CoW AMM

CoW AMM is an automated market maker running on top of CoW Protocol.

Overview

The AMM contract itself is a dedicated Safe multisig. It stores reserves of two tokens and allows anyone to create orders between these two tokens on CoW Protocol as long as the trade doesn't decrease the product of the reserves stored on the contract.

The order is based on the Programmatic order framework: this repository only specifies the code needed for the dedicated handler and the related price oracles. All the code for theComposableCoW framework ishere .

The AMM creates a CoW Protocol order at a regular time interval. These orders are created with a dedicated contract functiongetTradeableOrder : it returns an order that can be traded in a way that tries to rebalance the AMM to align with the reference price of a price oracle. These orders are provided for convenience to CoW Swap solvers so that basic CoW AMM usage does not need any dedicated implementation to tap into its liquidity. More sophisticated solvers can create their own order to better suit the current market conditions.

Thewatch tower is responsible for automatically creating the AMM order, without any necessity for the AMM to interact with the CoW Protocol API.

CoW AMM orders are executed in batches by CoW Protocol solvers. Only one order per AMM can be executed per batch.

Further information on the theoretical research work that serves as the base of CoW AMM can be found in the paperArbitrageurs' profits, LVR, and sandwich attacks: batch trading as an AMM design response.

Batch trading guarantees that despite the minimum viable order follows the constant-product curve (as it's the case for Uniswap v2) the surplus captured by CoW Protocol yields to a better execution price and thus higher profits for liquidity providers.

Limitations

The current setup doesn't support pooling liquidity among different users at this point in time. A new CoW AMM instance needs to be deployed every time a user wants to provide liquidity for a pair. Native tokens must be wrapped before using them as a token in the pair.

I'm a liquidity provider. How do I create my AMM?

Setting up a CoW AMM can be split in the following steps, which will be described in detail in their own section.

1. Deploy aSafe
2. .
3. Change the Safe's fallback handler toExtensibleFallbackHandler
4. .
5. Set the Safe's domain verifier for CoW Protocol toComposableCoW
6. .
7. Deposit and approve the tokens for trading on CoW Protocol.
8. Create the AMM order onComposableCoW
9. .

All these steps, with the exception of the data in the last one, aren't specific to the CoW AMM but are the same steps required for creating an order with theComposableCoW framework.

You can see an example of CoW AMM trading tokens COW/WETH at address0xBEEf5aFE88eF73337e5070aB2855d37dBF5493A4 . Example set-up transactions:step 1 ,step 2 to 4 ,step 5 .

Step 1: deploy a Safe

The CoW AMM funds will be stored in a Safe, and every pair of tokens requires its dedicated safe. All interactions of the owner with the AMM will go through the Safe interface. There are no requirements on the ownership structure of the Safe, but keep in mind that the owners have full access to the funds.

You can create a Safe following the instruction orthe dedicated page on the Safe interface.

You should not use this Safe for purposes other than running a CoW AMM: the token balances represent the state of the AMM and should not be altered outside of the normal AMM trading process. The same Safe mustnot be used as the AMM

for multiple simultaneous pairs.

Step 2: switch to the extensible fallback handler

CoW Protocol supports validation of smart-contract orders via ERC-1271 signatures. Signature verification forComposableCoW orders (like the order used in the CoW AMM) is handled by a dedicatedfallback handler , theExtensibleFallbackHandler .

You can find more details on the design ofComposableCoW in thededicated page of the CoW Protocol documentation .

On the AMM Safe Wallet interface, create a new transaction on the transaction builder with the following parameters:

- Address: the address of the AMM Safe itself (confirm loading the implementation ABI once prompted)
- To Address: again, the address of the AMM Safe itself
- Contract Method selector:setFallbackHandler
- handler
- : the address of theExtensibleFallbackHandler
- contract for the current chain (all addresses for each supported chain can be found in theComposableCoW contract repo
-)

Step 3: set the domain verifier

This step is used to tell the extensible fallback handler to useComposableCoW to verify CoW Protocol signatures.

You can find more details on the design ofComposableCoW in thededicated page of the CoW Protocol documentation .

On the AMM Safe interface, create a new transaction on the transaction builder with the following parameters:

- Address: the address of theExtensibleFallbackHandler
- contract for the current chain
- To Address: the address of the AMM Safe itself
- Contract Method selector:setDomainVerifier
- domainSeparator
- : the EIP-712 domain separator of the CoW Protocol settlement contract on the current chain (obtained by callingdomainSeparator()
- on the chain's instance of thesettlement contract
- ; for example, on mainnet it can be retrieved fromEtherscan
-)
- newVerifier
- : the address of theComposableCoW
- contract for the current chain (see theComposableCoW contract repo
-)

Step 4: deposit and approve tokens

To approve the pair, call theapprove function on each token for an unlimited amount (115792089237316195423570985008687907853269984665640564039457584007913129639935) and with CoW Protocol's vault relayer as the spender (seecontract repo'snetworks.json for the official deployment addresses).

To deposit the tokens into the CoW AMM, simply send the funds to the AMM Safe as you would do for any other Safe.

Caution: the funds must be sent before executing step 5. Otherwise, the watch tower will encounter an error and stop posting orders to the CoW Protocol API. If this happened, you'd need torecreate your order to restore automated trading.

Step 5: order creation

AComposableCoW order is created by calling the functioncreate(ConditionalOrderParams params, bool dispatch) on theComposableCoW main contract.

The conditional order parameters should be set as follows:

- IConditionalOrder handler
- : the address of the standard deployment ofConstantProduct
- for the desired chain. See filenetworks.json
- for a list of official deployments by chain id.
- bytes32 salt
- : this value is used to make the order unique. It's recommended to use a value that hasn't been used before for an order on the same safe. Note that it's discouraged to use the CoW AMM safe for other orders outside of setting up the AMM, in which case conflicts are not a concern.

- bytes staticInput
- : The configuration values for the CoW AMM. See next section for more details.

Ifdispatch is set to true, then the order will be automatically picked up on CoW Protocol's orderbook by the watchtower service .

ConstantProduct static input

The static input of the constant product handler comprises the following parameters:

- IERC20 token0
- : the first of the two tokens traded by the AMM.
- IERC20 token1
- : the second of the two tokens traded by the AMM.
- uint256 minTradedToken0
- : the minimum amount of token0 that needs to be traded for an order to be returned bygetTradeableOrder
- . Order with lower traded amount can still be created manually.
- IPriceOracle priceOracle
- : the address of a contract that implements the generic price oracle interface.
- See thesection below
- for more information on which price oracles are available.
- bytes priceOracleData
- : the extra oracle information needed to recover the price.
- See the section below for more information on how to set this value based on the chosen price oracle.
- bytes32 appData
- : The app data (as defined in a CoW Protocol order) that must be used for the order to be valid.

If Foundry is available in your system, you can generate the bytes calldata with the following command:

token0=0x1111111111111111111111111111111111111111 token1=0x2222222222222222222222222222222222222222 minTradedToken0=31337 priceOracle=0x1337133713371337133713371337133713371337 priceOracleData=0xca11d47a appData=0x3232323232323232323232323232323232323232323232323232323232323232 cast abi-encode 'f((address,address,uint256,address,bytes,bytes32))' "(token0, token1, minTradedToken0, priceOracle, priceOracleData, appData)"

Supported price oracles

Price oracles are an abstraction that transform disparate on-chain price information into a standardized price source that can be used by theConstantFroduct to retrieve token price information.

We support the following price oracles:

- UniswapV2PriceOracle
- , based on the limit price of a predefined Uniswap v2 pair.
- BalancerWeightedPoolPriceOracle
- , based on the limit price of a predefined Balancer weighted pool.

Contract addresses for each supported chain can be found in the filenetworks.json .

UniswapV2PriceOracle

The Uniswap v2 price oracle returns the limit price that can be computed from an address (pool) that supports theIUniswapV2Pair interface. The oracle data contains a single parameter:

- IUniswapV2Pair referencePair
- : the address of a Uniswap pool for the tokenstoken0
- andtoken1
- .

Note that the order of the tokens doesnot need to be consistent with the order of the tokens in the pool. The order of the tokens in the constant product static input determines that the price is expressed in terms of amount of token0 per amount of token1. If the tokens are not the same as those traded on the chosen reference pair, no order will be created.

If Foundry is available in your system, you can generate the bytes calldata with the following command:

referencePair=0x1111111111111111111111111111111111111111 cast abi-encode 'f((address))' "(referencePair)"

BalancerWeightedPoolPriceOracle

The Balancer weighted pool price oracle returns the limit price that can be computed from one of theBalancer weighted pool implementations.

The oracle data contains a single parameter:

- bytes32 poolId
- : theBalancer pool id
- representing the weighted pool that will be used to compute the reference price.

The reference weighted pool can use any weights and token combination. However, it must be a weighted pool and not a different pool type: there is currently no check in the smart contract to guarantee that the chosen pool is indeed a weighted pool. If used with a different type of pool, the output of the oracle is likely to be completely unreliable.

If the tokens in the AMM orders are not all included in the reference pool, then no order will be created.

If Foundry is available in your system, you can generate the bytes calldata with the following command:

poolId=0x1111111111111111111111111111111111111111111111111111111111111111 cast abi-encode 'f((bytes32))' "(poolId)"

Stopping the AMM

To stop trading, it's sufficient to execute a single transaction toComposableCoW to the functionremove(bytes32 singleOrderHash) .

The input parametersingleOrderHash can be computed with the help of the view functionhash inComposableCoW . Its input is the same as the parameters in the call tocreate ofstep 5 .

You can verify that the hash is valid by querying the view functionsingleOrders onComposableCoW with input the AMM Safe address and the order hash. The call returnstrue if the order is currently being traded andfalse after the call toremove has been executed.

On mainnet, you can compute the hash, as well as verify if the hash is valid, directly from Etherscan's contractread contract page .

Updating the AMM

To update the order parameters you should firststop the AMM .

Then, you just need to repeatstep 5 of the initial setup using the new order parameters. This leads to a single AMM transaction. Remember to use a different value for the salt!

Add funds after deployment

You can always deposit funds to the AMM Safe with simple transfers, both to add funds and to rebalance the AMM manually.

Be careful: depositing funds changes the state of the AMM. It's very important that you transfer both tokens at the same time with no side effects. If only one token transfer is executed, the AMM will be unbalanced until the second transfer is executed. After receiving the first transfer, the AMM may immediately rebalance by selling some of the funds just deposited.