CoW Grant - Settlement Wrapper Contract

Author

I am meetmangukiya. I am a software developer, have been in DeFi and web3 space for over 3 years now. I also have familiarity with CoW protocol technicals.

Grant Category

Decentralization

Grant Description

Problem

With the recent passage of CIP-44: Reduced bonding requirements, CoW DAO has made the determination to allow solvers a permissionless experience - with reduced bonding requirements - when participating in the batch auction, and crucially settling orders.

Notably CIP-44 does not change the prevailing protocol

bonding requirements, and in effect CIP-44 represents CoW DAO providing conditional permissionless access to the CoW DAO bonding pool

.

The “sub-bond” minimum deposit defined in CIP-44 is:

- $50k in yield bearing stable coins or ETH
- 500k COW

With this ramping to a target deposit of:

- $100k in yield bearing stable coins or ETH
- 1M COW

Scope

Implement a SubPoolFactory

that has deterministic deployment (salt = msg.sender

) and:

1. The SubPoolFactory

SHOULD

be authed by msg.sender

to spend their nominated tokens

1. When deploying a SubPool

, it is funded to at least

the minimum with the representative dollar values checked against a ChainLink oracle (restricted to collateral that has ChainLink oracles - and no multi-collateralisation)

1. Within the SubPoolFactory

, set the msg.sender

as a sub-bonded solver. This should be accompanied by an event, such as Vouch

.

NOTE

: Having deterministic address be bound to msg.sender

allows for programmatic mapping between the solver's private key and their sub-pool. Ensuring that the sub-pool contract address is the target of MEV Blocker rebates becomes more observable on-chain.

The SubPoolFactory

also provides a means for a keeper to update the total fiat value of collateral held by a sub-pool and subsequently given a threshold (time and value) - allow them to be disconnected / deauthed. The SubPool

should also have the ability to be fined by the DAO (whereby the specified collateral is transferred to the DAO). This fine

mechanism should be present on the SubPoolFactory

and authorised appropriately.

The SubPoolFactory

then also provides a "wrapper" / "facade" around the GPv2Settlement

contract such that it provides:

interface Auth { / * **Allows usr to perform root-level functions */ function rely(address usr) external auth;** / * Denies usr from being able to perform root-level functions _/ function deny(address usr) external auth; / * Determine if usr is able to perform root-level functions * @dev May be mplemented as a public mapping (address => bool) ;_ / function wards(address usr) external view returns (bool); }

interface SubPool is Auth { / * **Determine the amount required to heal the pool * @return amt of collateral and cowAmt of COW */ function dues() external view returns (uint256 amt, uint256 cowAmt);** / * Pull in from msg.sender the dues and subsequently call * the SubPoolFactory to thaw the sub-pool (if frozen). _/ function heal() external; / * Indicate a sub-pool's intent to quit the batch auction * At this point the solver who's sub-pool it is is now removed from * the competition. * This then starts the delay timer until exit is allowed._ / function quit() external auth; / * **Withdraw any remaining collateral / cow from the sub-pool. */ function exit() external;** / * URL to send the batches to the solver */ function backendUrl() external view returns (string memory); }

interface SubPoolFactory is Auth { / * **View function for a collateral's ChainLinkOracle * @dev Implemented as mapping(IERC20 token => IAggregatorV3Interface) */ function collateral(IERC20 token) external view returns (IAggregatorV3Interface);** / * Set the delay between quit and exit. _/ function setExitDelay(uint256 delay) external auth; / * Set the delay between poke / fine and freeze._ / function setFreezeDelay(uint256 delay) external auth; / * **Allow token to be used as collateral for the sub-pool. * @notice** MUST **have a chain link oracle available for USD determination */ function allowCollateral(IERC20 token, IAggregatorV3Interface oracle) external auth;** / * Revoke token as a collateral for sub-pools _/ function revokeCollateral(IERC20 token) external auth; / * Deploy a SubPool deterministically with the salt set as msg.sender. * @param token The nominated token to use for collateral * @param amt How much collateral to add to the sub-pool (must be minimally collateralised) * @param cowAmt Amount of COW to add as collateral * @notice Is payable to support ETH._ / function create(IERC20 token, uint256 amt, uint256 cowAmt) payable; / * **Mark the pool as under-collateralised if so depending on ChainLink oracles. */ function poke(SubPool pool) external;** / * Fine a sub-pool * @param pool Which sub-pool to fine * @param tokens The tokens to fine (sending to msg.sender) * @param amts The amount of tokens to fine _/ function fine(SubPool pool, IERC20[] calldata tokens, uint256[] calldata amts) external auth; / * Freeze a pool if they are under-collateralised_ / function freeze(SubPool pool) external; /* _* Thaw pool if they are above collateralisation requirements_ / function thaw(SubPool pool) external; }

interface SignedSettlement is Auth { / * **Approve signer to act as signing authority to attest to the settlements */ function hope(address signer) external auth;** / * Revoke approval for signer. _/ function nope(address signer) external auth; / * Determine if signer can attest_ / function can(address signer) external returns (bool);

```
/**
 * Signed settlement function that is only callable by a collateralised
 * sub-pool (recall `sub-pool` is bound to `msg.sender`).
 * Takes the required settlement data and verifies that it has been
 * signed by a `signer`, and subsequently calls `GPv2.Settlement.settle`.
 * @dev Note the separation of pre-interactions, and other interactions,
 * so as to reduce the signing / hashing requirements. Gas-golfing is
 * a high requirement here.
 */
function signedSettle(
    IERC20[] calldata tokens,
    uint256[] calldata clearingPrices,
    GPv2Trade.Data[] calldata trades,
    GPv2Interaction.Data[] calldata signedPreInteractions,
    GPv2Interaction.Data[][2] calldata otherInteractions,
    bytes calldata signature
) external onlySubPool;

}
```

Grant Goals and Impact

This settlement contract will be used in the implementation ofCIP-44 which furthers the decentralization of solvers.

Milestones

- The contracts - 1 week

Funding Request

3000 xDAI

Gnosis Chain Address

gno:0xd33d055ea52ec5a685995d40f19fa98d0744df0c

Terms and Conditions

By submitting this grant application, I acknowledge and agree to be bound by the CoW DAO Participation Agreement and the CoW Grant Terms and Conditions.