Bonding mechanism with liquidity on Solana

Overview

According to aip-1, here is the workflow to enable LP-token bonding on different chains:

1. Move OLAS token from Ethereum to the target chain, using (preferably) bridges with minimal trust assumptions, with the best-known security, and minimal manual and team interaction to start the process. Alternatively, acquire OLAS tokens on the target chain using other means of exchange.
2. Create an LP-token on the target chain with the bridged OLAS token on the target chain and use popular decentralized exchanges with a Uniswap-v2-style AMM design.
3. Transfer LP tokens back to Ethereum using the same bridge methods.
4. Enable LP tokens in Autonolas' Treasury in order to create bonding products.
5. Use transferred LP tokens for bonding in created bonding products on Ethereum.

On Solana, the Wormhole portal is considered for bridging.

In pursuit of the workflow above, the focus is on selecting a Solana-based Automated Market Maker (AMM) among the top of decentralized exchanges (DEXes) that adhere to a Uniswap-like approach. The aim is to smoothly integrate this with the existing depository model, minimizing the need for modifications. Orca, the AMM on Solana, shares similarities with Uniswap but does not allow the creation of non-concentrated pools.

Despite the possibility of creating a Full range deposit, LPs retain the ability to provide concentrated liquidity within a specific range, and receive non-fungible tokens as representations of their liquidity. Since our depository model requires fungible tokens, to address this characteristics of Orca's liquidity provision model, a specialized fungible liquidity wrapper contract is required, the liquidity lockbox program.

The lockbox program is designed to allow “bonders” to deposit concentrated liquidity tokens (NFTs) from Orca whirlpool contracts and receive fungible token equivalents. To make this work, only LP NFTs representing a full range can be deposited in order to be exchanged for fungible tokens. This approach ensures compatibility with the existing depository model (cf. liquidity lockbox contract section for a description of the contract and depository math section for the depository model).

To finalize the bonding process, users will transfer fungible tokens representing the liquidity from Solana to Ethereum via Wormhole, and use the bridged wrapped fungible liquidity tokens to participate in bonding programs on Ethereum mainnet.

Liquidity lockbox contract

The key methods to understand the contract purpose are the following.

Deposit method:

For a successful deposit, the following conditions must be met:
- Transfer of liquidity NFT to the program lockbox account occurs.
- NFT parameters are verified, including the whirlpool address, all associated accounts, and Tick index ranges (Tick_lower_index=-Max, Tick_upper_index=Max).

Upon verification, the lockbox data account becomes the rightful owner of the NFT. An amount of fungible liquidity tokens corresponding to the NFT position liquidity is minted in favor of the current LP owner.

Withdraw method

This method facilitates a liquidation process, allowing a user to exchange a specified quantity of fungible tokens (X) representing liquidity NFTs in the contract, for the liquidated assets making up the liquidity NFTs. Specifically, the method checks whether the liquidity (L) associated with the last deposited NFT is larger than or equals to the requested withdrawal amount of X. If this condition is met, the corresponding liquidity is decreased, applicable fees are accrued, and the X amount of fungible tokens is burnt. Additionally, when L equals X, the position is fully liquidated and closed.

Note. It's important to note that multiple withdrawal calls may be necessary, contingent on whether the user's fungible token amount X is greater than the last deposited NFT.Specifically, if X is less than or equal to the liquidity of the least recently wrapped NFT, users can perform the withdrawal with the full amount X. Conversely, if X exceeds the liquidity of the least recent NFT, users must perform successive withdrawals for each NFT, starting with the first and proceeding until the total withdrawal amount X is reached.

Depository math

In the current depository math model, the OLAS payout calculation for the specified amount of X LP tokens involves:
- applying the formula for liquidity removal in constraint product AMM,
- evaluating another pair asset (e.g. ETH, XDAI, or SOL) using their spot price within the pool.

For simplification, let's consider the Uniswap-v2 pool OLAS-ETH. The same principles apply to the OLAS-xDAI pool, given its 50:50 weighted balancer pool configuration. When creating a bonding program at time t, the ‘spot’ price of 1 LP in OLAS can be selected as an input parameter and this can be represented as:

priceLP= 2*RESERVE_OLAS(t)/totalSupply[1].

For a bond of X LP-tokens, the resulting OLAS payout is determined by:

OLAS_payput=priceLP* tokenAmount*IDF.[2]

In the Solana case, not all values accrued by the pool can be utilized, as we cannot guarantee that all the LP providers selected are full rangeTherefore, only the liquidity amount deposited for full range is considered, resembling the constraint product AMM, such as Uniswap v2.

When initiating bonding programs, there is no need to get the LP price on-chain, hence the following https://everlastingsong.github.io/account-microscope/#/whirlpool/listPositions/poolAddress can be utilized. Specifically, from this, we will accrue the amount of liquidity designated as full range, the balances of OLAS token and the assets deposited for full range. These values will be used to determine the LP ‘spot’ price and the OLAS spot price as for the constant AMM product.

It is important to note that this approach represents an approximation of real prices represented in the pool. Notably, the prices in the Orca pool, considering positions that are not in full range, may differ from the spot prices. However, this approximation is a necessary measure to maintain consistency with the current depository logic and avoid unnecessary changes.

---

[1] See Appendix to understand where this formula comes from

[2] IDF is an inverse discount factor applied to the bond, and depends on the production of code from the past tokenomics epoch. IDF is larger or equal then 1 and smaller or equal then 1.1.

Appendix

LP-price formula using Uniswap formula for liquidation

Hence, the first thing to do is try to evaluate an LP-share in OLAS. To do that, it is possible to use the following formula from Uniswap liquidation.

1. If user holds x LP-share and remove its liquidity at time t=t0, the users will receive

no=RESERVE_OLAS(t)*x/totalSupply OLAS and ne=RESERVE_ETH(t)*x/totalSupply ETH, (1)

Where RESERVE_OLAS(t) and RESERVE_ETH(t) are respectively the reserves of OLAS and ETH in the pool at the time t.

2. Since we need to estimate how many OLAS we need to allocate, we need to compute how many OLAS I can receive by swapping one ETH in the pool.

Which can be obtained as follows:

mo= RESERVE_OLAS(t)ne/(RESERVE_ETH(t)+ne)

Assuming that ne is significantly smaller than Re(t) this value can be approximated as

mo~ RESERVE_OLAS(t)/ RESERVE_ETH(t)*ne

In the above eq. replacing the ne value we get mo~ x/totalSupply*Ro(t).

Hence, in this idealist case, we can estimate the price of x LP-share as

x*2* RESERVE_OLAS(t)/totalSupply. (*)

References

1. https://pencilflip.medium.com/solanas-token-program-explained-de0ddce29714
2. https://solscan.io/token/So11111111111111111111111111111111111111112
3. https://docs.orca.so/orca-for-liquidity-providers/master
4. https://orca-so.gitbook.io/orca-developer-portal/orca/welcome

5. https://docs.orca.so/orca-for-liquidity-providers/community-listing/how-to-guides/how-to-add-a-token-to-the-orca-token-list
6. https://pencilflip.medium.com/solanas-token-program-explained-de0ddce29714
7. https://github.com/solana-nft-programs/token-manager
8. https://github.com/everlastingsong/solsandbox/tree/main/orca/whirlpool/whirlpools_sdk/create_own_whirlpools/tools
9. https://github.com/everlastingsong/solsandbox/tree/main/orca/pool/sdk/swap_using_phantom/with_wallet_adapter
10. https://github.com/everlastingsong/solsandbox/blob/main/orca/whirlpool/whirlpools_sdk/08a_close_position.ts#L216
11. https://github.com/everlastingsong/solsandbox/blob/main/orca/whirlpool/whirlpools_sdk/create_own_whirlpools/tools/03_create_whirlpool.ts
12. https://stackoverflow.com/questions/70082314/convert-ethereum-uint256-to-solana-rust-u64-in-smart-contract
13. https://github.com/orca-so/whirlpools/blob/2c9366a74edc9fefd10caa3de28ba8a06d03fc1e/programs/whirlpool/src/state/position.rs#L20
14. Deploy SPL and add liquidity to SPL/SOL pool on Orca https://gist.github.com/Flydexo/c4ed592dcb83c338c5ac7e4246512e83
15. Helper class to interact with a Whirlpool account and build complex transactions https://orca-so.github.io/whirlpools/interfaces/Whirlpool.html
16. https://medium.com/coinmonks/pricing-uniswap-v3-nft-positions-ad18608b8c81
17. https://solanacookbook.com/core-concepts/accounts.html#facts
18. https://www.sec3.dev/blog/solana-programs-part-2-understanding-spl-associated-token-account