requiem

finance

Whitepaper

Abstract A pool-governed liquidity acquiring AMM that allows liquidity providers to govern the terms of the ech pool based on ownership

Summary

The Requiem Protocol consists of multiple parts:
A **DEX** that allows users to generate fees,
A **pool governance** that allows liquidity owners to vote on trade parameters,
An **asset-backed token (abREQ)** that users obtain by selling liquidity to the protocol,
A **bond depository** for any asset type to sell liquidity in a structured way to the protocol,
A **governance token** that can be obtained by locking abREQ,
That **allows holders to vote on global protocol fees**, new directional decisions and
That can be **staked for earning** these **protocol fees**.

DEX

The DEX is initially made up of two trading structures - pairs and a stable swap. These will be integrated together with a standard router that allows users to trade through these pools. Exact output trading will be supported, too.

Pairs

The pairs follow an adjusted UniswapV2 trading structure that allows liquidity providers to more flexibly determine how much tokens are provided on each side for a pair. The token amounts in the pairs are scaled through weights in their exponents. In addition to that, we use an amplification parameter to simulate higher liquidity which results in lower slippage for traders.

AMM

The invariant equation is as follows.

$$(\lambda x)^{w_x/50} \cdot (\lambda y)^{w_y/50} = k,$$

Where

- $x > 0$ amount of the first token
- $y > 0$ amount of the second token
- $w_x \in [2, 98]$ weight for the first token, natural number divisible by two
- $w_y = 100 - w_x$ weight for the second token
- $\lambda \geq 1$ amplification parameter - scales liquidity amounts
- $k > 0$ invariant - only changes when liquidity is added or removed

In case of a trade of Δx for Δy the following equation has to hold:

$$(\lambda x - \Delta x)^{w_x/50} \cdot (\lambda y + \Delta y)^{w_y/50} = k$$

That provides the output amount:

$$\Delta y = \frac{k^{50/w_y}}{(\lambda x - \Delta x)^{w_x/w_y}} - \lambda y = \frac{(\lambda x)^{w_x/w_y} \cdot (\lambda y)}{(\lambda x - \Delta x)^{w_x/w_y}} - \lambda y$$
$$= \lambda \frac{(\lambda x)^{w_x/w_y} \cdot y}{(\lambda x - \Delta x)^{w_x/w_y}} - \lambda y = \lambda \left(\frac{(\lambda x)^{w_x/w_y}}{(\lambda x - \Delta x)^{w_x/w_y}} - 1\right) \cdot y$$

If both weights are 50, it is easy to derive an average price based on the quotient of amounts $\frac{y}{x}$ in the pool. However, if the pool has different weights, the estimation is not as straightforward. We look at the limit for the smallest amount Δx traded in (and receiving $\Delta y(\Delta x)$ output amount from it) towards zero and the resulting price:

$$\frac{\Delta y}{\Delta x} = \frac{\left(\frac{(\lambda x)^{w_x/w_y}}{(\lambda x - \Delta x)^{w_x/w_y}} - 1\right)}{\Delta x} \lambda y = \lambda y \frac{\lambda x^{w_x/w_y} - (\lambda x - \Delta x)^{w_x/w_y}}{\Delta x (\lambda x - \Delta x)^{w_x/w_y}} \to \frac{w_x}{w_y} \frac{\lambda y}{\lambda x} = \frac{w_x}{w_y} \frac{y}{x}, \ \Delta x \to 0$$

We see that the amplification parameter does not affect the pool price.

That structure allows providing liquidity amounts that do not directly determine the price, the weights can be set in a manner such that for specified liquidity amounts a more or less arbitrary price can be set if the appropriate weights are chosen.
The pool value now calculates differently, too. It is provided as

$$L_y = y + p \cdot x = y + \frac{w_x}{w_y}\frac{y}{x} \cdot x = (1 + \frac{w_x}{w_y}) \cdot y$$

A swap fee is also added that will simply be charged on the input or output amount.
Both the swap fee and amplification parameter can be adjusted dynamically - that allows us not to run into various problems such as too high volatility in case of high amplification parameters (resulting in a huge impermanent loss) and/or an uncompetitive pricing due to too high fees.

Next to the Requiem interface, we also implement the standard UniswapV2 type interface for swaps, making the pairs compatible with most aggregators in the DeFi world.

Pair Governance

We allow amplification parameters and swap fees to be set dynamically by a governing contract. That allows users to make their vote count if they think that the fee and ergo the earnings are too low or that the fees are too high so that the pool pricing is not competitive anymore.

Weighted n-Pools

We can do the same as above for more than 2 tokens in one pool. We want to aim for super low gas costs which we only will achieve when skipping the liquidity scaling feature due to the contract size limitations when using the highest optimization.

The trading model is exactly the same as for a pair, just for a full product of n:

$$\prod_{i=0}^{n} x_i^{w_i} = k, \quad \sum_{i=0}^{n} w_i = 1.$$

The first implementation will only allow swaps in pairs. For a swap of token k with token l, the following condition has to hold:

$$(x_k - \Delta x_k)^{w_k} \cdot (x_l + \Delta x_l)^{w_l} = k_{kl} := x_k^{w_k} \cdot x_l^{w_l}$$

This is exactly the constant product formula for a pair, meaning that the implementation is not much more computational intensive as for regular pairs.
Of course one could implement a general swap - allowing a swap of $X_K = \{\Delta x_k, k \in K\}$ against $X_L := \{\Delta x_l, l \in L\}$ for $K, L \subset \{0,\dots,n\}$ and $K \cap L = \{\}$- which would then require the following condition:

$$\prod_{i \in K \cup L} (x_i - \Delta x_i)^{w_i} = \prod_{i \in K \cup L}^{n} x_i^{w_i}$$

That is certainly feasible, however, for proper utility, it requires an additional calculation library (for calculating these output amounts for multiple provided inputs). In addition to that, it is quite expensive computation-wise. Therefore, a structure like this will only be implemented later.

In addition to general weights we will have a special implementation for equal weights, meaning that the pool with n tokens with weights of $1/n$ each will be implemented using less expensive calculation methods. That is, because the invariant is just the product of the normalised balances to the power of $1/n$. that only would require a single exponentiation. The trade calculations themselves would just be regular products and sums making it cost substantially less gas compared to the general weighted pool.

Stable swap

A StableSwap implementation aligned to compound, but extended with an exact out swap, logic allows low-slippage trading for stablecoins
Everything is implemented under a general interface that allows further integrations of structures like weighted 3-pools.
The initial deployment will be designed specifically for stable coins, but can later be extended to various interest bearing tokens or BTC/ETH variants

Diamond Swap

Implementation of an upgradeable pool - governed by its own LP Tokens

- The use of the diamond structure should allow for a more efficient implementation of the mathematics as there is no size limitation on the contract
- Allows tokens to be added and removed in a relatively flexible manner
- Full control of all kinds of parameters
- Stable swap, weighted pool, liquidity bootstrapping pool (time-dependent weights), meta pool (pool with derived price), and all of these with additional features like liquidity amplification

Router

We design the pools in a way to have maximum synergy with connecting them through routing. Both forward and backward swapping will be implemented.

Forward Swap

Forward swapping, i.e. providing an exact token amount in for a variable amount in return - that one is straight forward - we just calculate the return amount with the equation that is equivalent to the invariant equation (which makes the check of the invariant obsolete) and execute the swap.

Backward Swap

Backward swapping is more tricky - that is, because for a pool route $p_1, p_2,..., p_n$ and token path $t_1,... t_n, t_{n+1}$, we have to calculate the input amount in t_n for a given output t_{n+1} and then from t_{n-1} for the obtained t_n amount.

To tackle that problem, one could just use flash swaps, however, calling these costs more gas than just calculating the price twice, once when calculating the whole route amounts and a second time when executing the swaps (as they internally validate the swap with the same calculation).

Flash Swaps

These were introduced with UniswapV2. Essentially a flash swp is a flash loan that also can include a swap with the pool itself. We separate these for pools with more than two tokens, but still implement a flash swap for only two tokens. The following use cases arise:

- User withdraws tokenA and wants to pay back with tokenB
    - tokenA exact amount is provided
    - tokenB exact amount is provided

Treasury

The treasury is implemented as EIP-2535 Diamond - meaning that the contract has the maximum upgradeability possible.
The current features include

- Permission management to allow other contracts to interact with the treasury
- Deposit / Withdraw features for providing price-management of the Asset-Backed Requiem Token
- Bond Depositories (see next chapter) can be built on top of it

Planned are the following features:

- Requiem Bank - Design of an algorithmic stablecoin partly collateralized by the treasury reserves
- Lending feature

Bonding

Instead of Farming the governance token of the Requiem platform will be distributed through bonding aligned to the Olympus DAO framework.
That allows more controlled token minting and keeps the inflation in bounds.
In addition to that, the lock mechanism assures that the DEX is provided with liquidity over fixed time horizons.

Basic concept

Since the bonding structure is built on top of the treasury, we have full flexibility to add and remove bonding contracts that users interact with.
From the protocol's perspective, the objective is to sell abREQ for a target amount of LP tokens over a given period, e.g. aiming to purchase 1,000 LP tokens over 1 week.

Regular (Vanilla)

This is the OlypusDAO model - acquiring ABREQ through bonding (without assuming impact from other treasury parameters) decreases the price of the bond - making it more attractive for new buyers to get in. This model replicates demand induced by the protocol.
Generally the bonding price is given by

$$p = c \cdot d$$

where c is the control variable and d is the debt ratio which in return is provided by

$$d = \frac{cd}{s}$$

and $cd = td - decay$ with td being the total debt in ABREQ and $decay$ being a function that reduces the total debt in given intervals.
The total debt is reduced whenever ABREQ is paid out and when the time intervals are triggered.
On bonding, the user obtains

$$N = \frac{v}{p}$$

abREQ units, where v is the value of the asset amount that is bonded in USD.
These bonds are ideal for bonding (i.e. selling) stable tokens, stable pool liquidity or abREQ-xUSD (with xUSD being a stablecoin) liquidity to the protocol.

Digital Call

It has the same price development structure as the regular bond, however, each bond has an underlying index price I, a given maturity T, a strike percentage S and digital payoff percentage D. Similar to above, the user obtains $N = \frac{v}{p}$ tokens on bonding at time t. The index price I_t is also recorded for the user.

At maturity the user can claim the notional plus an optional payoff at T^* of

$$D \cdot N$$

if

$$(1 + S) \cdot I_t \geq I_{T^*},\ T \leq T^* \leq T + \Delta T.$$

ΔT is hereby a timespan in which the user can exercise (e.g. one day). If that is not the case, the user can only claim the notional.

Callable

The base price development follows the same logic as for the vanilla case. The parameters upon creation are an underlying index price I, a given maturity T, a strike percentage S and a payoff cap M.

If for any time t_1 after creation at t_0 with recorded index price I_{t_0} the condition

$$(1 + S) \cdot I_{t_0} - I_{t_1} > 0, \quad t_0 < t_1 \leq T$$

holds, the user can immediately claim

$$(1 + \min\{\frac{(1+S)\cdot I_{t_0} - I_{t_1}}{I_{t_0}}, M\}) \cdot N$$

units of abREQ at time t_1. If that condition does not hold until the maturity T, the user only can claim N units at time T and after.

Liquidity Tokens

The core objective is that the treasury holds a diversified portfolio of liquidity assets that appreciate in value over time. Liquidity with respect to all structures as described above will be accepted and can be sold to the treasury.
For these, we require contracts that calculate the LP value i.e. the value of tokens deposited.

Requiem-related LP

We use an adjusted formula for the bond valuation which is necessary due to the use of weighted pairs as LP.
Instead of the classic UniswapV2 constant product, we incorporate an adjusted formula. We already know that the pool price for the constant product

$$x^{w_x/50} \cdot y^{w_y/50} = k$$

is

$$p = \frac{w_y}{w_x}\frac{x}{y}.$$

The corresponding UniswapV2 pool that would produce the same price would consist of the amounts

$$x_{V2} \cdot y_{V2} = k_{V2}, \qquad x_{V2} = \frac{w_y}{w_x}x,\ y_{V2} = y,$$

or

$$x_{V2} \cdot y_{V2} = k_{V2}, \qquad x_{V2} = x,\ y_{V2} = \frac{w_x}{w_y}y.$$

It has to be noted that the trading behaviour slippage-wise is as if the two lower amounts are provided.

We therefore define the RFV (the amount of funds the treasury guarantees to use for backing the Requiem token) for the case of REQT LP as follows. Let x be the REQT token amount in the pool. The original valuation form is

$$LP_{RFV-U2} := 2\sqrt{x \cdot y} \leq x + y$$

We can see that the sum of the amounts is an upper boundary for the valuation as done by OlympusDao. Since we try to approach that problem with weighted pairs that allow a pool that mostly has REQT in it, we are required to do some adjustments. The factual value of the pool in currency Y is

$$L = y + p \cdot x = y + \frac{w_x}{w_y}\frac{y_A}{x_A}x = \frac{w_y x_A y + w_x y_A x}{w_y x_A} \geq \frac{2\sqrt{w_y w_x}\sqrt{y \cdot y_A x \cdot x_A}}{w_y x_A}.$$

We have

$$\frac{2\sqrt{w_y w_x}\sqrt{y \cdot y_A x \cdot x_A}}{w_y x_A} \leq \frac{w_y x_A y + w_x y_A x}{w_y x_A} \leq \frac{w_y x_A y + w_x y_A x}{w_y x_A}$$

Keeping in mind that a very low selection of w_y could allow a minimal amount of Y supplied that results in an extremely high liquidity value. We use therefore the following formula:

$$LP_{RFV} := \frac{2\sqrt{w_y w_x}\sqrt{y \cdot y_A x \cdot x_A}}{w_y x_A}$$

Non-Requiem Weighted Pair LP

If none of the tokens in the pool is a Requiem token, the LP will be priced at the exact value provided by the constant product of weights pairs

Stable Pool LP

Stableswap LP are also eligible for bonding, for the pricing a 1bp market depth valuation is used to price the whole LP. That is necessary as in some cases there is no regular pool price as for UniswapV2 or weighted pairs. Note that this approach works for every pool that contains a swap calculator function with n tokens and quote token index q:

$$value = reserve_q + \sum_{i=0,\, i \neq q}^{n} swapPriceExactIn(\frac{reserve_i}{10000}, \frac{reserve_q}{10000}) \cdot 10000$$

The exact-in swap function is quite standard for all kinds of pools which makes it a strong tool for general pricing.

Weighted Pool LP

Similar to a stable swap, (usually) more than 2 tokens are deposited into a weighted pool. For the pricing, we can directly estimate the pool price as defined above. Using the same approach as for the Stable Pool would work as well, however, it is unnecessarily gas consuming. Pricing the weighted pool works as follows for n tokens and quote token index q:

$$value = \sum_{i=0}^{n} \frac{weight_i}{weight_q}\frac{reserve_q}{reserve_i} \cdot reserve_i = reserve_q \cdot \sum_{i=0}^{n} \frac{weight_i}{weight_q}$$

Direct reserves

Reserves in the form of “regular” tokens will not be accepted in general. That is because these tokens usually do not produce value and just holding it would have a stabilising effect but would not appreciate in value over time.
As such we only want regular tokens to be deposited only if Asset Backed Requiem requires the backing (i.e. substantial drop in value).

Other Crypto Assets

We will consider using further crypto assets that can be sold to the treasury. That option will always be open as the treasury can add bonding calculators allowing it to also include interest bearing tokens.
However, such a mechanism needs to be fleshed out before it can be properly used.

Farms

Nowadays farms are a crucial tool to incentivize liquidity provision. Our goal is to help new projects to use the implemented distribution- and farming contract to incentivize liquidity provision directly on the Requiem platform.
Farming rewards are provided only at specific times when abREQ has been accumulated due to penalties from early unlocks of GREQ. These tokens will then be allocated as farming rewards for promoting lp staking.

Governance

The Governance framework consists of a dual-token system, one from the general governance through Asset-Backed Requiem Tokens, the other from

Requiem Token Governance

A new governance model for Requiem shall make the usage of a governance token less static and shall allow users to trade locked Requiem Token positions.
The new model will allow users to lock abREQ for different maturities where a set of specific maturity dates will be incentivized. It shall allow users

- …to trade a locked position or a share of it against abREQ and therefore against anything
- …to extend maturities of locked positions for a larger voting power, that can either be done flexibly to any maturity or it is done via roll-over to the next standardised maturity
- …to increase position sizes of the lock without extending the maturity

To make that possible, we track a list of locks and minted representing governance tokens for each user.
To measure the amount of governance tokens minted for each user, we use the lockedPosition as reference.
As a prototype function for calculating the minted governance token amount we use an exponential function for $a, b > 0,\ c < 0$

$$r_{base}(t_0, t_1) = a \cdot \exp(b \cdot (t_1 - t_0)) + c,$$
$$a + c = 0,$$
$$a \cdot \exp(b) + c = 1$$

as a prototype. That ensures values between 0 and 1. The second equation is equivalent to

$$b = \ln \frac{1-c}{a} = \ln \frac{1+a}{a}.$$

Compounding, i.e. adding time to the locked funds to unlock more is done via

$$r_{new}(t_{n-1}, t_n) = (r_{prev}(t_0, t_{n-1}) - c) \cdot \exp(b \cdot (t_n - t_{n-1})) + c$$
$$= a \cdot \exp(b \cdot (t_{n-1} - t_0)) \cdot \exp(b \cdot (t_n - t_{n-1})) + c = a \cdot \exp(b \cdot (t_n - t_0)) + c.$$

Assuming that a and b are always constant, we get the following equation for a lock with maturity (measured from inception) $t > 0$:

$$r(t) = a \cdot \exp(b \cdot t) + c$$

The amount of tokens minted for the position of size x is then

$$g(x, t_{start}, t_{end}) = x \cdot r(t_{end} - t_{start})$$

That specific amount is required to unlock the original Requiem tokens.

Governance-utility-dependent rate

We define a as the governance utility rate of abREQ

$$a := \frac{abREQ\ Supply - gREQ\ Supply}{abREQ\ Supply}.$$

The remaining parameters can be derived from that equation.

Creation of Lock

Creating a lock will set the lock parameters, the token amount minted as described above. The multiplier is initially set to

$$m_0 = 1 + \frac{t_{end} - t_{start}}{\tilde{T}} r$$

per $\tilde{T}$ unit of maturity.

Earning Power of locked abREQ

The earning power is defined as

$$earnings\ weight = \frac{\sum_{l\ is\ lock} m_l \cdot g_l}{g_{minted}} \cdot g_{staked}$$

This means that

Roll-overs and maturity extension

If a roll-over or maturity extension is done from t_{end1} to t_{end2}, we simply add

$$x \cdot \frac{t_{end2} - t_{now}}{T}$$

to the minted amount.

Increase of position size

An increase of the position size will result in additional tokens being minted for the respective maturity and an adjustment of the multiplier on the maturity. The increase shall not reset the unlock time, however, the multiplier will be amended if something to the position is added. For initial positions of p_0 and added amount p_1 that is done via position-weighted average:

$$m_1 = \frac{p_0 m_0 + p_1 \frac{mat - now}{maxTime}}{p_0 + p_1}$$

Note that the multiplier decreases if that is done.

Trading

Sale of a position

If a position is sold to another user, the position data will be removed for the seller. If only a share of it is sold, just the position size locked and minted amount will be reduced by the share.Adjustments of the multiplier are not made for the seller.

Purchase of a position

If a locked position is acquired, the effect has to be exactly the same as if a new position is created or if the size of an existing position is increased. That means for a new position:

$$y_0 = x \cdot (mat - refDate)$$

$$m_0 = \frac{mat - now}{mat - refDate}$$

And if a position already exists:

$$y_1 = y_0 + x \cdot (mat - refDate) = (x_{old} + x) \cdot (mat - refDate)$$

$$m_1 = \frac{x_{old} \cdot m_0 + x \cdot \frac{mat - now}{mat - refDate}}{x_{old} + x}$$