In this post I want to present an idea that was loosely discussed with various people in the past (mainly during ETH Bogota) and which could present a new way of building AMMs that protect their LPs from MEV.

Context

For context, liquidity providers (LPs) in automated market makers (AMMs) suffer from maximally extracted value (MEV) by arbitrageurs that incorporate outside information (e.g. price moves from centralized exchanges). The reason this is causing damage to LPs is a concept that Tim Roughgarden et. al coined LVR (loss versus rebalancing).

While LPs receive a fee for each trade, an arbitrageur will only trade against them if the gain they get from incorporating this information is strictly larger than the LP fee + gas costs. Thus LPs lose money compared to if they were actively rebalancing their portfolio themselves.

The permissionless-ness of swaps creates a race for whoever gets to update the AMM first, leading to most of the arbitrage value being paid in the form of priority fees to the miner/validator instead of the LP.

It is estimated that 60-80% of DEX trading volume today stems from this activity. Overall it’s hard to find studies highlighting that LPing on AMMs is profitable these days (cf. this lively twitter debate from this weekend).

There have been suggestions in the past on how to more fairly capture the MEV that comes from LVR for the LPs (rather than the validator). Those include:

1. MEV capturing AMM (McAMM) - Applications - Ethereum Research(by our very own@Cowry): LP auctioning off the right for first execution in a block to a designated party
2. https://arxiv.org/abs/2210.10601: more concrete implementation of 1)
3. MEV Minimizing AMM (MinMEV AMM) - Applications - Ethereum Research- block producers “tip off” AMMs what the appropriate after block equilibrium price is

Particularly 3) is very similar to the idea I will be presenting here.

Traditional AMMs

Traditional AMMs, such as Uniswap, SushiSwap and Balancer follow a simple but exact trading formula. If the price of ETH is currently $1300 and someone wants to buy 10ETH, they will buy the first couple of wei at a price very close to $1300, the next couple of wei at a price close to $1301, then $1302 and so on until the order is filled.

[

1040×866 87.5 KB

](https://europe1.discourse-cdn.com/business20/uploads/cow/original/1X/8d188f090f76c982996e4653651f06796bd8c369.png)

Graphically speaking, if a trader moves the AMM from “previous marginal price” to “new marginal price” they end up paying a price that is roughly the average between the two (not exactly, but in practice this turns out to be a good enough intuitive approximation).

Assuming the trade moves the AMM price to $1310, the trader will on average only pay ~$1305 (for every wei bought at $1301 they bough one at $1309 as well). The difference between the average execution price and the new (presumably fair market) price is captured by the trader and not by the LP in that case. We refer to this extra value - $50 in the example - as surplus.

This surplus distribution rule also applies in CoW Swap trades (when trading against standard AMMs). This is because the protocol aims to offer trades prices at least as good as other DEX aggregators, which do pass that surplus on to their trader.

Let’ take a concrete CoW batch as an example (we omit AMM fees for simplicity):

- User A buying 100 ETH for USDC
- User B selling 50 ETH for USDC
- AMM with ETH at $1300 (buying 50 ETH will move the price to $1310)

The outcome of this batch would likely be a clearing price at $1305 (trading 50 ETH peer to peer and buying the excess 50 ETH from the AMM, giving user A and B the average execution price of the AMM).

It can be argued that this solution is not “stable”, because

1. If the fair price of ETH was indeed $1305 but the AMM had been moved to $1310 this would open an arbitrage

opportunity for someone to correct the AMM price back from $1310 to $1305.

2. If the fair market price of ETH was indeed $1310 User B might be frustrated having sold ETH at $1305. Not getting the new equilibrium price creates an incentive to try to avoid “trading against the wind”. It may not be feasible to avoid this reliably in practice (and in expectation may not be an issue to the mechanism itself) but it still leads to unstable outcomes (people preferring not having traded in the batch).

Note, that the LVR for the AMM in case 1) would be ~$60 (while the initial trade happened at the fair $1305 price, the “correction” would be ~25 ETH at ~$1307.5 average price) and in case 2) $250 (“fair” price being $1310 average price being $1305)

Surplus Capturing AMMs

CoW Protocol’s general vision is that there should only be a single price per token per block. After all, ordering dependent asset pricing within a single block is what is causing LVR, backrunning and sandwich attacks. That price should ideally be an equilibrium price where demand equals supply and all agents end up with their preferred allocation at this price (no one is envious to trade more or less).

I therefore think it would be nice to have an AMM primitive that fits into this vision. CoW Swap is already creating a single uniform price vector per batch (one where demand equals supply for CoW Swap users while maximizing their surplus). What if an AMM’s trade would be entirely executed at that equilibrium price (rather than at slowly increasing prices over its curve)?

In the example from above the new outcome would be a clearing price of $1310 (trading 50 ETH peer to peer and buying the excess 50 ETH from the AMM for $1310). Note, that this would give the AMM $250 in “surplus revenue” turning it profitable for LPs even in the absence of trading fees (I don’t want to say no trading fees will be needed at all but I think they can be much lower than in today’s AMMs).

[

1036×820 102 KB

](https://europe1.discourse-cdn.com/business20/uploads/cow/original/1X/4eed070adaac28e23809069700aef84cea1a3066.png)

One reason that such a pricing function isn’t working in traditional AMMs is that a trader could split their trade into many tiny transactions each increasing the new spot price only by a bit, therefore effectively sampling the behaviour of tradtional AMMs (buying a few wei at 1300, then 1301, etc).

In CoW Swap however, since there is only one price per batch, such an “attack” would have to be carried out over many blocks (making it much less practical in the presence of “real” trading demand).

For regular users (noise traders) it could be cheaper to trade against such an AMM because the LPs can afford to charge a lower swap fee than competing AMMs (due to less toxic flow & price finding).

At the same time LPs would be protected from LVR (they always trade at the new equilibrium price). Overall volume may be significantly lower than in traditional AMMs due to the fact that arbitrage is slower and limited. However, I still think it could overall be more profitable for LPs to engage in this mechanism.

Implementation

One way to implement such an AMM would be using the newSmart Order primitive that ERC1271 powered orders enable.

Unlike regular AMM interactions these orders are subject to single price clearing and are therefore executed at the same price as all other user orders. A smart order can defer the correct quantity and price finding to a solver and - when asked to trade - only perform a check if a given price and quantity are satisfying its CFMM preference curve. The solver would be able to use this price curve in its computation and would - if applicable - create a “just in time” ERC1271 order on behalf of the smart order with the correct amount (the price would be the same as for all other orders).

Concretely, the smart order would implement ERC1271 something like this:

function isValidSignature(bytes32 hash, bytes calldata signature) external view returns (bytes4 magicValue) { GPv2Order.Data memory data = abi.decode(signature, GPv2Order.Data); require(hash == data.hash(domainSeparator), "invalid order"); require(this == data.receiver, "invalid receiver");

uint256 k = token_a.balanceOf(this) * token_b.balanceOf(this); if (token_a == data.sellToken && token_b == data.buyToken) { require(k <= (token_a.balanceOf(this) - data.sellAmount - data.feeAmount) * (token_b.balanceOf(this) + data.buyAmount), "K violated"); } else if (token_a == data.buyToken && token_b == data.sellToken) { require(k <= (token_a.balanceOf(this) + data.buyAmount) * (token_b.balanceOf(this) - data.sellAmount - data.feeAmount), "K violated"); } else { revert("invalid token pair"); } magicValue = ERC1271_MAGIC_VALUE; }

The “signature” passed into the verification function is the encoded preimage of a limit order that satisfies the AMM’s demand curve (think of it as the point on the curve where AMM demand intersects the excess user supply in the graph above).

The contract then computes its “k” based on current balances and verifies that after a swap at the proposed buy and sell amounts, the invariant is still satisfied. If this is the case it is guaranteed that the trade will at least be executed according to its demand curve (but can potentially capture extra surplus if the execution price is better).

Deciding on the correct limit order to pass in is a task that solvers would have to do as part of their solution finding (similar to how they decide the correct trading amounts for traditional AMMs). The main difference is that trading against surplus capturing AMMs would not happen in the form of “interactions” but instead as native CoW orders.

From the trader’s perspective, this type of AMM would just be an additional liquidity source to CoW Swap. If traditional AMMs offered a better exchange rate for a given trade size, solvers would still choose whatever price is best for CoW protocol traders.

It is also possible to implement this AMM as a “hybrid” CoW native AMM. The contract could still allow regular EOAs trade against it at will (no guarantee of uniform clearing price), but charge a significantly higher swap fee due to the increased likelihood of toxic trades.

We could also implement the original idea of McAMMs where permissionless trades are allowed only after the lead-searcher (in this case CoW Protocol) had traded first. In fact for other common aspects of AMM design (join/exit, LP tokens, etc) it’s probably best to piggy back on existing open source AMMs.

It would be cool to experiment with such a mechanism on CoW Swap today, so I’m hoping to spark a discussion and maybe find some interested community members to build a working version of this.

CoW DAO recently launched a Grants Program, which could be a good vehicle to fund this idea.