Audit Report

December 19, 2025

Sundae Labs - Stableswap

Contents

1 - Summary

This report provides a comprehensive audit of Sundae Stableswap, an adaptation of SundaeSwap V3 that implements AMM liquidity pools for assets that maintain a stable 1:1 exchange ratio.

The investigation spanned several potential vulnerabilities, including scenarios where attackers might exploit the validator to lock up or steal funds.

The audit is conducted without warranties or guarantees of the quality or security of the code. It's important to note that this report only covers identified issues, and we do not claim to have detected all potential vulnerabilities.

1.a - Overview

The core component of Sundae Stableswap is the liquidity pool. Liquidity pools are script UTxOs that hold liquidity for two fixed assets. Standard operations are supported, such as swapping and providing/removing liquidity, together with more advanced operations.

To address concurrency, users does not interact directly with LPs but place orders. Orders are script UTxOs that hold all the assets and information required for the execution of the desired operation. They can be directed to a specific pool, or open to any pool that is able to process it.

Orders are processed in batches in "scoop" transactions by authorized entities called "scoopers". A scoop transaction applies a sequence of orders to a specific pool, transforming the pool state and doing all the required payments to fulfill the orders purpose.

The protocol is booted by the creation of a single settings UTxO that governs the entire protocol. The settings UTxO determine, among other things, the list of authorized scoopers. Liquidity pools are created and validated with the minting of a pool NFT.

Orders are created with no validation, so it is up to the scoopers to select well-formed orders to be processed. There are several order types:

- Swap: to swap one token for another.
- Deposit: to provide liquidity and obtain LP tokens.
- Withdrawal: to redeem LP tokens and remove liquidity.
- Strategy: to lock funds for an operation that will be determined at processing time by a designated signer.
- Record: to create an output that can be used to do a snapshot of the pool state (than can be used later as an oracle).

1.b - Process

Our audit process involved a thorough examination of Sundae Stableswap validators. Areas vulnerable to potential security threats were closely scrutinized, including those where attackers could exploit the validator's functions to disrupt the platform and its users. This included evaluating potential risks such as unauthorized asset addition, hidden market creation, and disruptions to interoperability with other Plutus scripts. This also included the common vulnerabilities such as double satisfaction and minting policy vulnerabilities.

The audit took place over a period of several weeks, and it involved the evaluation of the protocol's mathematical model to verify that the implemented equations matched the expected behavior.

Findings and feedback from the audit were communicated regularly to the Sundae team through Discord. Diagrams illustrating the necessary transaction structure for proper interaction with the protocol are attached as part of this report. The Sundae team addressed these issues in an efficient and timely manner, enhancing the overall security of the platform.

1.c - Audited Files and covered commits

Below is a list of all audited files in this report. Any files **not** listed here were **not** audited.

The Stableswap project is based on SundaeSwap V3 AMM protocol, which was previously audited by TxPipe. This audit covers a range of commits starting from the final state of the previous audit at commit `edc118880d3baffcb7d5bd277faec2e7dc54c59b`, and ending at commit `e84f0ab8a76c332cafa026b0a4b1cd171b34eacc`.

Filename
./validators/stake.ak
./validators/settings.ak
./validators/pool.ak
./validators/oracle.ak
./validators/pool_stake.ak
./validators/order.ak
./lib/calculation/record.ak
./lib/calculation/swap.ak
./lib/calculation/strategy.ak
./lib/calculation/process.ak
./lib/calculation/stableswap.ak
./lib/calculation/withdrawal.ak
./lib/calculation/deposit.ak
./lib/calculation/shared.ak
./lib/types/settings.ak
./lib/types/pool.ak
./lib/types/oracle.ak

```
./lib/types/order.ak
```

```
./lib/shared.ak
```

2 - Specification

2.a - Business logic

In this section we describe the Stableswap business logic. We focus on the math behind the main operations, abstracting out all the technical details related to the Cardano ledger and the Stableswap architecture.

2.a.a - Stableswap invariant

As in a constant product AMM, Stableswap uses an invariant formula that defines how the X and Y liquidities are related to each other. The invariant must be preserved on each swap operation, so it determines the swapping price in every possible scenario.

To provide a stable 1:1 price, the invariant includes a linear component and a linear amplification parameter A that determines the influence of this term. For two assets X and Y with liquidities x and y respectively, the invariant formula is:

$$4A(x+y) + D = 4AD + \frac{D^3}{4xy}$$

Here, D represents the depth of liquidity in the pool. A full derivation of this formula can be found in Curve's Stableswap whitepaper.[1]

Figure 1 shows an example of the Stableswap invariant curve. The straight section enforces a stable price, because adding liquidity of one asset requires removing the same amount of the other asset. Y values are obtained numerically using Newton's method as explained in the RareSkills blog article "get_D() and get_y() in Curve StableSwap".[2]

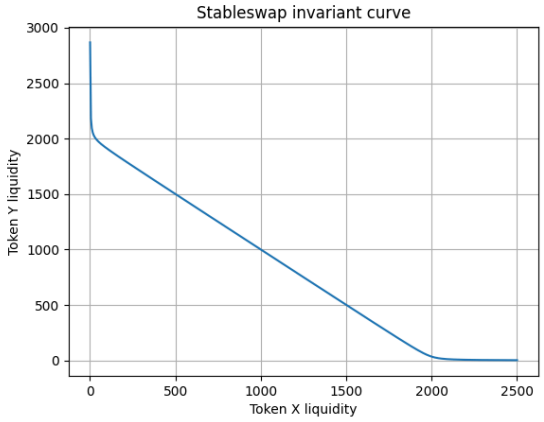


Figure 1: Stableswap invariant curve for sum invariant $D = 2000$ and linear amplification $A = 200$.

2.a.b - Swap overview

A swap operation starts with a user creating an order offering a certain amount of asset X in exchange for asset Y. The amount of asset Y extracted from the pool is calculated to ensure that the Stableswap invariant is preserved after offered asset X is added.

[1] Michael Egorov. "StableSwap - efficient mechanism for Stablecoin liquidity". https://docs.curve.fi/assets/pdf/whitepaper_stableswap.pdf

[2] https://www.rareskills.io/post/curve-get-d-get-y

To the extracted amount of asset Y, two types of fees are applied: protocol fees and liquidity provider (LP) fees. Protocol fees are stored in the pool but logically separated from the liquidity. LP fees are added back as liquidity. As liquidity is increased, a new value for the sum invariant D must be computed to maintain the stableswap invariant.

Finally, the resulting amount of asset Y after fees deduction is paid to the user.

2.a.c - Swap off-chain procedure

We describe here with more detail the off-chain procedure for processing a swap order.

Relevant constants:
- `precision`: Multiplier for liquidities in the Stableswap invariant.

Relevant pool settings:
- `lp_fee_rate`: Fee rate for the liquidity providers. 100% is 10_000.
- `protocol_fee_rate`: Fee rate for the protocol. 100% is 10_000.
- `linear_amplification`: Stableswap A constant.

Revelant pool initial state:
- `pool_quantity_x`: X liquidity.
- `pool_quantity_y`: Y liquidity.
- `sum_invariant`: Stableswap D value.

Swap order:
- `order_give`: offered amount of token X.

Offchain procedure:

1. Compute `raw_swap_result`: Given `precision`, `pool_quantity_x`, `pool_quantity_y`, `sum_invariant`, `linear_amplification` and `order_give`, find y such that the Stableswap invariant holds, using Newton's method:

   $$4A(x+y) + D = 4AD + \frac{D^3}{4xy}$$

   where:
   - A is `linear_amplification`
   - D is `sum_invariant`
   - x is `(pool_quantity_x + order_give) * precision`
   - y is `pool_quantity_y * precision - raw_swap_result`

2. Observe that `raw_swap_result` is a high precision integer. Obtain the actual amount in asset Y:

   ```
   swap_result = raw_swap_result / precision
   ```

3. Compute `total_fees`, `protocol_fees` and `lp_fees`:

   ```
   total_fees = ( swap_result * total_fee_rate + 9999 ) / 10000
   protocol_fees = total_fees * protocol_fee_rate / total_fee_rate
   lp_fees = total_fees - protocol_fees
   ```

   where `total_fee_rate = lp_fee_rate + protocol_fee_rate`

   Observe that:
   - `total_fees` is rounded up

- `protocol_fees` is rounded down
- `lp_fees` is rounded up

4. Compute `new_sum_invariant`: Given the new liquidities x and y, find D such that the Stableswap invariant holds, using Newton's method:

$$4A(x+y) + D = 4AD + \frac{D^3}{4xy}$$

where:
- A is `linear_amplification`
- D is `new_sum_invariant`
- x is `(pool_quantity_x + order_give) * precision`
- y is `(pool_quantity_y - swap_result + lp_fees) * precision`

2.a.d - Swap on-chain checks

The on-chain code must check that all off-chain computations were correctly done. Steps 1 and 4 of the off-chian procedure are solved using Newton's method, something that is not feasible to reproduce on-chain. Instead, both results `raw_swap_result` and `new_sum_invariant` are provided in the redeemer, and the on-chain code checks that the provided values correctly solve the equation.

In the case of step 1, the on-chain function is called `exchange_invariant`, and checks that y is the smallest integer such that

$$4A(x+y) + D \geq 4AD + \frac{D^3}{4xy}$$

where:
- A is `linear_amplification`
- D is `sum_invariant`
- x is `(pool_quantity_x + order_give) * precision`
- y is `pool_quantity_y * precision - raw_swap_result`

For step 4, the on-chain function is `liquidity_invariant`, and checks that D is the biggest integer such that

$$4A(x+y) + D \geq 4AD + \frac{D^3}{4xy}$$

where:
- A is `linear_amplification`
- D is `new_sum_invariant`
- x is `(pool_quantity_x + order_give) * precision`
- y is `(pool_quantity_y - swap_result + lp_fees) * precision`

2.a.e - Deposit overview

A deposit operation begins with a user offering liquidity to the pool. Any amount of assets X and Y can be offered, with no restriction on the ratio between them. In return, the user receives LP tokens in proportion to the observed increase in the sum invariant. The first deposit at pool creation follows the same logic, but the amount of LP tokens minted is equal to the resulting sum invariant.

Relevant constants:
- `precision`: Multiplier for liquidities in the Stableswap invariant.

Relevant pool settings:
- `linear_amplification`: Stableswap A constant.

Revelant pool initial state:
- `pool_quantity_x`: X liquidity.
- `pool_quantity_y`: Y liquidity.
- `pool_quantity_lp`: Circulating LP tokens.
- `sum_invariant`: Stableswap D value.

Deposit order:
- `deposited_x`: Offered X new liquidity.
- `deposited_y`: Offered Y new liquidity.

Offchain procedure:

1. Compute `next_sum_invariant`:

Given the new liquidities x and y, find D such that the Stableswap invariant holds, using Newton's method:

$$4A(x+y) + D = 4AD + \frac{D^3}{4xy}$$

where:
- A is `linear_amplification`
- D is `next_sum_invariant`
- x is `(pool_quantity_x + deposited_x) * precision`
- y is `(pool_quantity_y + deposited_y) * precision`

2. Compute `issued_lp_tokens`:

   - For the first deposit:

     ```
     issued_lp_tokens = next_sum_invariant
     ```

   - For subsequent deposits:

     ```
     issued_lp_tokens = (next_sum_invariant - sum_invariant) * pool_quantity_lp
                        / sum_invariant
     ```

2.a.f - Withdrawal overview

A withdrawal operation begins with a user offering an amount of LP tokens to be burned. In return, the user receives amounts of X and Y proportional to the precentage of LP tokens burned.

Relevant constants:
- `precision`: Multiplier for liquidities in the Stableswap invariant.

Relevant pool settings:
- `linear_amplification`: Stableswap A constant.

Revelant pool initial state:
- `pool_quantity_x`: X liquidity.
- `pool_quantity_y`: Y liquidity.

- `pool_quantity_lp`: Circulating LP tokens.
- `sum_invariant`: Stableswap D value.

Withdraw order:
- `withdrawn`: Offered LP tokens to be burned.

Offchain procedure:

1. Compute withdrawn liquidity:

   ```
   withdrawn_x = withdrawn * pool_quantity_x / pool_quantity_lp
   withdrawn_y = withdrawn * pool_quantity_y / pool_quantity_lp
   ```

2. Compute `next_sum_invariant`: Given the new liquidities x and y, find D such that the Stableswap invariant holds, using Newton's method:

   $$4A(x+y) + D = 4AD + \frac{D^3}{4xy}$$

   where:
   - A is `linear_amplification`
   - D is `next_sum_invariant`
   - x is `(pool_quantity_x - withdrawn_x) * precision`
   - y is `(pool_quantity_y - withdrawn_y) * precision`

2.b - UTxOs

2.b.a - Settings UTxO

A single script UTxO that is created at launch and used for the entire protocol. Creation is validated with the minting of the "Settings NFT" that is locked into the UTxO. A multivalidator is used to contain both the spend and the minting validators.

- Address:
  - Payment part: Settings validator script hash. Parameters:
    - `protocol_boot_utxo`: seed UTxO to be spent at settings NFT creation.
  - Staking part: any
- Value:
  - 1 settings
- Datum: `SettingsDatum`
  - `settings_admin`: multisig that identifies the settings admin.
  - `metadata_admin`: address to which the CIP-68 tokens are minted.
  - `treasury_admin`: multisig that identifies the treasury admin.
  - `treasury_address`: address to which the treasury funds must be withdrawn.
  - `treasury_allowance`: a specific percentage of operating funds that go to operating costs e.g. paying for things like the scoopers, server infrastructure, etc
  - `authorized_scoopers`: list of addresses that are allowed to scoop.
  - `authorized_staking_keys`: list of authorized staking keys that can be attached to a pool.
  - `base_fee`: base fee in lovelace for the protocol. To be collected each time a batch happens.
  - `simple_fee`: simple fee for the protocol, to charge each "simple" order, such as a swap, deposit, withdrawal, etc.
  - `strategy_fee`: strategy fee for the protocol, to charge each "strategy" execution.
  - `pool_creation_fee`: fee for the protocol, to charge each time a pool is created.
  - `extensions`: arbitrary data that can be added to the datum.

2.b.b - Pool UTxOs

- Address:
  - Payment part: Pool validator script hash. Parameters:
    - `manage_stake_script_hash`: hash of staking script that validates pool management operations.
    - `settings_policy_id`: minting policy of the Settings NFT.
  - Staking part: any
- Value:
  - accumulated protocol fees + min ADA
  - 1 pool NFT
  - (A, B): pair of assets contained by the pool.
- Datum: `StablePoolDatum`
  - `identifier`: identifier of the pool.
  - `assets`: the two asset IDs that this pool can exchange.
  - `circulating_lp`: total number of LP tokens in circulation.
  - `lp_fee_basis_points`: basis points to charge on each trade for bid (A -> B) and ask (B -> A) orders
  - `protocol_fee_basis_points`: percentage cut of the LP fees the protocol collects.
  - `fee_manager`: optional multisig condition under which the protocol fees can be updated.
  - `market_open`: UNIX millisecond timestamp at which trading against the pool should be allowed.
  - `protocol_fees`: 3-tuple of amounts of fees the protocol has collected from trading activity
    - first field is the ADA collected from raw scooper fees
    - second field is asset A collected as a percentage of LP fees
    - third field is asset B collected as a percentage of LP fees
  - `linear_amplification`: `A` from the Curve stableswap formula
  - `sum_invariant`: `D` from the Curve stableswap formula

2.b.c - Order UTxOs

- Address: hash of spend validator in `order.ak`. Parameters:
  - `stake_script_hash`: hash of staking script that validates for the presence of a valid liquidity pool.
- Value:
  - ADA: at least min ADA.
  - Other: assets relevant to the order + others.
- Datum: `OrderDatum`
  - `pool_ident`: optional pool identifier. If not present, the order is allowed to execute against any pool with the correct assets.
  - `owner`: order owner in a multisig scheme. Allows cancellation and updates.
  - `max_protocol_fee`: max amount of ADA that could be taken as protocol fee.
  - `destination`: where the order execution output should be sent.
  - `details`: description of how to actually execute the order.
  - `extensions`: arbitrary data that can be added to the datum.

2.c - Assets

2.c.a - Pool NFTs

Minted by the Mint purpose of the pool multivalidator when creating a pool. Locked into the pool UTxO.

- Policy ID: Pool validator script hash.
- Asset name: `(222)` + pool identifier.

2.c.b - Pool Reference NFTs

Minted by the Mint purpose of the pool multivalidator when creating a pool. Paid to the metadata administrator, and is meant to be used to track the pool's on-chain metadata.

- Policy ID: Pool validator script hash.
- Asset name: `(100)` + pool identifier.

2.c.c - LP Tokens

Minted by the Mint purpose of the pool multivalidator, paid to liquidity providers. Represent a percentage ownership of the pool.

- Policy ID: Pool validator script hash.
- Asset name: `(333)` + pool identifier.

2.c.d - Settings NFTs

Minted by the Mint purpose of the settings multivalidator. Paid to the Settings UTxO.

- Policy ID: Settings validator script hash.
- Asset name: `settings_nft_name`.

2.c.e - Oracle NFTs

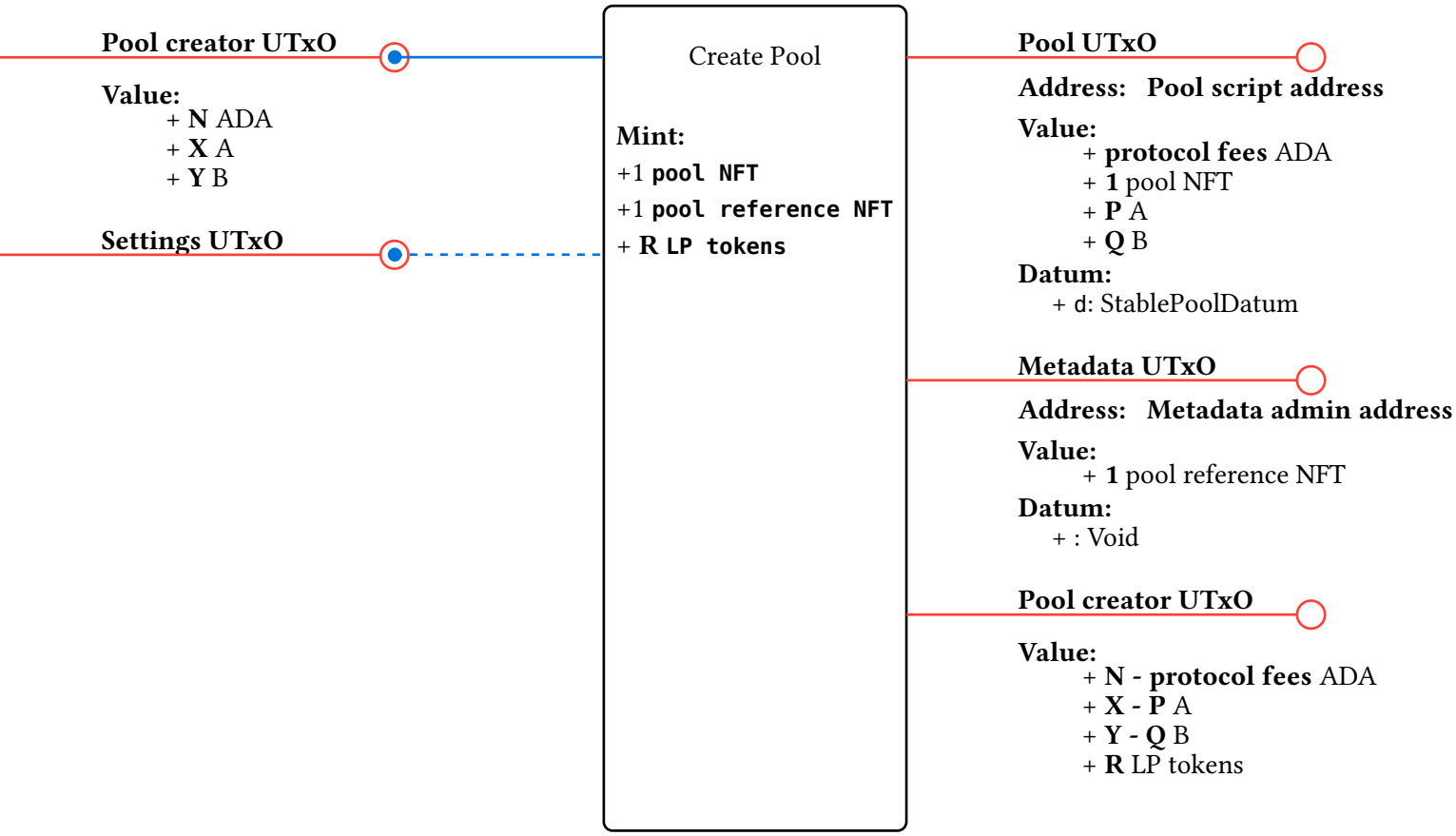
Minted by the Mint purpose of the oracle multivalidator. Paid to the oracle UTxO.

- Policy ID: Oracle validator script hash.
- Asset name: `oracle_sft_name()`.

2.d - Transactions

2.d.a - Pools

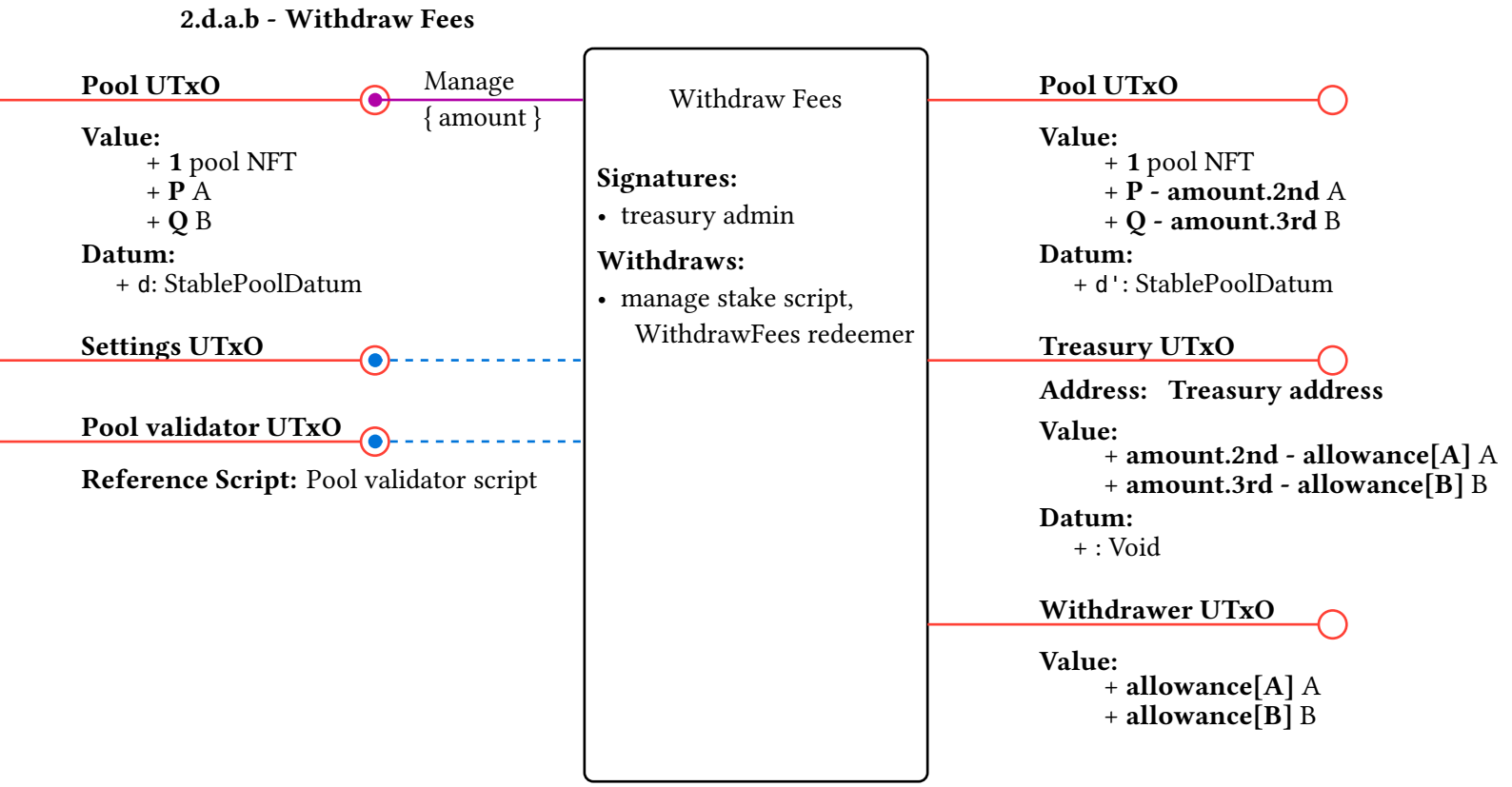
2.d.a.a - Create pool



Note:

```
d: StablePoolDatum = {
  identifier: ByteArray
  assets: (AssetClass, AssetClass) = (A, B)
  circulating_lp: Int = R
  lp_fee_basis_points: (Int, Int)
  protocol_fee_basis_points: (Int, Int)
  fee_manager: Option<MultisigScript>
  market_open: Int
  protocol_fees: (Int, Int, Int) = (settings_datum.pool_creation_fee, 0, 0)
  linear_amplification: Int
  sum_invariant: Int
}
```

Figure 2: Create Pool transaction

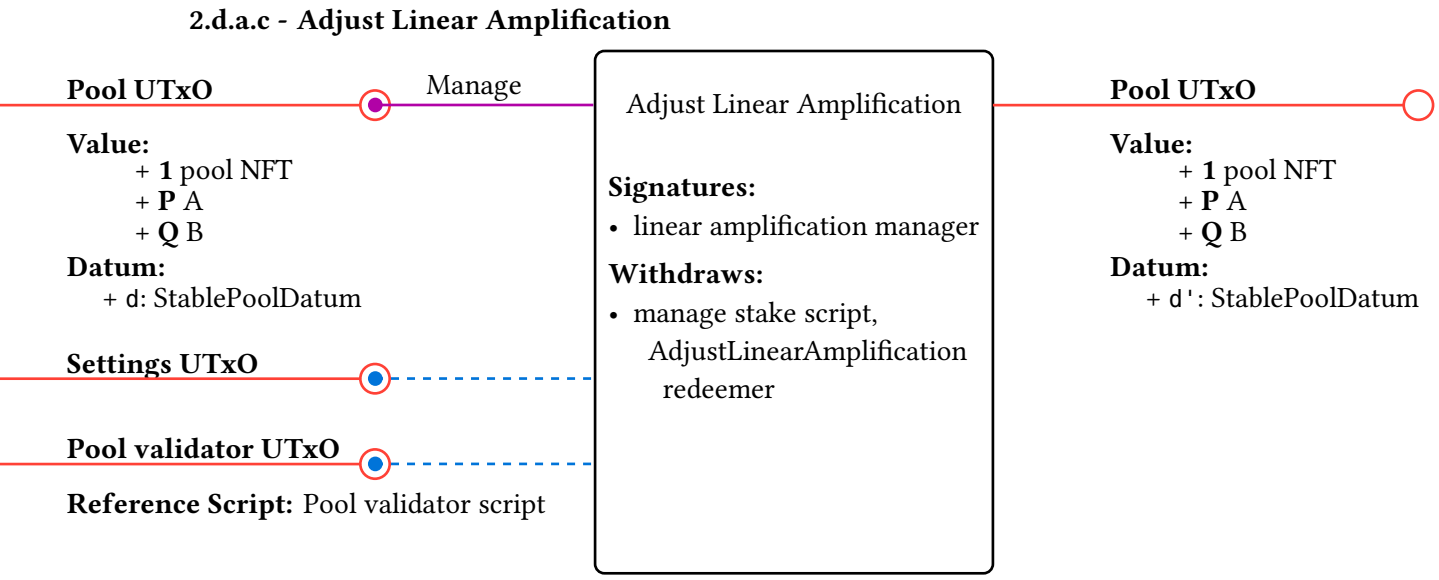


Note:

amount: (Int, Int, Int), where 1st is ADA collected as scooper fees, and 2nd is A and 3rd is B, both collected as LP fees

d'.protocol_fees = (d.protocol_fees.1st - amount.1st, d.protocol_fees.2nd - amount.2nd, d.protocol_fees.3rd - amount.3rd)

Figure 3: Withdraw Fees transaction



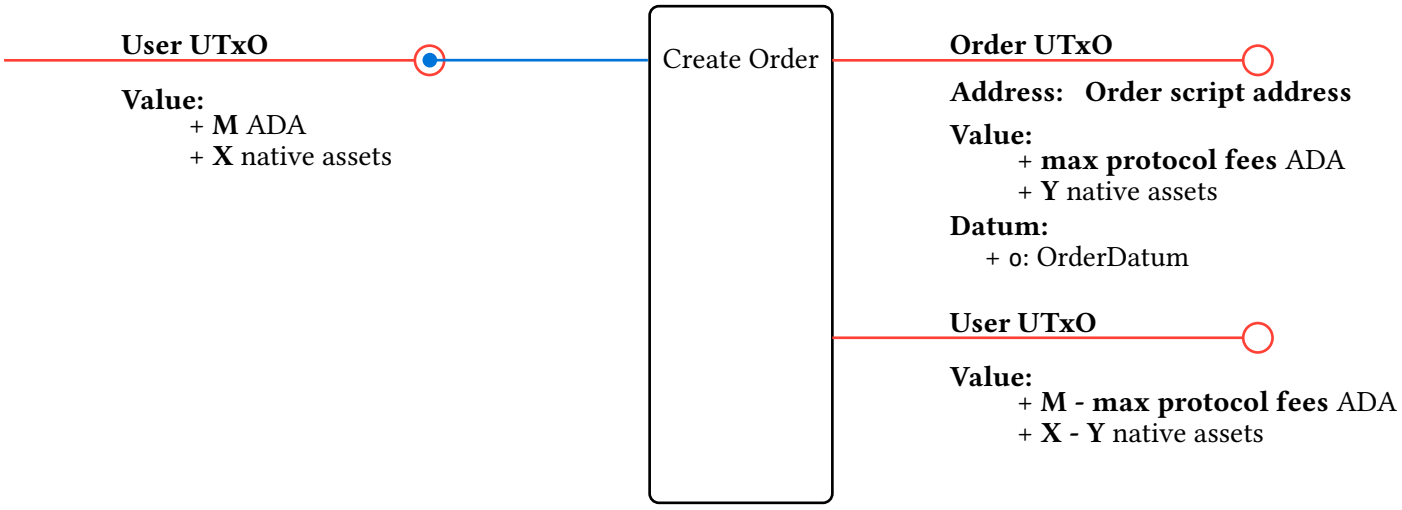
Note:

$$\text{AdjustLinearAmplification}(\text{d.linear\_amplification}) = \text{d'.linear\_amplification}$$
$$\text{AdjustLinearAmplification}(\text{d.sum\_invariant}) = \text{d'.sum\_invariant}$$
$$\text{AdjustLinearAmplification}(\text{d.linear\_amplification\_manager}) = \text{d'.linear\_amplification\_manager}$$

Figure 4: Adjust Linear Amplification transaction

2.d.b - Orders

2.d.b.a - Create order



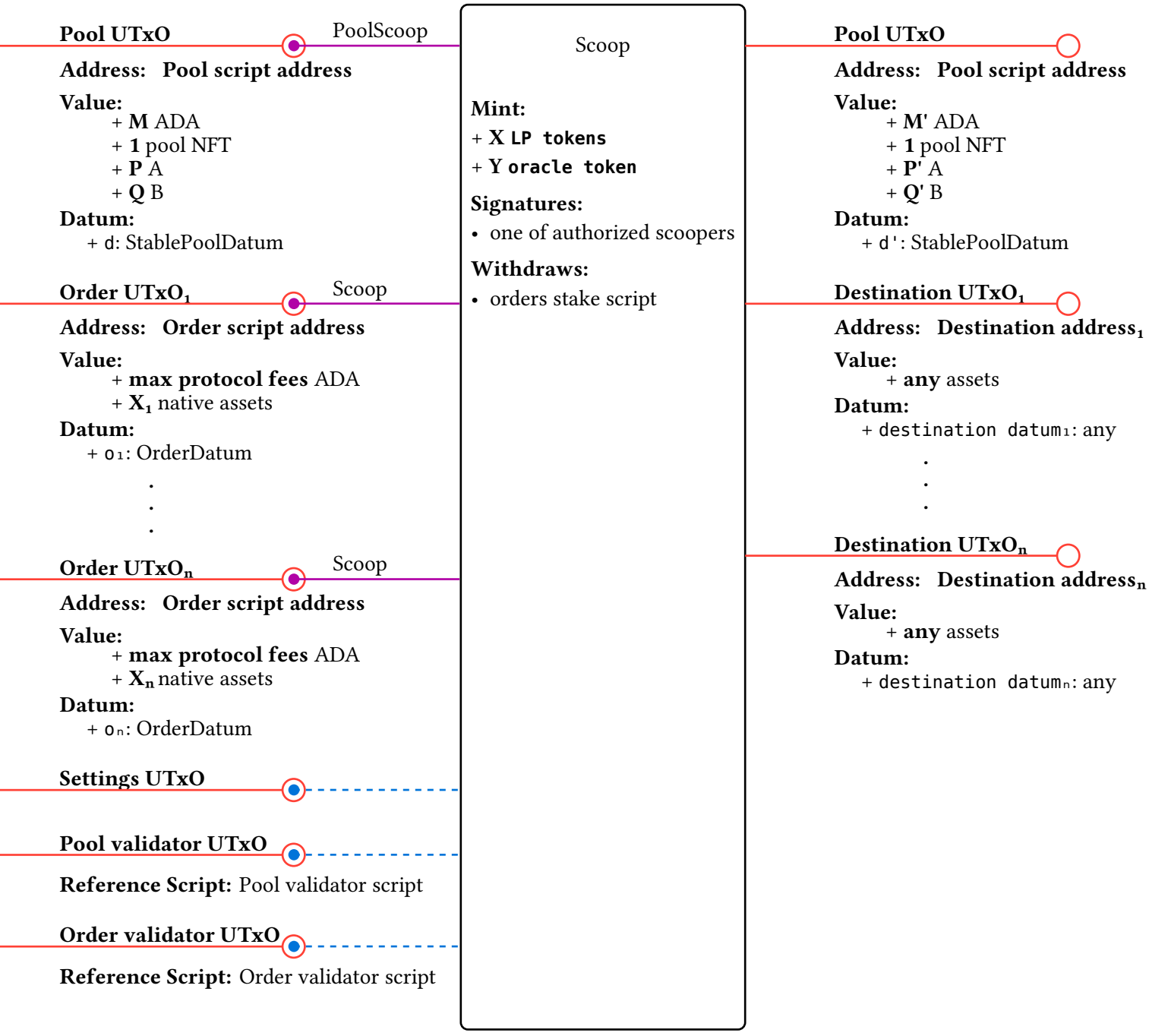
Note:

```
o: OrderDatum = {
  pool_ident: Option<ByteArray>
  owner: MultisigScript
  max_protocol_fee: Int
  destination: Destination
  details: AnyOf { Strategy, Swap, Deposit, Withdrawal, Record }
  extension: Data
}
```

Figure 5: Create Order transaction

2.d.c - Pools and Orders

2.d.c.a - Scoop



Note:

$$\text{process}(M, P, Q, d) = (M', P', Q', d')$$

Figure 6: Scoop transaction

3 - Launch Details

In this section we list the parameters used for the Stableswap mainnet launch of the audited script.

3.a - Settings validator

Parameters	
Name	**Value**
protocol_boot_utxo	`382b27b28c70343161f9abebdab78264e0fd7271baf3bb88ca04b52e5f0067ef#1`

Script Hash	`6d9d7acac59a4469ec52bb207106167c5cbfa689008ffa6ee92acc50`

3.b - Manage validator

Parameters	
Name	**Value**
settings_policy_id	`6d9d7acac59a4469ec52bb207106167c5cbfa689008ffa6ee92acc50`

Script Hash	`907c341fbc305a170dbdc9d322d4420939b6b402fdeef468d5696761`

3.c - Pool validator

Parameters	
Name	**Value**
manage_stake_script_hash	`907c341fbc305a170dbdc9d322d4420939b6b402fdeef468d5696761`
settings_policy_id	`6d9d7acac59a4469ec52bb207106167c5cbfa689008ffa6ee92acc50`

Script Hash	`4de79a0c17180030bff4c36825cb6e99caa007bc632f789561a26d56`

3.d - Order Stake validator

Parameters	
Name	**Value**
pool_script_hash	`4de79a0c17180030bff4c36825cb6e99caa007bc632f789561a26d56`

Script Hash	`598e5522eeb4faef80158a5c1d47ec1f1eaf7750b538dc3110a1cf64`

3.e - Order validator

Parameters	
Name	**Value**
stake_script_hash	`598e5522eeb4faef80158a5c1d47ec1f1eaf7750b538dc3110a1cf64`

Script Hash	`6ab62945d0d8d6288e243b3b6437ff9c099a38e088288f5a6b7c5e8b`

4 - Findings

ID	Title	Severity	Status
SST-001	Stableswap invariant not checked at pool creation	Critical	Resolved
SST-101	Initial LP token minting not following Curve's model	Major	Resolved
SST-102	`WithdrawFees`: bad calculation of treasury output B asset amount	Major	Resolved
SST-201	LP and protocol fees: rounding error at calculation	Minor	Resolved
SST-202	`UpdatePoolFees` and `AdjustLinearAmplification`: Pool value check might prevent multisig script updates	Minor	Resolved
SST-203	Prevent inclusion of reference scripts	Minor	Resolved
SST-204	Fee management: `protocol_fee_basis_points` not validated	Minor	Resolved
SST-301	No tests for swap with `protocol_fee_rate > 0`	Info	Resolved
SST-302	Definitions of different relevance mixed together	Info	Resolved
SST-303	Constant `precision` affects modularization and understandability	Info	Resolved
SST-304	Implicit LP fees in swap validation	Info	Resolved
SST-305	Counterintuitive ordering and missing documentation in `stableswap.ak`	Info	Resolved

SST-306	Redundant check for `circulating_lp` at pool creation	Info	Resolved

5 - SST-001 Stableswap invariant not checked at pool creation

Category	Commit	Severity	Status
Bug	40843b7fc4e926c1c40f8c32997d7a46e284e7fe	Critical	Resolved

5.a - Description

There are no checks on the `sum_invariant` and `linear_amplification` fields in the datum when creating a pool. As a result, it is possible to create a pool that does not satisfy the Stableswap invariant. However, it is an essential condition that the pool is always in a state that satisfies the invariant.

5.b - Recommendation

Check that the Stableswap invariant holds for pool's initial state at creation:

```
stableswap.liquidity_invariant(
    reserve_a,
    reserve_b,
    pool_output_datum.linear_amplification,
    pool_output_datum.sum_invariant,
)
```

5.c - Resolution

Resolved in commit `93fb5ce06e431de9d516d5d53333c086c3eeeea1`.

6 - SST-101 Initial LP token minting not following Curve's model

Category	Commit	Severity	Status
Bug	40843b7fc4e926c1c40f8c32997d7a46e284e7fe	Major	Resolved

6.a - Description

According to Curve's implementation, the amount of LP tokens minted at the first deposit is equal to the resulting sum invariant for the deposited liquidity.[3]

However, the Sundae Stableswap implementation still follows the constant product AMM model, where the minted amount of LP tokens is `sqrt(reserve_a * reserve_b)`.[4]

6.b - Recommendation

Modify the check for initial LP tokens (`initial_lq`) to match the sum invariant (`sum_invariant`). As done with liquidity, a change in precision can be applied by dividing the sum invariant by a constant.

6.c - Resolution

Resolved in commit `135a1cae9dfcc1e903e8c9a89e705f69dc528b50`.

[3] https://github.com/curvefi/curve-stablecoin/blob/762968ea930b7392f0fd174354a98840704fbef9/contracts/Stableswap.vy#L660

[4] https://github.com/SundaeSwap-finance/sundae-contracts/blob/40843b7fc4e926c1c40f8c32997d7a46e284e7fe/validators/pool.ak#L446

7 - SST-102 `WithdrawFees`: bad calculation of treasury output B asset amount

Category	Commit	Severity	Status
Bug	40843b7fc4e926c1c40f8c32997d7a46e284e7fe	Major	Resolved

7.a - Description

In `WithdrawFees` redeemer, the amount of B asset to be sent to the treasury is bad calculated:

```
let to_treasury_a = amount.2nd - allowance_a
let to_treasury_b = amount.2nd - allowance_b
```

where `to_treasury_b` should be

```
let to_treasury_b = amount.3rd - allowance_b
```

This could produce a potentially dangerous mismatch between the pool value and the pool datum.

7.b - Recommendation

Modify the calculation of `to_treasury_b` to match the correct amount:

```
let to_treasury_b = amount.3rd - allowance_b
```

7.c - Resolution

Resolved in commit `fe8774e22538089df2457f4569f4fb3fb0c6ca28`.

8 - SST-201 LP and protocol fees: rounding error at calculation

Category	Commit	Severity	Status
Bug	c8056d16c20c933783cf4d8c8cdac0c2e8cb816d	Minor	Resolved

8.a - Description

In swap operations, total fees are divided into LP fees and protocol fees. Both are computed as the relative rate of total fees using standard integer division:

```
lp_fees = total_fees * lp_fee_rate / total_fee_rate
```

```
protocol_fees = total_fees * lp_fee_rate / total_fee_rate
```

As both divisions are rounded down, `lp_fees + protocol_fees` and `total_fees` may differ by 1.

8.b - Recommendation

If the intention is to round in favour of the liquidity providers, use rounded up integer division, as follows:

```
lp_fees = (total_fees * lp_fee_rate + total_fee_rate - 1) / total_fee_rate
```

8.c - Resolution

Resolved in commit `45523e693ebe628a5e6bd5c668fe49b6219d6bbd`.

9 - SST-202 `UpdatePoolFees` and `AdjustLinearAmplification`: Pool value check might prevent multisig script updates

Category	Commit	Severity	Status
Bug	29e1adb3f7ace2c764ff97eb3fd8eb9e9c8c9a66	Minor	Resolved

9.a - Description

In `UpdatePoolFees` redeemer and `AdjustLinearAmplification` redeemers, if the pool input contains just min ADA, the manager multisig script could not be updated to be bigger (for example, adding a new signature requirement) because of the `pool_output_value == pool_input.value` check.

9.b - Recommendation

By allowing to add ADA to the pool UTxO, the datum size can vary.

9.c - Resolution

Resolved in commit `6ffa0b61763bb5e6d193d75cdd2065de267aabb2`.

10 - SST-203 Prevent inclusion of reference scripts

Category	Commit	Severity	Status
Improve-ment	40843b7fc4e926c1c40f8c32997d7a46e284e7fe	Minor	Resolved

10.a - Description

With the addition of the `minFeeRefScriptsCoinsPerByte` protocol parameter in the Voltaire era, including a reference script in any input (whether it's a reference or not) will impact the transaction fees, regardless of whether the script is executed. Given that the reference script field is not validated in any script output of the protocol, there's an attack vector where a malicious party includes a huge reference script in every output of a transaction, costing more fees to the next party interacting with those UTxOs.

10.b - Recommendation

Ensure that any script UTxO belonging to the protocol does not include a reference script.

10.c - Resolution

Resolved in commit `27faf78e6a84bdaa0e854571d2dc39ed5b976b83`.

11 - SST-204 Fee management: `protocol_fee_basis_points` not validated

Category	Commit	Severity	Status
Robustness	40843b7fc4e926c1c40f8c32997d7a46e284e7fe	Minor	Resolved

11.a - Description

All kinds of fees involved in the protocol are validated using the `fees_in_legal_range` function for constraining it to be between 0 and 10000 i.e. between 0% and 100%, except for `protocol_fee_basis_points`. For updating it in `WithdrawFees` redeemer, as well as when defining it at the Settings UTxO creation.

11.b - Recommendation

Validate the protocol fees using the `fees_in_legal_range` function.

11.c - Resolution

Resolved in commit `9170636e2347636fe651605b6536091f23710748`.

12 - SST-301 No tests for swap with `protocol_fee_rate > 0`

Category	Commit	Severity	Status
Testing	c8056d16c20c933783cf4d8c8cdac0c2e8cb816d	Info	Resolved

12.a - Description

All current tests use `protocol_fee_rate = 0`. Including tests with `protocol_fee_rate > 0` is particularly interesting for testing rounding errors when splitting fees. This is especially relevant to the issue described in finding SST-201.

12.b - Recommendation

Add tests with protocol_fee_rate > 0.

12.c - Resolution

Resolved in commit `e66a090afaa8f1d89ed66e46fe13781c8f92b976`.

13 - SST-302 Definitions of different relevance mixed together

Category	Commit	Severity	Status
Coding style	40843b7fc4e926c1c40f8c32997d7a46e284e7fe	Info	Resolved

13.a - Description

Some definitions are only used as auxiliaries for specific checks, but are mixed in with other, more relevant definitions. This is the case for the definition of `calculated_fees` in `do_swap()`. This definition is placed at the same level as more relevant definitions such as `swap_result` and `total_fees`, making the code harder to understand.

13.b - Recommendation

Introduce local scopes for auxiliary definitions. For instance, for `calculated_fees` do:

```
expect {
  let calculated_fees = ( swap_result * total_fee_rate + 9999 ) / 10000
  (calculated_fees == total_fees)?
}
```

13.c - Resolution

Resolved in commit `31c381a60895232a224a431b6ae4a41bd72f3820`.

14 - SST-303 Constant `precision` affects modularization and understandability

Category	Commit	Severity	Status
Design	40843b7fc4e926c1c40f8c32997d7a46e284e7fe	Info	Resolved

14.a - Description

A constant `precision` is defined in module `stableswap.ak` where the invariant checking functions are defined. It is used locally as a multiplier for pool liquidity parameters `new_pool_gives`, `new_pool_takes` and `old_pool_takes` in order to increase the precision of the calculations. Other parameters such as `raw_swap_result`, `old_sum_invariant` and `new_sum_invariant` are assumed to be already given in high precision. The introduction of the precision parameter, along with the use of parameters in different scales, makes the invariant functions harder to understand.

The `precision` constant is also imported and used in `stableswap.ak` to convert the high precision `raw_swap_result` into the actual result in token units `swap_result`. This introduces a subtle and non-intuitive coupling between the `stableswap.ak` and `swap.ak` modules.

14.b - Recommendation

Remove all references to `precision` from `stableswap.ak`, reverting to the previous, cleaner implementations of the invariant checking functions.

Instead, define and use the `precision` constant within `swap.ak`, converting all liquidities to high precision before passing them to the invariant checking functions.

14.c - Resolution

Resolved in commit `d7782b23686d0acc4fc79165a0c0a6f78eb37520`.

15 - SST-304 Implicit LP fees in swap validation

Category	Commit	Severity	Status
Coding style	40843b7fc4e926c1c40f8c32997d7a46e284e7fe	Info	Resolved

15.a - Description

In the validation logic for the swap operation, LP fees are not explicitly mentioned. Instead, they are only implicitly present as an amount that is not deducted from the pool liquidity.

However, LP fees are a key part of the business logic: The fact that they are added back as new liquidity explains the need to compute the new sum invariant off-chain and verify it on-chain.

15.b - Recommendation

Explicitly define the LP fees:

```
lp_fees = total_fees - protocol_fees
```

and use it in the expression that represents the final pool liquidity:

```
pool_quantity_b - swap_result + lp_fees
```

This way, the `pool_deduction` definition can be removed.

15.c - Resolution

Resolved in commit `5c474dc15155e93a51a6fed57392950869f0542e`.

16 - SST-305 Counterintuitive ordering and missing documentation in `stableswap.ak`

Category	Commit	Severity	Status
Documentation	40843b7fc4e926c1c40f8c32997d7a46e284e7fe	Info	Resolved

16.a - Description

Module `stableswap.ak` contains the implementation of the main validation functions related to the preservation of the Stableswap invariant.

In the case of the `swap_invariant` function, two Stableswap invariants are checked: one for the correct calculation of the swap result (`exchange_invariant`) and the other one for the correct update of the sum invariant after the addition of the LP fees (`liquidity_invariant`). As explained in Section 2.a.c, there is a logical precedence of `exchange_invariant` over `liquidity_invariant`. However, in the code, the calls appear in the reverse order, which we find counterintuitive.

Despite the importance of this module, all functions lack documentation. In particular, in `swap_invariant` it is difficult to understand the purpose of calling two different invariant-checking functions.

16.b - Recommendation

In `swap_invariant`, reverse the ordering of the `and` expression, calling first `exchange_invariant` and second `liquidity_invariant`.

Add inline documentation for all the defined validation functions. Section 2.a.d can be used as a resource to document `exchange_invariant` and `liquidity_invariant`.

16.c - Resolution

Resolved in commit `0311dac8056e7b44d8017b90a4d5e77af763b553`.

17 - SST-306 Redundant check for `circulating_lp` at pool creation

Category	Commit	Severity	Status
Redundancy	40843b7fc4e926c1c40f8c32997d7a46e284e7fe	Info	Resolved

17.a - Description

At pool creation, the `circulating_lp` datum field has the following check:[5]

```
pool_output_datum.circulating_lp == initial_lq
```

However, `initial_lq` is previously defined exactly as `pool_output_datum.circulating_lp`.

17.b - Recommendation

Remove the redundant check.

17.c - Resolution

Resolved in commit `cffda02c92b7b5a77c65fd8c36af25f500a8f43b`.

[5] https://github.com/SundaeSwap-finance/sundae-contracts/blob/40843b7fc4e926c1c40f8c32997d7a46e284e7fe/validators/pool.ak#L513

18 - Minor issues

In this section we list some issues we found that do not qualify as findings such as typos, coding style, naming, etc. We used the Github issues system to report them.

- typo in inline doc: “intitially” -> “initiallly” #88
- use `is_sqrt` from Aiken stdlib #89
- typo in inline doc: “protofol” -> “protocol” #93

A Appendix

A.1 Terms and Conditions of the Commercial Agreement

A.1.1 Confidentiality

Both parties agree, within a framework of trust, to discretion and confidentiality in handling the business. This report cannot be shared, referred to, altered, or relied upon by any third party without Txpipe LLC, 651 N Broad St, Suite 201, Middletown registered at the county of New Castle, written consent.

The violation of the aforementioned, as stated supra, shall empower TxPipe to pursue all of its rights and claims in accordance with the provisions outlined in Title 6, Subtitle 2, Chapter 20 of the Delaware Code titled "Trade Secrets,", and to also invoke any other applicable law that protects or upholds these rights.

Therefore, in the event of any harm inflicted upon the company's reputation or resulting from the misappropriation of trade secrets, the company hereby reserves the right to initiate legal action against the contractor for the actual losses incurred due to misappropriation, as well as for any unjust enrichment resulting from misappropriation that has not been accounted for in the calculation of actual losses.

A.1.2 Service Extension and Details

This report does not endorse or disapprove any specific project, team, code, technology, asset or similar. It provides no warranty or guarantee about the quality or nature of the technology/code analyzed.

This agreement does not authorize the client Sundae Labs to make use of the logo, name, or any other unauthorized reference to Txpipe LLC, except upon express authorization from the company.

TxPipe LLC shall not be liable for any use or damages suffered by the client or third-party agents, nor for any damages caused by them to third parties. The sole purpose of this commercial agreement is the delivery of what has been agreed upon. The company shall be exempt from any matters not expressly covered within the contract, with the client bearing sole responsibility for any uses or damages that may arise.

Any claims against the company under the aforementioned terms shall be dismissed, and the client may be held accountable for damages to reputation or costs resulting from non-compliance with the aforementioned provisions. **This report provides general information and is not intended to constitute financial, investment, tax, legal, regulatory, or any other form of advice.**

Any conflict or controversy arising under this commercial agreement or subsequent agreements shall be resolved in good faith between the parties. If such negotiations do not result in a conventional agreement, the parties agree to submit disputes to the courts of Delaware and to the laws of that jurisdiction under the powers conferred by the Delaware Code, TITLE 6, SUBTITLE I, ARTICLE 1, Part 3 § 1-301. and Title 6, SUBTITLE II, chapter 27 §2708.

A.1.3 Disclaimer

The audit constitutes a comprehensive examination and assessment as of the date of report submission. The company expressly disclaims any certification or endorsement regarding the subsequent performance, effectiveness, or efficiency of the contracted entity, post-report delivery, whether resulting from modification, alteration, malfeasance, or negligence by any third party external to the company.

The company explicitly disclaims any responsibility for reviewing or certifying transactions occurring between the client and third parties, including the purchase or sale of products and services.

This report is strictly provided for ***informational purposes*** and reflects solely the due diligence conducted on the following files and their corresponding hashes using sha256 algorithm:

Filename: `./validators/stake.ak` **Hash**: `5516d2eb35bc222c04d09707267e84b59656bde3137095b56cb2057cab40363d`
Filename: `./validators/settings.ak` **Hash**: `e8d5872e3db10ddc3395499c0ba61167d3e6197b088dfadc05e6109fb319eb30`
Filename: `./validators/pool.ak` **Hash**: `c0831beaa9c0a2478ceede43088e29a62b12d2472d96635b48eb09dac8208cdf`
Filename: `./validators/oracle.ak` **Hash**: `e1b2251f338972aac20466b4c9240b82499b71f23fac0c4fed0713b0ade27be6`
Filename: `./validators/pool_stake.ak` **Hash**: `883369843f31209563baa12c896a3030ef29edcb4525dd7417f037a0dd3c4c3d`
Filename: `./validators/order.ak` **Hash**: `1018f9fe564ffbdc4e2847fb26dd5db1bf86c11ffafad29fb41d6a4ee7fc8f74`
Filename: `./lib/calculation/record.ak` **Hash**: `de07da01fe07f757f629642617412ada06efdafc837722ab425b4035ac40dac4`
Filename: `./lib/calculation/swap.ak` **Hash**: `4c5ece9330a8a4f2d25c9127b9133183e4c972579de898581fe61045f97e31aa`
Filename: `./lib/calculation/strategy.ak` **Hash**: `98f12214d0327e8e5efca7bb99a1f9c345100259bf714773b4bc8b08c440e1b4`
Filename: `./lib/calculation/process.ak` **Hash**: `57637bcd5ae0aa28441d66013a7914f00dcac3bf6d87b394cce71721bfcc705f`
Filename: `./lib/calculation/stableswap.ak` **Hash**: `23a59881d749c7e19789958ca5c6bb83af44b6cf58c9d6733c6b32aef4747d68`
Filename: `./lib/calculation/withdrawal.ak` **Hash**: `c532183de59ee7daac5ffdf0e7117f13969d06e1cb71ced82ae43bbe600e5279`
Filename: `./lib/calculation/deposit.ak` **Hash**: `6e4de2c76d08f1d17ea0d3d6870ddccc010888f04a0e416b1c012bc911f4a302`
Filename: `./lib/calculation/shared.ak` **Hash**: `7bc92b11235726f883dfab1edda9af5a71932f57d63608db05bafbcef3b118dd`
Filename: `./lib/types/settings.ak` **Hash**: `19511ff82d7e887b91c4547af8f193c50c6d7b08f5b58a32b243bf8c51edae7d`
Filename: `./lib/types/pool.ak` **Hash**: `6874f996dc1d16a6531bf8e7fa750b53a7ad8a7233b889159368399af52d73d5`
Filename: `./lib/types/oracle.ak` **Hash**: `e1b2251f338972aac20466b4c9240b82499b71f23fac0c4fed0713b0ade27be6`
Filename: `./lib/types/order.ak` **Hash**: `44c5d13e51a3560e4a5a734fdb11ea97674b2790dd8e248368b2e943682b3cb3`
Filename: `./lib/shared.ak` **Hash**: `7a9aa62b0b546adce3660b6a726bce687e6cce4d6cd0d1504e7c718371aeb740`

TxPipe advocates for the implementation of multiple independent audits, a publicly accessible bug bounty program, and continuous security auditing and monitoring. Despite the diligent manual review processes, the potential for errors exists. TxPipe strongly advises seeking multiple independent opinions on critical matters. It is the firm belief of TxPipe that every entity and individual is responsible for conducting their own due diligence and maintaining ongoing security measures.

A.2 Issue Guide

A.2.1 Severity

Severity	Description
Critical	Critical issues highlight exploits, bugs, loss of funds, or other vulnerabilities that prevent the dApp from working as intended. These issues have no workaround.
Major	Major issues highlight exploits, bugs, or other vulnerabilities that cause unexpected transaction failures or may be used to trick general users of the dApp. dApps with Major issues may still be functional.
Minor	Minor issues highlight edge cases where a user can purposefully use the dApp in a non-incentivized way and often lead to a disadvantage for the user.
Info	Info are not issues. These are just pieces of information that are beneficial to the dApp creator. These are not necessarily acted on or have a resolution, they are logged for the completeness of the audit.

A.2.2 Status

Status	Description
Resolved	Issues that have been **fixed** by the **project** team.
Acknowledged	Issues that have been **acknowledged** or **partially fixed** by the **project** team. Projects can decide to not **fix** issues for whatever reason.
Identified	Issues that have been **identified** by the **audit** team. These are waiting for a response from the **project** team.

A.3 Revisions

This report was created using a git based workflow. All changes are tracked in a github repo and the report is produced using typst. The report source is available here. All versions with downloadable PDFs can be found on the releases page.

A.4 About Us

TxPipe is a blockchain technology company responsible for many projects that are now a critical part of the Cardano ecosystem. Our team built Oura, Scrolls, Pallas, Demeter, and we're the original home of Aiken. We're passionate about making tools that make it easier to build on Cardano. We believe that blockchain adoption can be accelerated by improving developer experience. We develop blockchain tools, leveraging the open-source community and its methodologies.

A.4.1 Links

- Website
- Email
- Twitter