Splash StablePool protocol: Formal Spec

March 7, 2024

1 Model

Let n denote a number of distinct stablecoins $n, n \in N \setminus \{0, 1\}$.
Let NFT denote a unique non-fungible identifier of the pool.
Let $\boldsymbol{a} = \{a_k\}_{k=1}^{n}$ denote a set of distinct stablecoins.
Let $P_{cp} = P^{a_1 \otimes a_2 \otimes \cdots \otimes a_n}$ denote a unique pool comprised of assets $\boldsymbol{a}$.
Let P_i denote an i'th state of a unique liquidity pool P.
Let l denote an asset representing the liquidity provider's share of the P.
Let $d^{a_1 \oplus a_2 \oplus \cdots \oplus a_n}$ denote an integer multiplier for stablecoins. Multipliers are calculated as the ratio of the maximum value of decimals from set $\boldsymbol{a}$ to the decimals of the stablecoin $a_i \in \boldsymbol{a}$.
Let $M^{a_1 \oplus a_2 \oplus \cdots \oplus a_n}$ denote an integer amount of any asset from $\boldsymbol{a}$.
Let $T^{a_1 \oplus a_2 \oplus \cdots \oplus a_n}$denote an integer amount of treasury any asset from $\boldsymbol{a}$.
Let Y^l denote supply of the liquidity token l.
Let $C = P^{num} \otimes L^{num} \otimes F^{den}$ denote pool fee configuration, where P^{num} - integer value of the protocol fee numerator, L^{num} - integer value of the liquidity provider fee numerator, F^{den} - integer value of pool fees denominator.
Let A denote an integer value of the amplification coefficient in the StableSwap invariant.
Let Z denote script credential of treasury script address.
Let D denote StablePool-proxy DAO script witness.
Let V denote Splash DAO voting script witness.
Let I denote an integer value of the StableSwap invariant (This value can be calculated from the pool's state parameters, but since we make calculations off-chain we store it as a separate pool's parameter, which can be invalid and thus affects the state).

Using the notations above a unique StablePool can be defined as:

$$\boxed{P = \quad M_{cp} \otimes T_{cp} \otimes d_{cp} \otimes Y^l \otimes C \otimes A \otimes Z \otimes D \otimes I}$$

where $M_{cp} = M^{a_1} \otimes M^{a_2} \otimes \cdots \otimes M^{a_n}$, $T_{cp} = T^{a_1} \otimes T^{a_2} \otimes \cdots \otimes T^{a_n}$, $d_{cp} = d^{a_1} \otimes d^{a_2} \otimes \cdots \otimes d^{a_n}$.

Lets now define sum of possible state transitions allowed to be applied to P. We will highlight 2 main categories:

- AMM-actions $A = D \otimes R \otimes S$, where:
  - $D = M_{cp} \otimes Y^l$ - deposit liquidity action;
  - $R = Y^l \otimes M_{cp}$ - redeem liquidity action;
  - $S = M^{a_i \oplus a_j} \otimes M^{a_{i_1} \oplus a_{j_1}} \otimes \cdots \otimes M^{a_{i_m} \oplus a_{j_m}}, i_k \neq j_k \wedge m = C_n^2$ - swap action a_{i_k} to a_{j_k} and vise versa.
- DAO-actions $U \equiv \{U_{ulf}, U_{upf}, U_{wt}, U_{uta}, U_{upd}\}$, where:

– Update liquidity provider fee $U_{ulf} = t \otimes \sigma_D \otimes \sigma_V$, where t is a new parameter integer value and $\sigma_D = \mathbb{1}$, $\sigma_V = \mathbb{1}$ stand for validation results of D and V DAO-scripts respectively;

– Update protocol fee $U_{upf} = t \otimes \sigma_D \otimes \sigma_V$;

– Update treasury address $U_{uta} = z \otimes \sigma_D \otimes \sigma_V$ to new address z;

– Update StablePool-proxy DAO script witness $U_{upd} = d \otimes \sigma_D \otimes \sigma_V$ to new witness d;

– Withdrawal treasury and set a new value vector of treasury balances $\vec{t_u}$: $U_{wt} = \vec{t_u} \otimes \sigma_D \otimes \sigma_V$.

As a result, a complete set of state transitions that can be applied to the pool P is $\Omega = A \otimes U \otimes K$ and the state transition function can be defined as follows:

$$\boxed{apply: \quad P_i \to \Omega \to P_{i'} \oplus \mathbb{1}}$$

2 Security Assumptions

General preliminary assumptions are:

- **Pool creation**. The correctness of the creation of the pool is validated only off-chain, since adding on-chain validations will permanently affect the pool contract by increasing execution units (exUnits) and memory usage (exMemory), which is undesirable. All users can can independently verify the pool's creation accuracy due to the open-source nature of contracts and off-chain code.

- **Minting policy**. Minting policies validators for NFT and l tokens are already implemented and are out of the scope of the this protocol.

- **Orders**. There is no restriction on the types of orders the pool can execute, this is protocol feature.

2.1 Notations

It's useful to introduce some commonly used notations. First of all, we will denote all sets of length n in form of vectors, i.e. $P_i.\vec{d}$ is a vector of assets set $\boldsymbol{a}$ multiplier values and etc. Also, we will use Δ notations to represent difference between integer values of the same type, i.e. $\Delta Y^l = P_{i'}.Y^l - P^i.Y^l$ and etc.

- $\vec{M_{i'}^{tr}} = P_{i'}.\vec{M} - P_{i'}.\vec{T}$ - tradable balances (without protocol fees since we accumulate is inside the pool);

- $\vec{M_{i'}^{trn}} = P_{i'}.\vec{M} - P_i.\vec{T}$ - "native" tradable balances (without latest protocol fees);

- $\vec{M_{i'}^{trnl}} = P_{i'}.\vec{M} - P_i.\vec{T} - \vec{L_{i'}^{fee}}$ - tradable balances without latest protocol fees and liquidity provider fees $\vec{L_{i'}^{fee}}$, which aren't stored anywhere and are calculated on the fly;

- $Inv\_check(\vec{M}, I) \to 0|1$ - function to check the validity if the StableSwap invariant value I for the given balances M (it is assumed for simplicity that this function is parameterized by the parameters from the $conf$, also, it's implementation is a bit different for the deposit/redeem and swap actions checks, ee more details in the source code);

- I_n - "native" StableSwap invariant value, i.e. calculated for $\vec{M_i^{trn}}$ balances.

- I_{nl} - StableSwap invariant value without any fees applied to the latest action.

Values I_n and I_{nl} aren't stored on-chain, they only are passed as a parameters to the D or R action to confirm that all balances and fees for the corresponding state transition were obtained in accordance with the invariant calculation procedure.

There is also list of pre-defined constants that are the same for pools with any number of tradable assets n:

- E - total emission of Y^l tokens;
- imb_avg_den - averaging swap fee denominator.

Let us also introduce so-called "config set of parameters":

$$P_i.conf = \{P_i.n,\ P_i.NFT,\ P_i.d_{cp},\ P_i.C,\ P_i.A,\ P_i.Z,\ P_i.D\}.$$

2.2 Deposit and Redeem

To validate the correctness of the applied deposit/redeem state transition we must firstly perform the following steps:

1. Extract $(I_n, I_{nl}) \leftarrow D \oplus R$;
2. Calculate all necessary values:

$$\begin{aligned}
&\vec{M_{i'}^{tr_ideal}} = \vec{M_i^{tr}} \cdot I_n / P_i.I,\\
&\text{f_avg_den} = \text{imb_avg_den} \cdot (P_i.n - 1) \cdot P_i.F^{den},\\
&\Delta\vec{T} = |\vec{M_{i'}^{trn}} - \vec{M_{i'}^{tr_ideal}}| \cdot P_i.P^{num} \cdot P_i.n / \text{f_avg_den},\\
&\vec{L_{fee}} = |\vec{M_{i'}^{trn}} - \vec{M_{i'}^{tr_ideal}}| \cdot P_i.L^{num} \cdot P_i.n / \text{f_avg_den}.
\end{aligned}$$

In deposit/redeem actions we assume that $apply : P_i \to D \oplus R \to P_{i'} \oplus \mathbb{1}$ satisfies the following conditions:

$$\boxed{\begin{array}{rl}
P_{i'}.conf = & P_i.conf\\
P_{i'}.\vec{T} = & P_i.\vec{T} + \Delta\vec{T}\\
-\Delta Y^l = & ((I_{nl} - P_i.I) - (E - P_i.Y^l))/P_i.I\\
Inv_check(\vec{M_{i'}^{tr}} \cdot P_{i'}.\vec{d},\ P_{i'}.I) = & 1\\
Inv_check(\vec{M_{i'}^{trn}} \cdot P_{i'}.\vec{d},\ I_n) = & 1\\
Inv_check(\vec{M_{i'}^{trnl}} \cdot P_{i'}.\vec{d},\ I_{nl}) = & 1
\end{array}}$$

2.3 Swap

To validate the correctness of the applied deposit/redeem state transition we must firstly perform the following steps:

1. Extract $(b, q) \leftarrow S$, where b and q is base and quote asset indexes respectively;
2. Calculate all necessary values:

$$\begin{aligned}
&\text{f_num_rev} = (P_i.F^{den} - P_i.L^{num} - P_i.P^{num}),\\
&\Delta M_{tr}^k = (P_{i'}.M^k - P_{i'}.T^k) - (P_i.M^k - P_i.T^k),
\end{aligned}$$

We assume that $apply : P_i \to S \to P_{i'} \oplus \mathbb{1}$ satisfies the following conditions:

$$\boxed{\begin{array}{rl} P_{i'}.conf = & P_i.conf \\ |\Delta T^q \cdot \text{f\_num\_rev} + \Delta M^q \cdot P_i.P^{num}| \leq & \text{max\_t\_err} * P_i.F^{den} \\ |\Delta M^q_{tr} \cdot \text{f\_num\_rev} - \Delta M^q \cdot (P_i.F^{den} - P_i.L^{num})| \leq & \text{max\_t\_err} * P_i.F^{den} \\ \sum_{k=1,k\neq q}^{k=n} \Delta T^k = & 0 \\ \sum_{k=1,k\neq b \wedge k\neq q}^{k=n} \Delta M^k_{tr} = & 0 \\ \Delta M^b = & \Delta M^b_{tr} \\ \Delta M^q = & \Delta M^q_{tr} + \Delta T^q \\ Inv\_check(\vec{M^{tr}_{i'}} \cdot P_{i'}.\vec{d},\ P_{i'}.I) = & 1 \\ Inv\_check(\vec{M^{trnl}_{i'}} \cdot P_{i'}.\vec{d},\ P_i.I) = & 1 \end{array}}$$

2.4 DAO Actions

All DAO-actions is validated by the StablePool-proxy DAO script. We assume that since main pool contract is involved, proxy DAO script checks only validity of mutable pool datum fields $\{P^{num}, L^{num}, Z, D, \vec{T}\}$ updates.

2.4.1 Update liquidity provider fee

We assume that $apply : P_i \to U_{ulf} \to Pi' \oplus \mathbb{0}$ satisfies the following conditions:

$$\boxed{\begin{array}{rl} P_{i'}.\vec{T} = & P_i.\vec{T} \\ P_{i'}.\vec{M} = & P_i.\vec{M} \\ P_{i'}.Y^l = & P_i.Y^l \\ P_{i'}.I = & P_i.I \\ P_{i'}.L^{num} = & U_{ulf}.t \\ P_{i'}.P^{num} = & P_i.P^{num} \\ P_{i'}.Z = & P_i.Z \\ P_{i'}.D = & P_i.D \end{array}}$$

2.4.2 Update protocol fee

We assume that $apply : P_i \to U_{upf} \to Pi' \oplus \mathbb{0}$ satisfies the following conditions:

$$\boxed{\begin{array}{rl} P_{i'}.\vec{T} = & P_i.\vec{T} \\ P_{i'}.\vec{M} = & P_i.\vec{M} \\ P_{i'}.Y^l = & P_i.Y^l \\ P_{i'}.I = & P_i.I \\ P_{i'}.L^{num} = & P_i.L^{num} \\ P_{i'}.P^{num} = & U_{upf}.t \\ P_{i'}.Z = & P_i.Z \\ P_{i'}.D = & P_i.D \end{array}}$$

2.4.3 Update treasury address

We assume that $apply : P_i \to U_{uta} \to Pi' \oplus \mathbb{0}$ satisfies the following conditions:

$$\boxed{\begin{array}{rl} P_{i'}.\vec{T} = & P_i.\vec{T} \\ P_{i'}.\vec{M} = & P_i.\vec{M} \\ P_{i'}.Y^l = & P_i.Y^l \\ P_{i'}.I = & P_i.I \\ P_{i'}.L^{num} = & P_i.L^{num} \\ P_{i'}.P^{num} = & P_i.P^{num} \\ P_{i'}.Z = & U_{upd}.d \\ P_{i'}.D = & P_i.D \end{array}}$$

2.4.4 Update StablePool-proxy DAO script witness

We assume that $apply : P_i \to U_{upd} \to Pi' \oplus \mathbb{0}$ satisfies the following conditions:

$$\boxed{\begin{array}{rl} P_{i'}.\vec{T} = & P_i.\vec{T} \\ P_{i'}.\vec{M} = & P_i.\vec{M} \\ P_{i'}.Y^l = & P_i.Y^l \\ P_{i'}.I = & P_i.I \\ P_{i'}.L^{num} = & P_i.L^{num} \\ P_{i'}.P^{num} = & P_i.P^{num} \\ P_{i'}.Z = & P_i.Z \\ P_{i'}.D = & U_{ulf}.d \end{array}}$$

2.4.5 Withdrawal treasury

We assume that $apply : P_i \to U_{wt} \to Pi' \oplus \mathbb{0}$ satisfies the following conditions:

$$\boxed{\begin{array}{rl} P_{i'}.\vec{T} = & U_{wt}.\vec{t_u} \\ P_{i'}.\vec{M} = & P_i.\vec{M} - P_i.\vec{T} + U_{wt}.\vec{t_u} \\ P_{i'}.Y^l = & P_i.Y^l \\ P_{i'}.I = & P_i.I \\ P_{i'}.L^{num} = & P_i.L^{num} \\ P_{i'}.P^{num} = & P_i.P^{num} \\ P_{i'}.Z = & P_i.Z \\ P_{i'}.D = & P_i.D \end{array}}$$