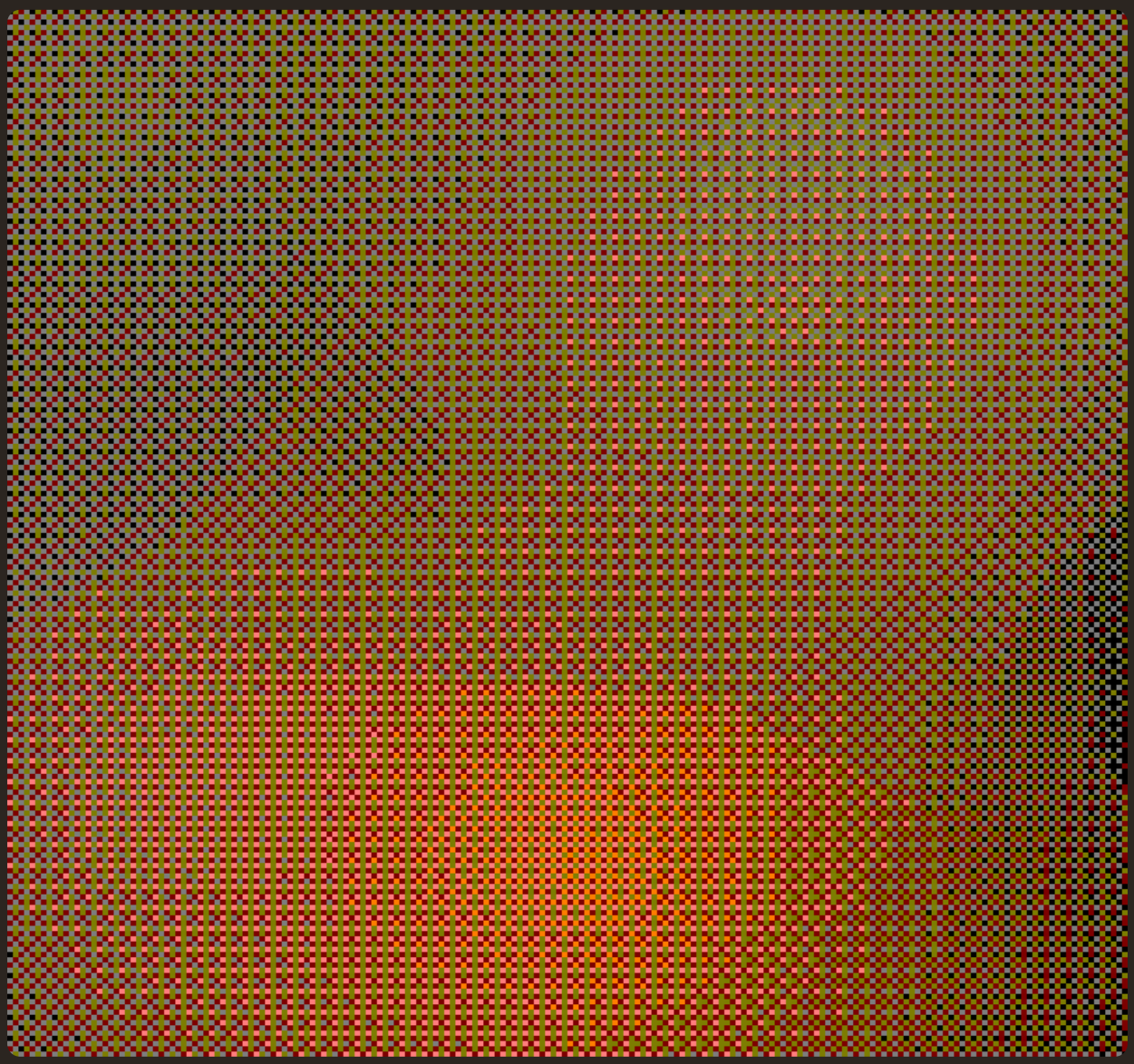
ZKSECURITY

Audit of Lighter's Spot Market Circuits and Multi-Asset Support

NOVEMBER 24TH, 2025 • TECHNICAL REPORT



Introduction

This report presents the findings of a security audit conducted on Lighter's ZK circuits for spot market trading. The audit was performed by zkSecurity starting November 24th, 2025, with two consultants over a period of two weeks. The engagement focuses on the circuit changes required to introduce spot market trading functionality and multi-asset support. This is the 6th audit in a series of audits on the Lighter protocol; we give a brief overview of the newly reviewed functionality below.

Scope

The audit was conducted in two phases on a private `lighter-prover` repository:

Phase 1 (commit `11c373555dd56478ddf7c9fbae27c2c2c4052665`) focused on the foundational multi-asset infrastructure:

COMPONENT	FILES
Delta Circuit	`delta_constraints.rs`, `account_delta_full_leaf.rs`, `utils.rs`
Asset Management	`l1_register_asset.rs`, `l1_update_asset.rs`
Deposit/Withdraw	`l1_deposit.rs`, `l1_withdraw.rs`, `l2_withdraw.rs`
Transfer	`l2_transfer.rs`
Supporting Types	`account_asset.rs`, `asset.rs`, `constants.rs`

Phase 2 (commit `b31b173ced2df143a7289b21df9020bfc664f3f6`) extended the review to order-related transactions and included fixes for Phase 1 findings:

COMPONENT	FILES
Matching Engine	`matching_engine.rs`, `apply_trade.rs`

COMPONENT	FILES
Order Transactions	`l1_create_order.rs`, `l2_create_order.rs`, `l2_cancel_order.rs`, `l2_modify_order.rs`, `internal_create_order.rs`, `internal_cancel_order.rs`, `internal_claim_order.rs`
Market Management	`l1_create_market.rs`, `l1_update_market.rs`, `market.rs`
Liquidation	`internal_liquidate_position.rs`, `internal_deleverage.rs`, `internal_exit_position.rs`
Transaction State	`tx_constraints.rs`, `tx_state.rs`, `transfer.rs`

Overview of Key Concepts

Multi-asset support. Prior to these changes, Lighter supported only USDC as the collateral asset for perpetual trading. The spot market feature introduces support for multiple assets (up to 62), each identified by an `asset_index`. Valid indices range from 1 to 62, with index 0 reserved as nil and 63 unused. USDC is assigned index 3. Each asset has configurable parameters including `margin_mode` (only enabled for USDC) and `extension_multiplier` for decimal precision handling. Assets are registered and configured via L1 priority transactions.

Spot vs perpetual operations. The protocol distinguishes between perpetual and spot operations across both fund movements and trading. For deposits, withdrawals, and transfers, two route types are supported: the `PERPS` route operates on the USDC-denominated `collateral` field used for perpetual margin, while the `SPOT` route operates on per-asset balances stored in an `aggregated_assets` array. The circuit enforces that `PERPS` routes can only be used with USDC. For trading, perpetual markets (indices 0–254) operate on collateral and positions, while spot markets (indices 2048+, with 255 reserved as nil) enable direct trading between asset pairs—the matching engine transfers base and quote assets directly between maker and taker accounts rather than updating margin positions.

Data availability. To support multi-asset accounts, the delta circuit was extended with an asset delta tree. This 64-element structure tracks balance changes per asset index and is included in the public digest. Only indices 1-62 are included in the

digest, matching the valid asset index range.

Exit hatch. The change to the account model also incurs a change in the emergency withdrawal mechanism (known as the *exit hatch* or *desert exit*). A call to the exit hatch must now specify an asset that the user wishes to withdraw. The circuit then performs a balance check for this asset. If the asset is traded in the spot market, then the balance check is a straightforward comparison. However, if the asset is used in the perps market (currently only USDC), then balance calculation reverts to the TAV calculation detailed in the report titled *Audit of Lighter's Exit Hatch*.

> *Note: The code in commit `b31b173` did not yet include witness generation for the desert exit circuits.*

Summary and Recommendations

The spot market and multi-asset implementation follows a sound architectural approach and no major architectural flaws were identified beyond the findings documented below. The circuit correctly separates SPOT and PERPS routes, enforces USDC-only constraints for collateral operations, and extends the matching engine to handle direct asset transfers for spot trades.

However, our review identified multiple significant issues requiring attention, including two high-severity findings related to collateral inflation and data availability. In general we observed that the management of reserved asset indices is somewhat fragile. The codebase defines `MIN_ASSET_INDEX` and `MAX_ASSET_INDEX` constants suggesting a configurable valid range, yet the validation logic (e.g., `ensure_valid_asset_index`) checks only for the specific boundary values 0 and 63 rather than validating against the constants. This inconsistency could lead to errors if the range definition changes. We recommend unifying the approach: either validate against `MIN_ASSET_INDEX` and `MAX_ASSET_INDEX` throughout, or simplify to explicit checks for 0 and 63 if these are the only reserved indices and will remain so.

The phased nature of the engagement (with fixes for Phase 1 findings incorporated into the Phase 2 commit) reflects that the codebase was under active development during the audit. We recommend establishing a stabilization period before deployment to ensure all findings are addressed and the changes are thoroughly tested.

Findings

Below are listed the findings found during the engagement. High severity findings can be seen as so-called "priority 0" issues that need fixing (potentially urgently). Medium severity findings are most often serious findings that have less impact (or are harder to exploit) than high-severity findings. Low severity findings are most often exploitable in contrived scenarios, if at all, but still warrant reflection. Findings marked as informational are general comments that did not fit any of the other criteria.

ID	COMPONENT	NAME	RISK
#00	circuit/src/transactions/l1_deposit.rs	L1Deposit PERPS route lacks USDC check	**High**
#01	circuit/src/tx_constraints.rs, circuit/src/delta/delta_constraints.rs	Missing asset index range validation corrupts data availability	**High**
#02	circuit/src/types/tx_state.rs	Inverted logic in maximum order quote amount check	**Medium**
#03	circuit/src/transactions/l1_update_asset.rs	L1UpdateAsset allows enabling margin mode on non-USDC assets	**Medium**
#04	circuit/src/transactions/l1_update_asset.rs, circuit/src/transactions/l1_register_asset.rs	Asset index validation inconsistencies in asset management transactions	**Low**

ID	COMPONENT	NAME	RISK
#05	circuit/src/delta/delta_constraints.rs	Non-zero asset count in delta digest off by two	**Informational**

#00 - L1Deposit PERPS route lacks USDC check

Severity: High **Location:** circuit/src/transactions/l1_deposit.rs

Description. The `L1Deposit` transaction supports two route types: `SPOT` (credits to asset balance) and `PERPS` (credits to collateral). When using the `PERPS` route, the circuit does not validate that the deposited asset is USDC or has `margin_mode` enabled.

In `l1_deposit.rs` lines 195-235, the `apply()` function determines the route and credits collateral without any asset validation:

```
let is_perps = builder.is_equal_constant(self.route_type, ROUTE_TYPE_PERPS as
u64);

let extended_balance_delta_biguint = builder.mul_biguint_non_carry(
    &self.accepted_amount,
    &tx_state.assets[TX_ASSET_ID].extension_multiplier,
    BIG_U96_LIMBS,
);

// ...

let should_update_perps_balance = builder.and(should_update_balance, is_perps);
tx_state.accounts[OWNER_ACCOUNT_ID].collateral = builder.select_bigint(
    should_update_perps_balance,
    &BigIntTarget { abs: trimmed_abs_collateral_after, sign: balance_after.sign },
    &tx_state.accounts[OWNER_ACCOUNT_ID].collateral,
);
```

The collateral field is an USDC asset (as evidenced by `USDC_TO_COLLATERAL_MULTIPLIER` usage throughout the codebase). However, the circuit allows any asset to be deposited via the `PERPS` route and credited to collateral using that asset's `extension_multiplier`.

Compare this to `l2_transfer.rs` lines 206-218, which correctly validates both `margin_mode` and `is_usdc_asset` for PERPS routes:

```
let is_asset_margin_enabled = builder.is_equal_constant(
    tx_state.assets[TX_ASSET_ID].margin_mode,
    ASSET_MARGIN_MODE_ENABLED,
);
// ...
let is_invalid_route_type = builder.and_not(is_perps, is_asset_margin_enabled);
builder.conditional_assert_false(is_enabled, is_invalid_route_type);

let is_to_perps_invalid_route = builder.and_not(is_perps, is_usdc_asset);
builder.conditional_assert_false(self.success, is_to_perps_invalid_route);
```

Impact. An attacker can inflate their USDC-denominated collateral by depositing a low-value token via the `PERPS` route:

1. Attacker deposits 1000 units of cheap token X via L1 with `route_type=PERPS`
2. Circuit credits `1000 * X.extension_multiplier` to their collateral
3. Attacker now has inflated USDC collateral without depositing actual USDC

This inflated collateral can then be used to withdraw real USDC via the PERPS route, or to take leveraged positions beyond the attacker's actual capital.

Recommendation. Add validation in `L1Deposit` that when `route_type` is `PERPS`, the `asset_index` must be `USDC_ASSET_INDEX` (or alternatively, verify `margin_mode` is enabled). This should mirror the validation present in `l2_transfer.rs`.

Client response. Fixed in commit `b31b173` (Phase 2), refined in commit `f6818da`. The fix now checks that `margin_mode` is enabled for the asset when using the PERPS route with non-zero accepted amount, rather than hardcoding a check for `USDC_ASSET_INDEX`. Note that currently only USDC has `margin_mode` enabled. If margin mode is enabled for additional assets in the future while collateral remains USDC-denominated, this check would need to be revisited to prevent the same inflation issue.

#01 - Missing asset index range validation corrupts data availability

Severity: High **Location:** circuit/src/tx_constraints.rs, circuit/src/delta/delta_constraints.rs

Description. Transaction processing constraints do not validate that `asset_index` values fall within the valid range of 1 to 62. The system reserves indices 0 and 63 as invalid, but this is not enforced at the circuit level.

In `tx_constraints.rs`, asset indices are created as virtual targets without range constraints:

```
asset_indices: core::array::from_fn(|_| builder.add_virtual_target()),
```

These indices are subsequently used to derive Merkle paths and access asset data, but are never checked against `MIN_ASSET_INDEX` (1) or `MAX_ASSET_INDEX` (62).

This becomes problematic due to an asymmetry in the delta circuit. In `account_delta_full_leaf.rs` lines 143-166, the asset delta root calculation iterates over all 64 indices:

```
pub fn get_asset_delta_root(&self, builder: &mut Builder) -> HashOutTarget {
    let mut level_hashes = self
        .aggregated_asset_deltas
        .iter()  // All 64 elements (0-63)
        .map(|a| {
            // ... hash each element
        })
        .collect::<Vec<_>>();
    // ... builds merkle tree from all 64
}
```

However, in `delta_constraints.rs` lines 174-193, the public digest computation only includes indices 1-62:

```
for i in MIN_ASSET_INDEX as usize..=MAX_ASSET_INDEX as usize {
    let is_asset_empty =
builder.is_zero_bigint(&delta.aggregated_asset_deltas[i]);
    // ...
}
```

Indices 0 and 63 are neither checked for emptiness nor included in the public digest, yet they contribute to the asset delta root and thus the state root.

Impact. An attacker can create transactions targeting asset indices 0 or 63. Since the circuit does not enforce range constraints, such transactions will:

1. Pass circuit verification (the user signs a valid transaction, just with an out-of-range index)
2. Affect the state root (via asset delta root which includes all 64 indices)
3. Not be reflected in the public digest (which only covers indices 1-62)

This corrupts data availability: the published pubdata no longer represents the complete state. Users relying on the escape hatch mechanism cannot reconstruct the full state from pubdata, potentially preventing emergency withdrawals.

This attack does not require a malicious prover or operator; any user can sign a transaction with an invalid asset index, and the sequencer will process it since the circuit accepts it.

Recommendation. Add range validation for `asset_index` in transaction constraints to ensure values fall within `MIN_ASSET_INDEX` to `MAX_ASSET_INDEX`. This is the primary fix as it prevents invalid transactions from being processed.

As defense in depth, consider also enforcing in the delta circuit that asset deltas at indices 0 and 63 must be zero, ensuring consistency between the asset delta root and the public digest.

Client response. Fixed in commit `b31b173` (Phase 2). Added `ensure_valid_asset_index` checks in relevant transactions that reject asset indices 0 and 63 (i.e., `MIN_ASSET_INDEX - 1` and `MAX_ASSET_INDEX + 1`).

#02 - Inverted logic in maximum order quote amount check

Severity: Medium **Location:** circuit/src/types/tx_state.rs

Description. The `is_valid_base_size_and_price` function contains a logic error that inverts the check for the maximum order quote amount.

The code calculates the boolean flag `quote_gt_max_quote_amount` as follows:

```
let quote_gt_max_quote_amount = builder.is_lt_biguint(&quote_big, &max_quote_big);
```

This expression evaluates to `true` when `quote_big` is strictly less than `max_quote_big`. However, the variable name and subsequent logic imply it should be `true` when the quote amount *exceeds* the maximum limit.

Later in the function, this flag is used to invalidate the order:

```
let should_be_false = builder.or(should_be_false, quote_gt_max_quote_amount);
builder.not(should_be_false)
```

Because `quote_gt_max_quote_amount` is true for valid orders (those within the limit), the circuit incorrectly rejects them. Conversely, it accepts orders that are greater than or equal to the maximum limit (where `is_lt_biguint` returns false).

Recommendation. Correct the comparison to check if the quote amount is greater than the maximum limit. This can be done by replacing `is_lt_biguint` with `is_gt_biguint`.

Client response. Fixed in commit `f6818da` on the `spot` branch. Changed `is_lt_biguint` to `is_gt_biguint` as recommended.

#03 - L1UpdateAsset allows enabling margin mode on non-USDC assets

Severity: Medium **Location:** circuit/src/transactions/l1_update_asset.rs

Description. The `L1UpdateAsset` transaction allows updating asset configuration parameters including `margin_mode`. However, the circuit does not enforce the invariant that only USDC (the collateral asset) should have `margin_mode` set to `ENABLED`.

In `l1_update_asset.rs` lines 83-104, the `verify()` function only checks that `margin_mode` is a boolean value:

```
builder.assert_bool(BoolTarget::new_unsafe(self.margin_mode));
```

It does not verify that if `margin_mode` is being set to `ENABLED`, the asset must be USDC. Compare this to the correct implementation in `l1_register_asset.rs` lines 125-130:

```
let is_margin_enabled = builder.is_equal_constant(self.margin_mode,
ASSET_MARGIN_MODE_ENABLED);
let is_usdc_asset = builder.is_equal_constant(self.asset_index, USDC_ASSET_INDEX);
let should_be_false = builder.and_not(is_margin_enabled, is_usdc_asset);
self.success = builder.and_not(self.success, should_be_false);
```

The `L1RegisterAsset` correctly enforces that enabling margin mode on a non-USDC asset causes the transaction to fail. This same constraint is missing from `L1UpdateAsset`.

Impact. An operator could enable `margin_mode` on a non-USDC asset, breaking a protocol invariant. Several transaction types rely on the `margin_mode` check as a guard for PERPS route operations (e.g., `l1_withdraw.rs`, `l2_withdraw.rs`). If this invariant is violated, those guards become ineffective, potentially allowing non-USDC assets to interact with collateral incorrectly.

The severity is reduced because exploiting this requires operator action (L1 priority operation), and impact depends on other transaction behaviors. However, it represents a defense-in-depth failure that could become critical if combined with other issues.

Recommendation. Add the same constraint present in `L1RegisterAsset` to `L1UpdateAsset`. Specifically, when `margin_mode` is set to `ENABLED`, the circuit should verify that `asset_index` equals `USDC_ASSET_INDEX` and fail the transaction

otherwise.

Client response. Fixed in commit `b31b173` (Phase 2). Added the same constraint from `L1RegisterAsset`: if `margin_mode` is enabled on a non-USDC asset, the transaction now fails.

#04 - Asset index validation inconsistencies in asset management transactions

Severity: Low **Location:** circuit/src/transactions/l1_update_asset.rs, circuit/src/transactions/l1_register_asset.rs

Description. The asset management transactions have minor inconsistencies in asset index validation:

1. `l1_update_asset.rs` does not check that `asset_index` falls within `MIN_ASSET_INDEX` to `MAX_ASSET_INDEX`. Currently this is mitigated because assets must be non-empty to be updatable, and assets cannot be registered outside the valid range. However, adding an explicit range check would provide defense in depth.

2. `l1_register_asset.rs` checks for invalid indices using `builder.is_zero(self.asset_index)` instead of comparing against `MIN_ASSET_INDEX`. While functionally equivalent (since `MIN_ASSET_INDEX = 1`), using the constant would be more consistent and self-documenting.

Recommendation. Add explicit range validation in `l1_update_asset.rs` and replace the hardcoded zero check in `l1_register_asset.rs` with `MIN_ASSET_INDEX` for consistency.

Client response. Fixed in commit `b31b173` (Phase 2). Both transactions now use `ensure_valid_asset_index` which checks against `MIN_ASSET_INDEX` and `MAX_ASSET_INDEX` constants.

#05 - Non-zero asset count in delta digest off by two

Severity: Informational **Location:** circuit/src/delta/delta_constraints.rs

Description. The `nonzero_asset_count` calculation in the delta circuit is off by two due to a mismatch between the loop iteration range and the total asset list size.

The circuit computes the count of non-zero asset deltas as follows:

```
let mut zero_asset_count = builder.zero();
for i in MIN_ASSET_INDEX as usize..=MAX_ASSET_INDEX as usize {  // 1..=62
    let is_asset_empty =
builder.is_zero_bigint(&delta.aggregated_asset_deltas[i]);
    zero_asset_count = builder.add(zero_asset_count, is_asset_empty.target);
}
let total_asset_count = builder.constant_usize(ASSET_LIST_SIZE);  // 64
let nonzero_asset_count = builder.sub(total_asset_count, zero_asset_count);
```

The loop only iterates over indices 1 through 62 (the valid asset range, as indices 0 and 63 are reserved), counting at most 62 zero assets. However, `total_asset_count` was set to 64 (the full `ASSET_LIST_SIZE`). This meant `nonzero_asset_count` is always 2 higher than the actual count of non-zero assets.

For example, if all 62 valid assets had zero deltas, `zero_asset_count` would be 62, and `nonzero_asset_count` would compute to `64 - 62 = 2` instead of the correct value of 0.

This value is included in the delta digest used for public data reconstruction. If the off-chain reconstruction code uses the same logic, the system remains internally consistent. However, the semantic meaning of the field would be incorrect.

Recommendation. Initialize `zero_asset_count` to 2 to account for the reserved indices 0 and 63, or adjust `total_asset_count` to 62 to match the iteration range.

Client response. Fixed in commit `b31b173` (Phase 2) by initializing `zero_asset_count` to `builder.two()`.