Cod3x Reliquary Whitepaper

Beirao[1]

[1]Conclave, Head of Security, beirao@conclave.io

Abstract

Conventional emissions systems distribute reward tokens proportionally to a user's deposit. Reliquary enhances the capability by allowing for a time-weighted approach to deposit calculations, rewarding users for the size of their deposit, as well as their deposit duration. The tokenized nature of Reliquary has unique implications for attracting long term liquidity, as well as empowering loyal users with governance. The protocol owner can credit the contract with reward tokens and set the desired issuance.

Keywords: Cod3x, Reliquary, Rewards Emission, Maturity Curve, NFT

1. Relic Design

The Reliquary System (Cod3x-Labs, 2024) is designed to manage incentives for deposited assets such that behaviors can be programmed on a per-pool basis using maturity levels. Stake in a pool, also referred to as position, is represented by means of a Non-Fungible Token (NFT) called a *Relic*. Each position has a *maturity* which represents the age of the position.

Deposits are tracked by $RelicID$ instead of by user. This allows for increased composability without affecting accounting logic, and users can trade their *Relics* without withdrawing liquidity or affecting the position's maturity.

Compared to conventional systems like the popular MasterChef contract (Sushiswap, 2021), the Reliquary contract offers more flexibility and customization:

- Emits tokens based on the Maturity of a user's investment.
- Binds variable $EmissionRates$ to a base maturity curve designed by the developer for predictable emissions.
- Supports deposits and withdrawals along with these variable rates, which has historically been impossible.
- Issues a 'financial NFT' ('*Relic*') to users which represents their underlying positions, able to be traded and leveraged without removing the underlying liquidity.
- Can emit multiple types of rewards for each investment, as well as handle complex reward mechanisms based on deposit and withdrawal.

1.1 Reward Mechanism

There are two types of rewards:

- $InfiniteRewards$ is when the $EmissionRate$ is the parameter that determines the reward distribution. Issued tokens are owed to stakers regardless of whether the pool is sufficiently funded or not. If this is the case, the reward distribution will be delayed and the admin is required to refund the Reliquary contract with reward tokens. This system works well for governance tokens.
- $FiniteRewards$ is when the quantity of tokens in the reserve is the parameter that determines the reward distribution. Unlike $InfiniteRewards$, $FiniteRewards$ stops distributing rewards when the reserve is empty. This system is well suited for protocol revenue distribution.

System architecture can be observed in Figure 1.

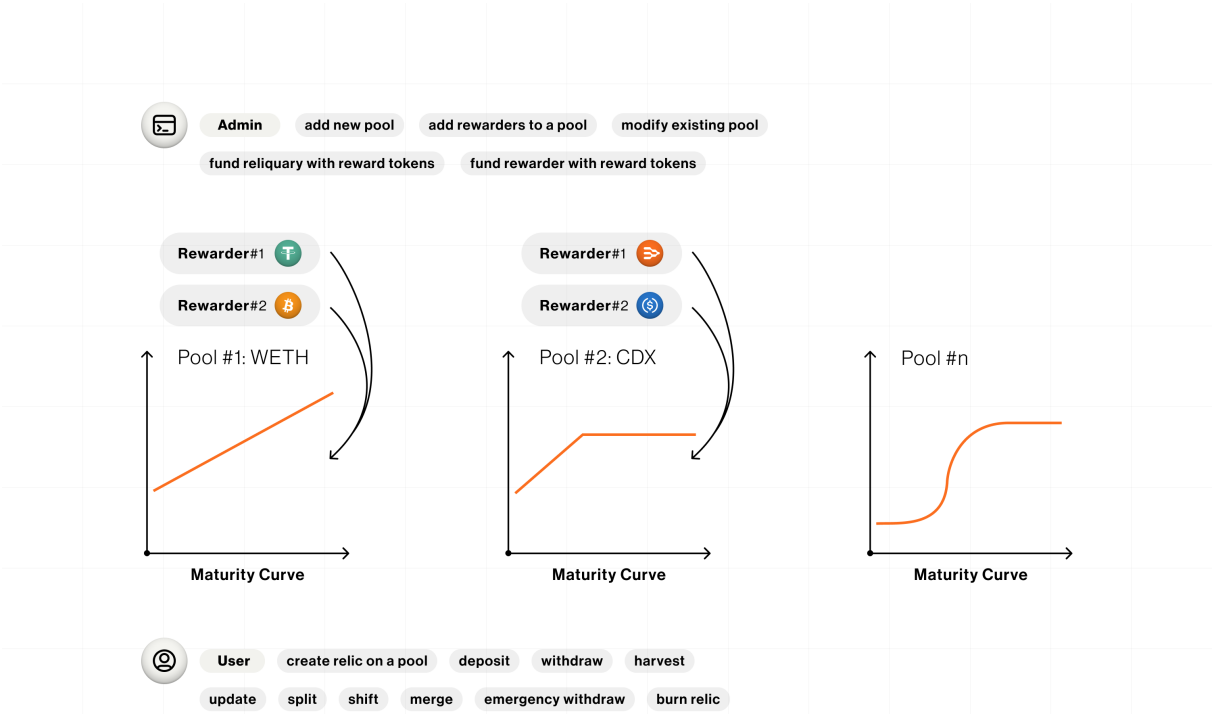

Figure 1: Reliquary system architecture.

Upon deploying Reliquary, an immutable *MainRewardToken* must be designated, which leverages the Infinite Rewards system. The *EmissionRate* is globally applicable to all Reliquary pools (up to 255 pools) and can be adjusted by the administrator.

Rewarders are also managed by the administrator and utilize the Finite Rewards system.

Each pool is associated with a maturity curve. Both the *Rewarders* and the distribution of the *MainRewardToken* adhere to the maturity curve of their respective pools.

1.2 Reward Calculation

1.2.1 Pool Update

Rewards are incremented on a given $pool_n$ each timestep using Equations 1 and 2:

$$\Delta T = T_m - T_{m-1}, \tag{1}$$

where:

- ΔT is the change in time between now and the last rewards update, in seconds
- T_m is the timestamp of the current update
- T_{m-1} is the timestmap of the previous update.

$$\Delta R_{pool_n} = \Delta T * E * \beta_{pool_n}, \tag{2}$$

where:

- ΔR_{pool_n} is the rewards the pool has accumulated during ΔT, in seconds
- E is the global $EmissionRate$, in wei per second
- β_{pool_n} is the pool's share of the global $EmissionRate$.

The reward calculated is then added to a pool variable that tracks the reward accumulation per shares. Given a $pool_n$ that has R positions, the relationship is defined in Equations 3 and 4:

$$S_{pool_n} = \sum_{r=0}^{R-1} A_{r,pool_n} * \varphi(M_{r,pool_n}), \tag{3}$$

where:

- S_{pool_n} is the maturity-adjusted liquidity supplied in the pool
- $A_{r,pool_n}$ is the *amount* in the position
- $\varphi(M_{r,pool_n})$ is the maturity curve function
- $M_{r,pool_n}$ is the maturity of the position.

$$\lambda_{pool_n} = \frac{\Delta R_{pool_n}}{S_{pool_n}}, \tag{4}$$

where:

- λ_{pool_n} is the rewards the pool has accumulated during ΔT.

1.2.2 Relic Update

Given a simple *Relic* 'update' (without any fund deposits or withdrawals), and given a *Relic* r as part of the $pool_n$, the reward owed to r between T_{m-1} and T_m is determined by Equation 5:

$$\Delta C_{r,pool_n} = A_{r,pool_n} * \lambda_{pool_n} * \varphi * \Delta T, \tag{5}$$

where:

- $\Delta C_{r,pool_n}$ is the *Relic*'s reward credit for the period ΔT.

1.3 Advanced Operations

In addition to typical operations such as deposit, withdraw, harvest, and update, the tokenized nature of Reliquary offers three advanced operations that users can apply to their *Relics*:

- *Split*: Users can split a specified *amount* from an owned *Relic* into a new one, while maintaining the maturity of the original *Relic*. Since the underlying liquidity position represented by the *Relic* is still committed, users are not penalized for splitting their *Relics*. Users can split their positions to manage risk, sell partial amounts, etc. without sacrifice their rewards and their time.

- $Shift$: Users can transfer a specified *amount* from one *Relic* to another, updating the maturity in the receiving *Relic*. Since the underlying liquidity position represented by the *Relic* remains committed, users are not penalized for shifting *amounts* between their relics. For example, if a user acquires another position and wants to cross load maturity or capital that can do so freely without sacrificing the time they have committed to develop their maturity.

- $Merge$: Users can transfer the entire position, including rewards, from one *Relic* to another. This process involves burning the original *Relic* and updating the maturity in the receiving *Relic*. By extention of $shift$, users are free to aggregate their capital and maturity into one *Relic* without sacrificing the maturity of their original position.

These advanced operations provide users with greater flexibility and control over their *Relics* within the Reliquary platform.

1.4 Emergency Features

Reliquary is designed with an immutable codebase to ensure stability and security. Two additional emergency features are implemented to mitigate potential issues:

- Pause Functionality: The administrator has the capability to pause the system in case of an emergency.

- Emergency Withdrawals: In critical situations, users can utilize emergency withdrawals to retrieve funds without going through the standard rewards distribution process.

These features serve as a safety mechanism, ensuring the integrity and reliability of the Reliquary platform.

2. Maturity Curves

The initial version of Reliquary comes with three highly customizable maturity curves to suit any protocol needs; for example: revenue or governance token distribution.

Examples of generic maturity curves can be observed in Figure 2.

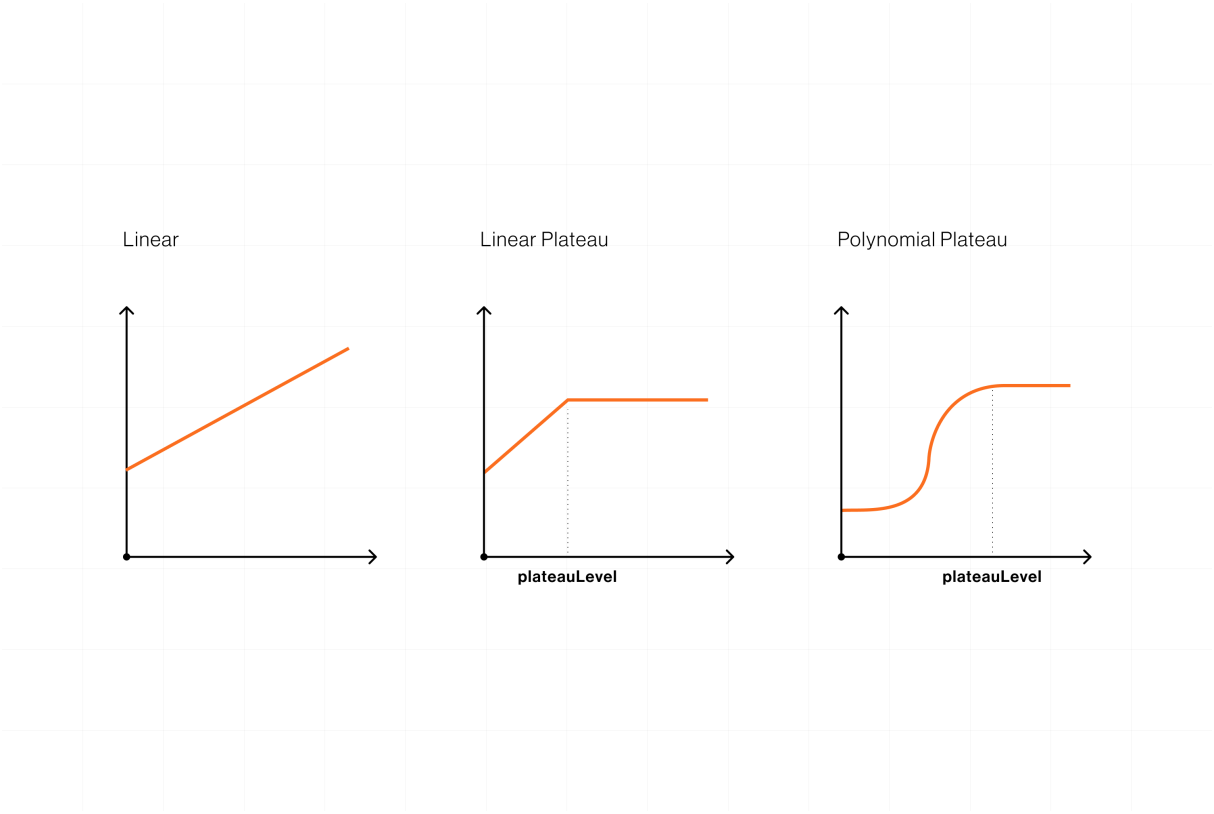

Figure 2: Reliquary system generic maturity curve examples.

The maturity curve is used to determine the multiplier that will be applied to a *Relic*. The older the *Relic* is the higher the multiplier will be. Given a *Relic* r, the reward multiplier applied is presented in Equation 6:

$$\sigma_r = \frac{\varphi(M_r)}{\varphi(0)} \tag{6}$$

where:

- σ_r is the *Relic* multiplier, whereby a multiplier of two indicates that the *Relic* earns twice the rewards compared to a newly created *Relic* of the same size
- M_r is the *Relic* maturity, measured in seconds
- $\varphi(t)$ is the *Relic*'s maturity curve function.

Reliquary's modular design means that new maturity curves can be easily integrated in the future.

2.1 Linear

With the *Linear* maturity curve, older *Relics* will continue earning additional rewards following a linear function.

$$maturity = t * slope + minMaturity \tag{7}$$

where:

- t is time
- $slope$ is the linear function gradient
- $minMaturity$ is the linear function maturity offset, typically set to one.

2.2 Linear Plateau

With the *LinearPlateau* maturity curve, older *Relics* will continue earning an increasing number of rewards following a linear function until reaching the plateau level, where rewards are constant.

$$t < plateauLevel \Rightarrow maturity = t * slope + minMaturity \tag{8}$$

$$t \geq plateauLevel \Rightarrow maturity = plateauLevel * slope + minMaturity \tag{9}$$

where:

- $plateauLevel$ is the level at which the maturity plateaus.

2.3 Polynomial Plateau

The *PolynomialPlateau* allows the developer to input an array of coefficients (c_k) representing a polynomial function that defines the maturity curve of the *Relic*. This provides broad functionality for almost any use case that involves Reliquary, that isn't captured by the linear and linear plateau maturity curves.

$$t < plateauLevel \Rightarrow maturity = \sum_{k=0}^{K-1} c_k * t^k \tag{10}$$

$$t \geq plateauLevel \Rightarrow maturity = \sum_{k=0}^{K-1} c_k * plateauLevel^k \tag{11}$$

3. Conclusion

The Cod3x Reliquary system represents a significant advancement over traditional incentive distribution systems, offering enhanced flexibility, customization, and composability.

By emitting tokens based on investment maturity and binding variable *EmissionRates* to a base maturity curve, Reliquary provides predictable and programmable incentives. The tokenized nature of *Relics* allows users to trade and leverage their positions without affecting the underlying liquidity, further enhancing the utility and liquidity of their investments.

The ability to emit multiple types of rewards and handle complex reward mechanisms based on deposits and withdrawals opens up new possibilities for incentive design and user engagement. The modular design of Reliquary ensures that new maturity curves and features can be easily integrated in the future, making it a robust and adaptable platform for decentralized finance applications.

Reliquary's innovative approach to reward distribution and asset management sets a new standard for incentive mechanisms in DeFi, offering both users and developers a powerful and versatile tool for optimizing their reward distribution strategies.

4. Acknowledgements

Cod3x expresses gratitude to Goober for conceptualizing the notion of time-weighted rewards, Justin Bebis for implementation of the initial prototype, and Zokunei for his implementation work on Reliquary's first iteration.

References

Cod3x-Labs. (2024). Cod3x-labs/reliquary. https://github.com/Cod3x-Labs/Reliquary
Sushiswap. (2021). Sushiswap/masterchef. https://github.com/sushiswap/masterchef