Polymer.fi: Milestone 1 Report

Michael Lisano Alex McDowell Gabriel Arrouye

June 30, 2022

Abstract

We're excited to announce our first development milestone in creating Polymer.fi. This sprint was mostly focused on building out the core smart contracts to a state where they implement all of the features necessary to a functioning private DEX, successfully deploy to a local devnet, and pass all the unit tests we run on them.

Contents

1 Smart Contracts Summary

Polymer's smart contract architecture involves multiple contracts which handle various parts of the exchange. The vault contract acts as the main storage contract for user funds and handles all activities related to pool creation, liquidity providing, and trade execution. Unlike typical AMMs, with Polymer, pools are not individual contracts, but rather, they are represented by data structures that are stored within the main vault contract. The vault contract contains all logic for providing liquidity or swapping assets. This further reduces the gas costs associated with pool creation and trade execution. Polymer's balancer-inspired vault design allows for users to execute trades across multiple pools, while minimizing gas costs.

When a user preforms a swap, polymer's client-side router calculates the optimal swap path and submits the swap path to the `polymer-swap-router` contract. The router contract is responsible for executing a swap path and returning the swapped funds to the user. The router checks for slippage and reverts the transaction if the trade output does not match the user's minimum output amount.

When a trade is preformed, a 0.3% liquidity fee is taken. Of this fee, 0.2% is distributed to liquidty providers while 0.1% is distributed to polymer token holders. The `polymer-staking` contract allows users to earn trading fees by staking their polymer holdings. This contract is responsible for collecting and distributing trading fees, which are automatically swapped and paid to token holders in `SSCRT`. The contract keeps track of staked funds by recoridng each deposit and withdraw event within the contract's storage.

For liquidity providers, the `polymer-distributor` contract is responsible for handling liquidity mining rewards. This contract interacts with the vault in order to retrieve the balance of a liquidity provider and distribute rewards proportionally. The reward schedule can be set during initialization and linear, stepwise, and logarithmic, distribution models can be specified.

The `polymer-vesting` contract contains the logic for locking a specified amount of Polymer tokens and distributing them linearly over a given time period. The development team's share of the token supply will be vested across 4 years.

Finally, for assets bridged from Axelar, the `polymer-ibc-wrapper` contract handles wrapping Axelar assets into SNIP-20 compatible tokens than can be supported by the core exchange contracts.

A brief description of each contract is stated below:

- `polymer-vault`: The vault is the main contract responsible for storing user funds, managing liquidity, facilitating pool creation, preforming swaps, and keeping track of user balances.
- `polymer-distributor`: The distributor contract is responsible for managing liquidity mining rewards given to polymer LPs.
- `polymer-staking`: The staking contract allows users to stake their `PLMR` tokens to earn trading fees from the exchange.
- `polymer-vesting`: The vesitng contracts are responsible for linearly distributing the development team's tokens across 4 years.
- `polymer-swap-router`: The router contract is responsible for executing swap paths provided by the client-side order router.
- `polymer-token`: `PLMR` is a SNIP-20 compatible token contract that earns trading fees, paid in `SSCRT`.
- `polymer-ibc-wrapper`: The IBC wrapper contract is responsible for converting assets bridges from Axelar into SNIP-20 compatible tokens for use on the exchange.

2 Vault Contract

Link to Code on GitHub

2.1 Overview

The Polymer Vault is the most complex contract, and manages:

- Deposits
- Withdraws
- Swaps (supports chained swaps)
- Simulated swaps
- Weighted pool creation
- Stable pool creation
- Reading/writing LP events
- Reading/writing LP balances

Deposits, withdraws, and swaps are executed by sending tokens to the contract with the request encoded in the SNIP-20 transaction's private msg field. For both gas efficiency and security reasons, Polymer pools are created not by deploying individual pool contracts, but rather by instantiating hardcoded structs within the vault contract.

The vault also exposes readable user data endpoints, specificically LP deposit/withdraw history, as well as current LP balances (and by extension current LP portfolio balance). Adhering to Secret Network's privacy mantra, these endpoints are protected by user-revokable permit queries and only can be viewed by authorized users.

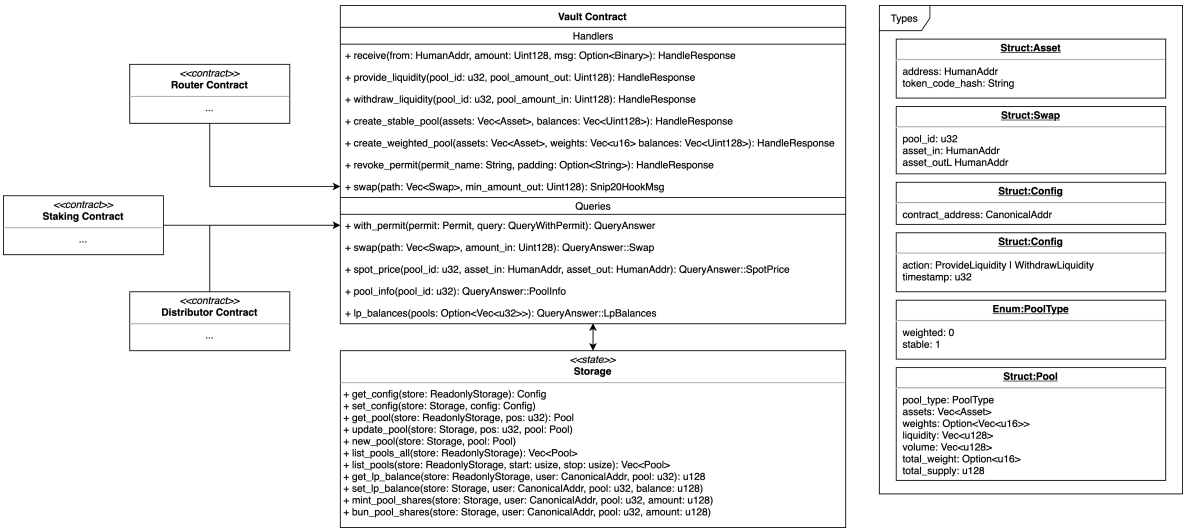


Figure 1: Smart Contract - Class diagram for the vault implementation.

Handle Methods

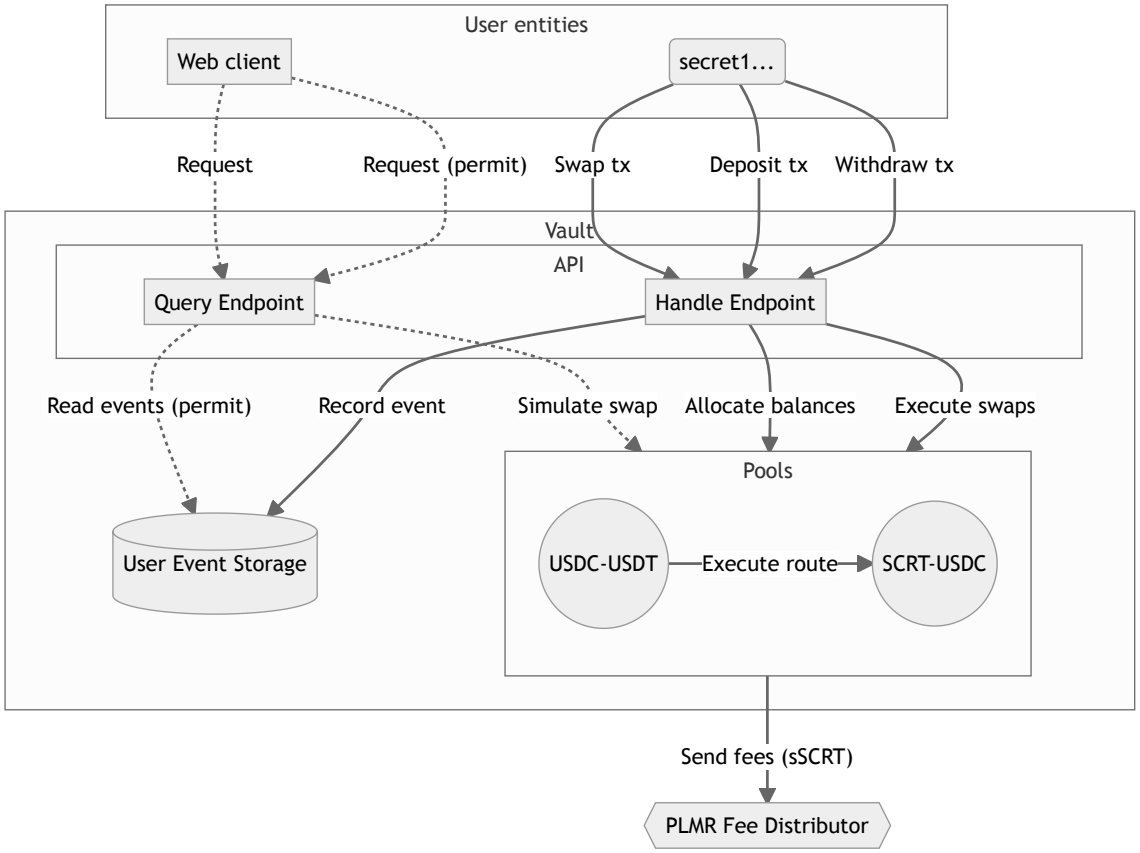
- *receive(from: HumanAddr, amount: Uint128, msg: Option):* Receive hook function called when a Snip20 token is sent to the contract. `msg` contains a `Snip20HookMsg::Swap`
- *provide_liquidity(pool_id: u32, pool_amount_out: Uint128):* Provides liquidity for a given `pool_id` and collects tokens from the user.
- *withdraw_liquidity(pool_id: u32, pool_amount_in: Uint128):* Withdraws liquidity for a given `pool_id` and returns tokens to the user.

- *create_stable_pool(assets: Vec, balances: Vec):* Creates a new stable pool given a list of assets and initial balances. Returns the `pool_id` of the new pool.
- *create_weighted_pool(assets: Vec, weights: Vec balances: Vec):* Creates a new weighted pool given a list of assets, weights, and initial asset balances. Returns the `pool_id` of the new pool.
- *revoke_permit(permit_name: String, padding: Option):* Revokes a query permit for a given user.
- *swap(path: Vec, min_amount_out: Uint128):* Receive hook collects assets from the user and executes the swaps.

Query Methods

- *with_permit(permit: Permit, query: QueryWithPermit):* Executes a given query using a query permit.
- *swap(path: Vec, amount_in: Uint128):* Simulates a swap path using a given amount of assets.
- *spot_price(pool_id: u32, asset_in: HumanAddr, asset_out: HumanAddr):* Returns the spot price of a given `pool_id`.
- *pool_info(pool_id: u32):* Returns the pool information for a given `pool_id`.
- *lp_balances(pools: Option<Vec>):* Returns the balance staked in a given pool. Requires a query permit.

2.2 Architecture



3 PLMR Token Distribution Contract

Link to Code on GitHub

3.1 Overview

The PLMR Token Distributor is a generic asset payout contract that, while central to Polymer's tokenomics, can actually be applied in a variety of use cases. Its primary functionality consists of holding a pool of any SNIP-20 token (PLMR in our case) and releasing it over a specified period of time.

3.2 Usage

The token release behavior is defined by the deployer, who must pass in a distribution schedule of the format `[timestamp1, timestamp2, amount][]`.

```
// Example distribution schedule: 3 increments of
// 3600 seconds, each releasing 1000 tokens
let distSchedule = [
    [1656578038, 1656581638, '10000000000000000000'],
    [1656581638, 1656585238, '10000000000000000000'],
    [1656585238, 1656588838, '10000000000000000000']
]
```

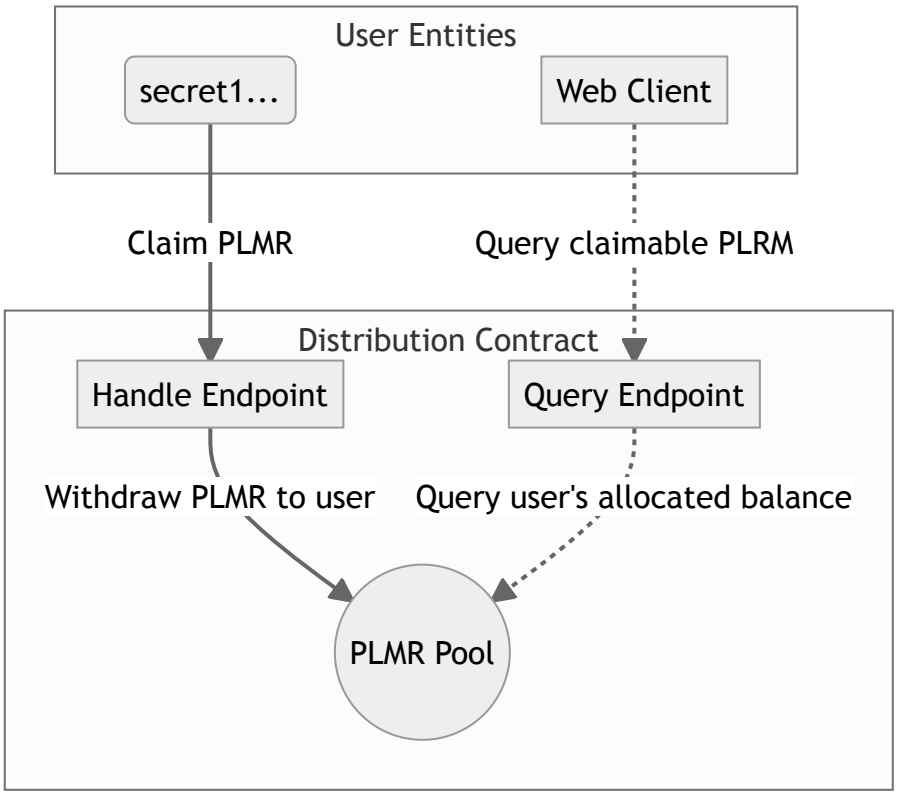
The process for iteratively generating such a distribution schedule is rather straightforward, especially in the case of a linear or exponential-decay distribution pattern. A simple example might be:

```
const release_decrement = 100;
const release_delay = 3600; // one hour
let tokens = 100000; // total token count we want to distribute
let sched = [];
let date = Math.floor(Date.now() / 1000) + release_delay; // start in 1 hour
while (tokens > 0) {
    sched.push([
        date,
        date + release_delay,
        toWei(release_decrement) // convert to token decimals
    ]);
    tokens -= release_decrement;
    date += release_delay
}
```

In our case, we will deploy individual PLMR distributor contracts that will linearly distribute PLMR to each of the following:

Entity	Allocated PLMR Supply	Distribution Period
Liquidity Providers	62%	TBD
Airdrop Recipients	5%	TBD
Community treasury multisig	10%	4 years

3.3 Architecture



3.4 UML Specification

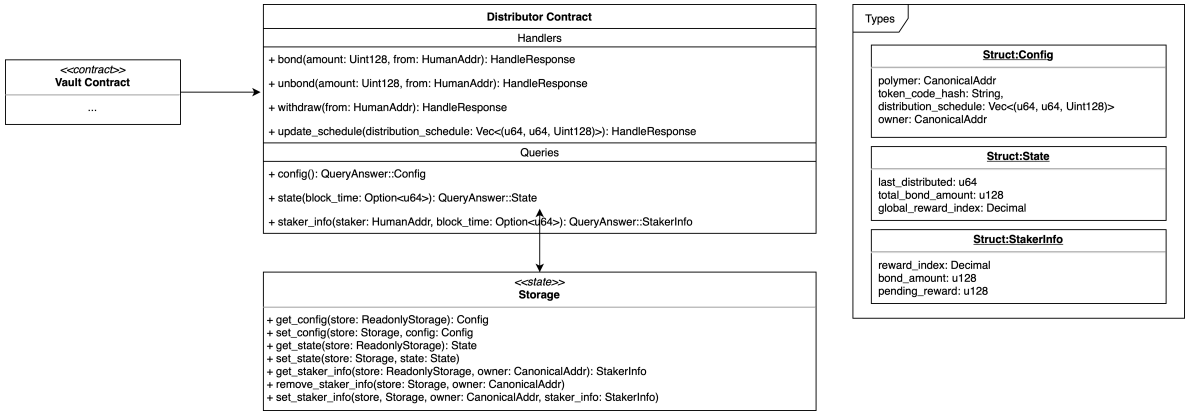


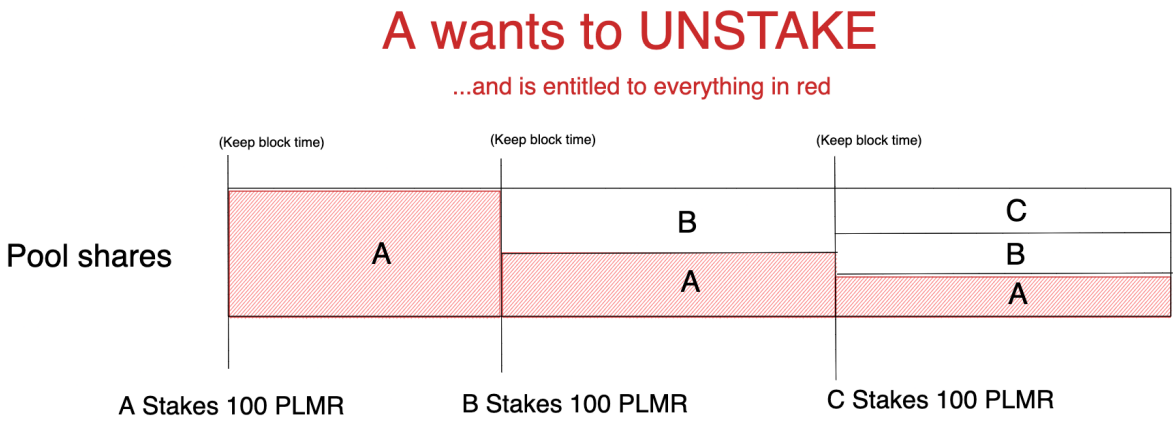
Figure 2: Smart Contract - Class diagram for the distributor implementation.

4 PLMR Reward Distribution Contract

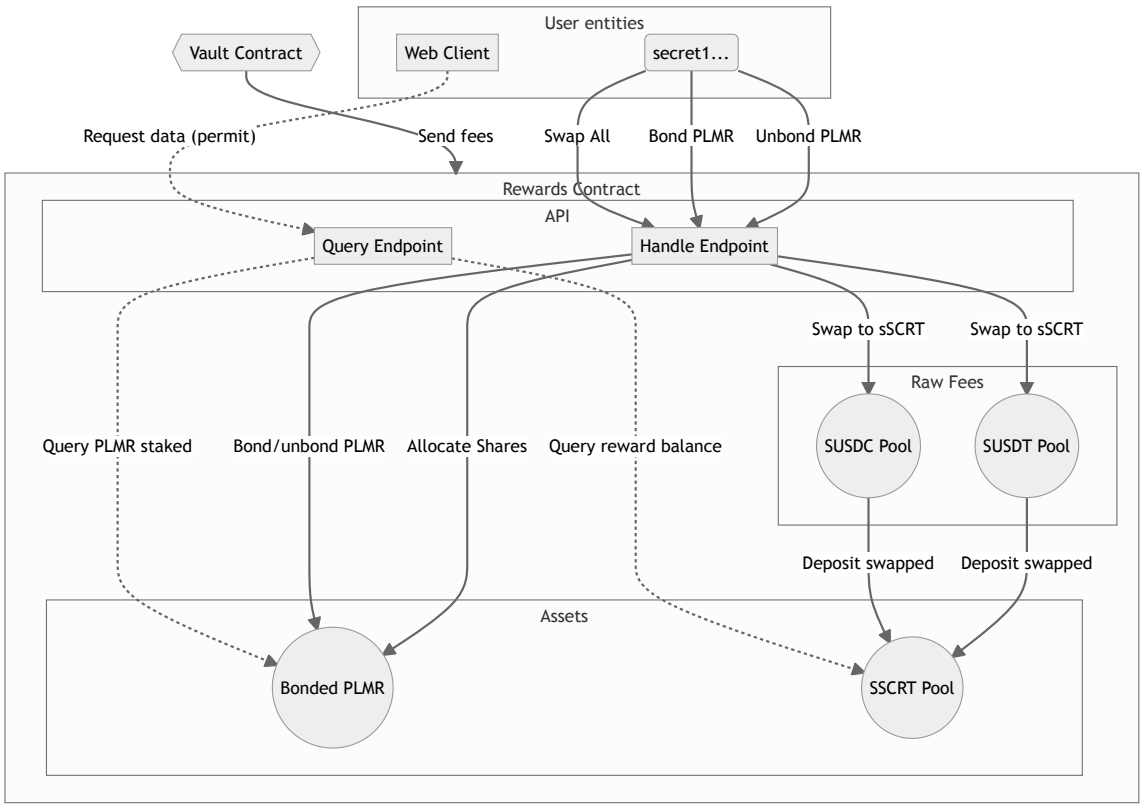
Link to Code on GitHub

4.1 Overview

PLMR holders are entitled to a proportional share of exchange fees, and must bond their PLMR to collect said fees. Unbonding a user's PLMR will claim all the fees the user is entitled to. In order to prevent newcomers from simply bonding and immediately withdrawing an instantly-allocated share of the pool, the rewards contract factors in how much a user is allowed to withdraw by logging every staking/unstaking event and adding/reducing a multiplier on every other bonded user's share of the pool, which results in every user, upon PLMR withdrawal, being entitled to their fair share of exchange fees *only while they had their PLMR bonded*. A simple illustration of this concept is included below.



4.2 Architecture



4.3 UML Specification

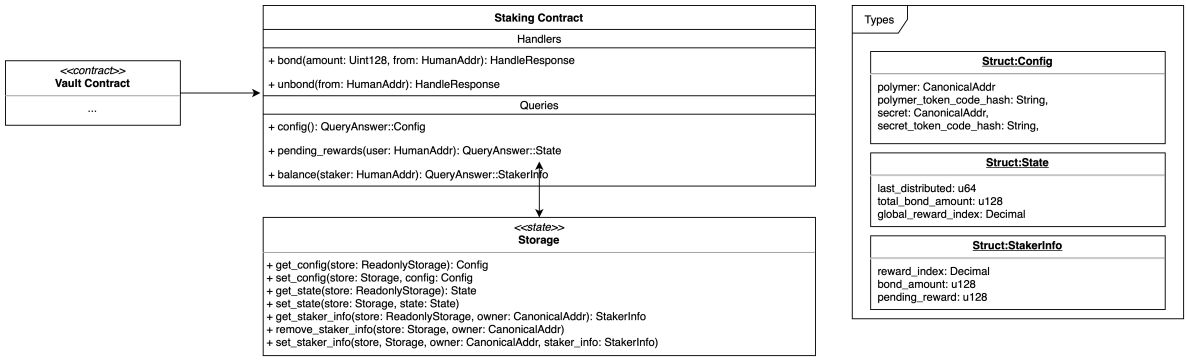


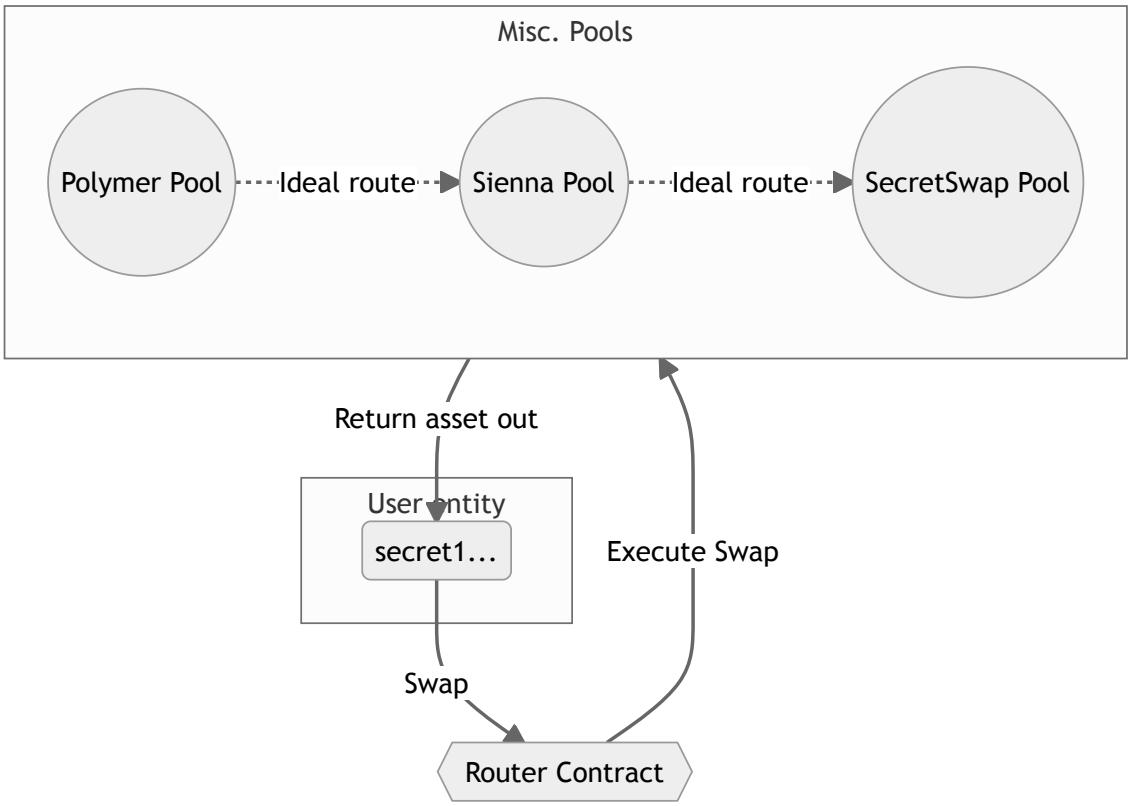
Figure 3: Smart Contract - Class diagram for the staking implementation.

5 Router Proxy Contract

5.1 Overview

While Polymer's vault is able to run batch swaps on its own internal pools, the exchange also includes a router proxy contract that routes trades between additional external liquidity sources. Calling this rather than the vault's built-in swap functionality allows for "interleaved" trades that hop across Polymer, SecretSwap, SiennaSwap. This will be especially helpful right after launch, as Polymer is still in the process of bootstrapping liquidity.

5.2 Architecture



5.3 UML Specification

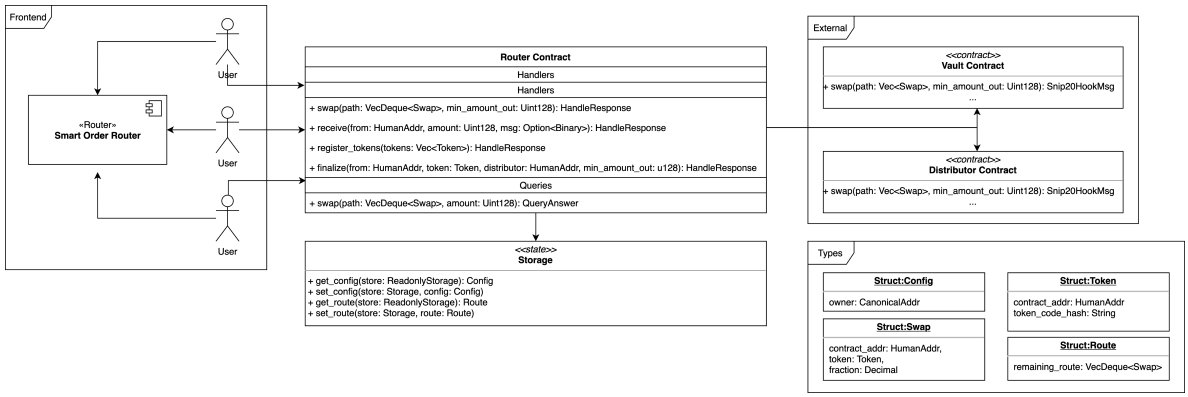


Figure 4: Smart Contract - Class diagram for the router implementation.

6 Vesting Schedule Contract

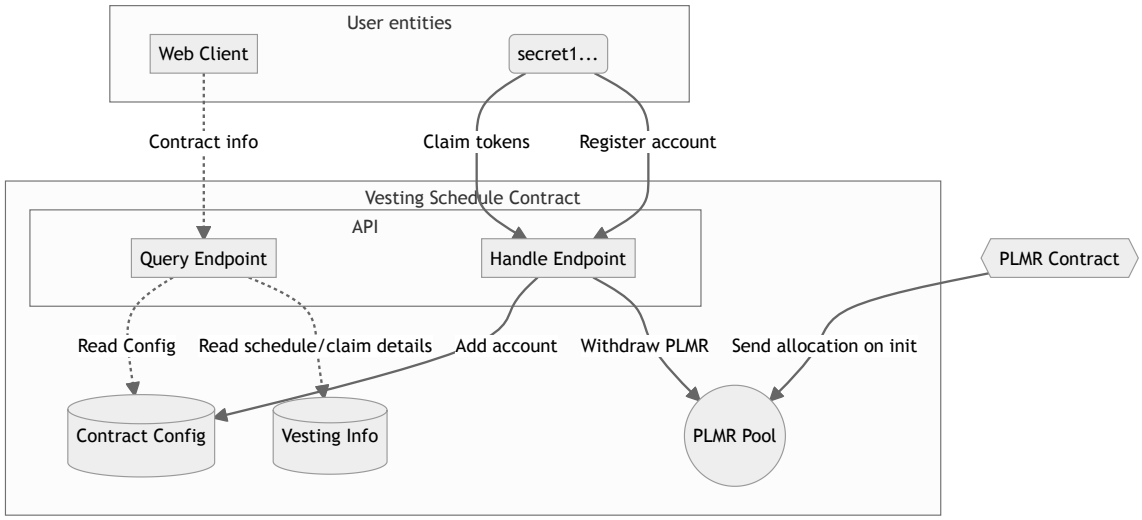
Link to Code on GitHub

6.1 Overview

The vesting schedule contract manages a pool of PLMR and distributes it linearly over a 4-year period to developers (for testing purposes, the payout time and nature of the function - e.g. linear, exponential decay - can be customized when deploying the contract). The list of developer wallets to pay out and a corresponding payout share for each wallet can be specified one time after launch by the owner account calling the `register_vesting_accounts` handlemsg. After this, developers can claim accumulated PLMR at any time by calling `claim`. The contract's codebase is based on Anchor Protocol's vesting schedule, and makes several modifications:

- Remove receive hooks and make the vesting schedule only callable by the vault contract
- Serialize stored data using `bincode2` to improve gas efficiency
- Store & use token codehash for faster querying
- Use query permits to make personal balance access permissioned per-user

6.2 Architecture



6.3 UML Specification

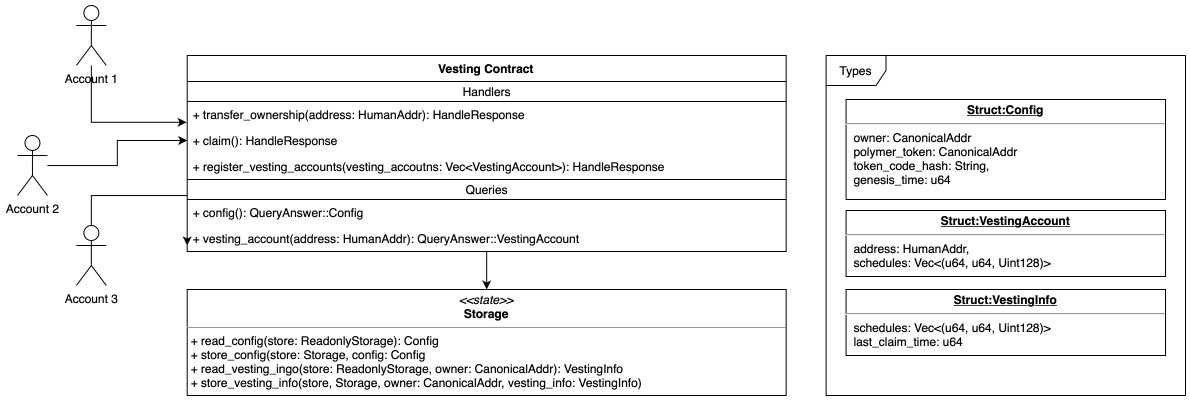


Figure 5: Smart Contract - Class diagram for the vesting implementation.

7 IBC / Bridged Asset Wrapping Contract

Link to Code on GitHub

7.1 Overview

The IBC bridging contract is a slightly modified version of the SNIP-20 reference implementation. The current state of this contract doesn't support wrapping IBC assets, but there are pending changes that would add this support, so said changes were checked then merged early in this version. Overall, the functionality offered will be about even with wrap.scrt.network. As far as Polymer's smart contracts are concerned, this covers the necessary functionality to ship with integrated bridging.