

An AMM minimizing user-level extractable value and loss-versus-rebalancing

Conor McMenamin¹, Vanesa Daza^{1,2}

¹*Department of Information and Communication Technologies, Universitat Pompeu Fabra, Barcelona, Spain*

²*CYBERCAT - Center for Cybersecurity Research of Catalonia*

Abstract

We present V0LVER, an AMM protocol which solves an incentivization trilemma between users, passive liquidity providers, and block producers. V0LVER enables users and passive liquidity providers to interact without paying MEV or incurring uncontrolled loss-versus-rebalancing to the block producer. V0LVER is an AMM protocol built on an encrypted transaction mempool, where transactions are decrypted after being allocated liquidity by the AMM. V0LVER ensures this liquidity, given some external market price, is provided at that price in expectancy. This is done by incentivizing the block producer to move the pool price to the external market price. With this, users transact in expectancy at the external market price in exchange for a fee, with AMMs providing liquidity in expectancy at the external market price. Under block producer and liquidity provider competition, all the fees in V0LVER approach zero. Without block producer arbitrage, V0LVER guarantees fall back to those of an AMM, albeit free from loss-versus-rebalancing and user-level MEV.

Keywords

Extractable Value, Decentralized Exchange, Decentralized Finance, Blockchain

1. Introduction

AMMs have emerged as a dominant medium for decentralized token exchange. This is due to several important properties making them ideal for decentralized liquidity provision. AMMs are efficient computationally, have minimal storage needs, matching computations can be done quickly, and liquidity providers (LPs) can be passive. Thus, AMMs are uniquely suited to the severely computation- and storage-constrained environment of blockchains.

Unfortunately, the benefits of AMMs are not without significant costs. For users sending orders to an AMM, these orders are typically front-run, sandwiched, back-run, or censored by the block producer in a phenomenon popularized as MEV [1]. Current estimates for MEV against AMM users on Ethereum are upwards of \$600M [2, 3]. By the nature of AMMs and their continuous liquidity curves, the amount of MEV extractable from an order is increasing in order impact (related in large part to order size and slippage tolerance). Thus, MEV effectively upper-bounds the trade size allowable on current AMMs when compared to the costs for execution on MEV-protected centralized exchanges. This is a critical barrier for DeFi, and blockchain adoption in general.


DLT 2023: 5th Distributed Ledger Technology Workshop, May 25–26, 2023, Bologna, Italy

✉ conor.mcmenamin1994@gmail.com (C. McMenamin); vanesa.daza@upf.edu (V. Daza)

🆔 0000-0002-5280-3273 (C. McMenamin); 0000-0003-0583-7929 (V. Daza)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

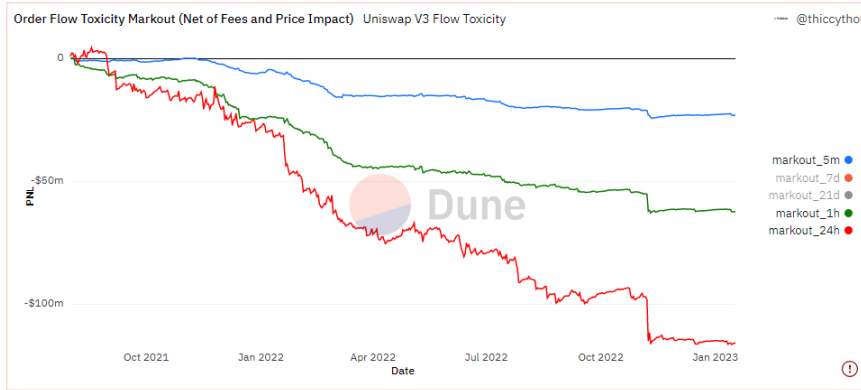


Figure 1: Toxicity of Uniswap V3 Order Flow [6]. This graph aggregates the PnL of all trades on the Uniswap V3 WETH/USDC pool, measuring PnL of each order after 5 minutes, 1 hour, and 1 day. These are typical time periods within which arbitrageurs close their positions against external markets. This demonstrates the current losses being suffered by AMM pools are significant, consistent, and unsustainable. As LVR is significant and consistent, a large part of these losses can be prevented by minimizing LVR.

Another significant cost for AMMs is definitively formalized in [4] as *loss-versus-rebalancing* (LVR). It is proved that as the underlying price of a swap moves around in real-time, the discrete-time progression of AMMs leave arbitrage opportunities against the AMM. In centralized finance, market makers (MMs) typically adjust to new price information before trading. This comes at a considerable cost to AMMs (for constant function MMs (CFMMs), [4] derives the cost to be quadratic in realized moves), with similar costs for AMMs derived quantitatively in [5]. These costs are being realized by LPs in current AMM protocols. Furthermore, toxic order flow, of which LVR is a prime example, is consistently profiting against AMM LPs (Figure 1). These costs together are dooming DeFi, with current AMM design clearly unsatisfactory. In this paper, we provide V0LVER, an AMM protocol which formally protects against both order-level MEV and LVR.

1.1. Our Contribution

In this paper we introduce V0LVER¹, an AMM which provides arbitrarily high protection against user-level MEV and LVR. V0LVER is the first AMM to align the incentives of the three, typically competing, entities in AMMs; the user, the pool, and the block producer. This is done by ensuring that at all times, a block producer is incentivized to move the pool to the price maximizing LVR. When the block producer chooses a price, the block producer is forced to assert this is correct, a technique introduced in [7]. Unfortunately, the protocol in [7] gives the block producer total power to extract value from users, due to order information being revealed to the block producer before it is allocated a trading price in the blockchain. To address this, V0LVER is built on an encrypted mempool. Modern cryptographic tools allow us to encrypt the mempool using zero-knowledge based collateralized commit-reveal protocols [8, 9, 10, 11],

¹near-0 Extractable Value and Loss-Versus-Rebalancing \rightsquigarrow V0LVER

delay encryption [12, 13] and/or threshold encryption [14]. We assume the existence of such a mempool within which all sensitive order information is hidden until the order has been committed a price against the AMM. Given these encrypted orders, we demonstrate that a block producer forced to show liquidity to such an order maximizes her own utility by showing liquidity centred around the external market price (bid below the price and offered above the price).²

As such, the external market price is the price point maximizing the block producers LVR extraction (due to the replicated LVR protection of [7]), around which profit is maximized when forced to trade against some (varying) percentage of indistinguishable orders. This strictly incentivizes block producers to move the price of a V0LVER pool to the external market price. This provides users with an AMM where the expected trade price in the presence of arbitrageurs is always the external market price, excluding fees, and the LVR against the pool is minimized when these arbitrageurs are competing. Although batching orders against AMM liquidity has been proposed as a defense against LVR [16], naively batching orders against an AMM still allows a block producer to extract LVR by censoring user orders. In V0LVER, block producers are effectively forced to immediately repay LVR, while being incentivized to include order commitments in the blockchain and allocate liquidity to these orders through the AMM.

1.2. Organisation of the Paper

Section 2 provides an overview of existing work on LVR protection the use of encrypted mempools for order submission. Section 3 provides the terminology, definitions, and existing AMM protocols needed to introduce and formally reason about V0LVER. Section 4 outlines the V0LVER protocol, first introducing the protocol model in Section 4.1. This is followed by an exhaustive description of each of the states and actions possible within the V0LVER protocol in Section 4.2, culminating in a unified description of V0LVER in Section 4.3. Section 5 details the properties of V0LVER, proving that V0LVER does indeed prevent MEV at the user-/order-level, while providing arbitrarily high LVR protection for the liquidity providers. Section 6 discusses some important considerations for the deployment of V0LVER. The paper concludes in Section 7.

2. Related Work

As the phenomenon of LVR has only recently been identified, there are only two academic papers on the subject of LVR protection [17, 7] to the best of our knowledge, with no work protecting against both LVR and user-level MEV.

In [17], the AMM must receive the price of a swap from a trusted oracle before users can interact with the pool. Such sub-block time price data requires centralized sources which are prone to manipulation, or require the active participation of AMM representatives, a contradiction of the passive nature of AMMs and their liquidity providers. We see this as an unsatisfactory dependency for DeFi protocols.

²This holds true in many CFMMs, including the famous Uniswap V2 protocol [15]

Our work is based on some of the techniques of the Diamond protocol as introduced in [7]. The Diamond protocol requires block producers to effectively attest to the final price of the block given the orders that are to be proposed to the AMM within the block. This technique requires the block producer to know exactly what orders are going to be added to the blockchain. This unfortunately gives the block producer total freedom to extract value from users submitting orders to the AMM. With V0LVER, we address this issue while keeping the LVR protection guarantees of Diamond.

Encrypting the transaction mempool using threshold encryption controlled by a committee has been proposed in [14] and applied in [18]. In [18], a DEX involving an AMM and based on frequent batch auctions [19] is proposed. This DEX does not provide LVR resistance, and incentivizes transaction censorship when a large LVR opportunity arises on the DEX. This is protected against in V0LVER.

3. Preliminaries

This section introduces the key terminology and definitions needed to understand LVR, and the proceeding analysis. In this work we are concerned with a single swap between token x and token y . We use x and y subscripts when referring to quantities of the respective tokens. The external market price of a swap is denoted by ϵ , with the price of a swap quoted as the quantity of token x per token y .

3.1. Constant Function Market Makers

A CFMM is characterized by *reserves* $(R_x, R_y) \in \mathbb{R}_+^2$ which describes the total amount of each token in the pool. The price of the pool is given by *pool price function* $P : \mathbb{R}_+^2 \rightarrow \mathbb{R}$ taking as input pool reserves (R_x, R_y) . $P()$ has the following properties:

- (a) $P()$ is everywhere differentiable, with $\frac{\partial P}{\partial R_x} > 0$, $\frac{\partial P}{\partial R_y} < 0$.
- (b) $\lim_{R_x \rightarrow 0} P = 0$, $\lim_{R_x \rightarrow \infty} P = \infty$, $\lim_{R_y \rightarrow 0} P = \infty$, $\lim_{R_y \rightarrow \infty} P = 0$. (1)
- (c) If $P(R_x, R_y) = p$, then $P(R_x + cp, R_y + c) = p$, $\forall c > 0$.

These are typical properties of price functions. Property (a) states the price of y is increasing in the number of x tokens in the pool and decreasing in the number of y tokens. Property (b) can be interpreted as any pool price value is reachable for a fixed R_x , by changing the reserves of R_y , and vice versa. Property (c) states that adding reserves to a pool in a ratio corresponding to the current price of the pool does not change the price of the pool. These properties trivially hold for the Uniswap V2 price function of $\frac{R_x}{R_y}$, and importantly allow us to generalize our results to a wider class of CFMMs.

For a CFMM, the *feasible set of reserves* C is described by:

$$C = \{(R_x, R_y) \in \mathbb{R}_+^2 : f(R_x, R_y) = k\} \quad (2)$$

where $f : \mathbb{R}_+^2 \rightarrow \mathbb{R}$ is the pool invariant and $k \in \mathbb{R}$ is a constant. The pool is defined by a smart contract which allows any player to move the pool reserves from the current reserves

$(R_{x,0}, R_{y,0}) \in C$ to any other reserves $(R_{x,1}, R_{y,1}) \in C$ if and only if the player provides the difference $(R_{x,1} - R_{x,0}, R_{y,1} - R_{y,0})$.

Whenever an arbitrageur interacts with an AMM pool, say at time $t + 1$ with reserves $(R_{x,t+1}, R_{y,t+1})$, we assume as in [4] that the arbitrageur always moves the pool reserves to a point which maximizes arbitrageur profits, exploiting the difference between the pool reserves from the previous block at time t , and the external market price at time $t + 1$, denoted ϵ_{t+1} . Therefore, under this assumption, the LVR between two blocks B_t and B_{t+1} where the reserves of the AMM at the end of B_t are $(R_{x,t}, R_{y,t})$ and the external market price when creating block B_{t+1} is ϵ_{t+1} is:

$$R_{x,t} - R_{x,t+1} + (R_{y,t} - R_{y,t+1})\epsilon_{t+1}. \quad (3)$$

In this paper, we consider only the subset of CFMMs in which, given the LVR extracted in block B_{t+1} corresponds to reserves $(R_{x,t+1}, R_{y,t+1})$, $P(R_{x,t+1}, R_{y,t+1}) = \epsilon_{t+1}$. This holds for Uniswap V2 pools, among others.

3.2. LVR-resistant AMM

We provide here an overview of Diamond [7], an AMM protocol which has been proven to provide arbitrarily high LVR protection under block producer competition to retain LVR. We also explain how Diamond manages to retain LVR from the block producers. In VOLLVER, we adapt these features for use on an encrypted transaction mempool, necessary to further provide MEV-protection to user orders. This adaption is explained in Section 4.

A Diamond pool Φ is described by reserves (R_x, R_y) , a pricing function $P()$, a pool invariant function $f()$, an *LVR-rebate parameter* $\beta \in (0, 1)$, and *conversion frequency* $T \in \mathbb{N}$. The authors also define a *corresponding CFMM pool* of Φ , denoted $CFMM(\Phi)$. $CFMM(\Phi)$ is the CFMM pool with reserves (R_x, R_y) whose feasible set is described by pool invariant function $f()$ and pool constant $k = f(R_x, R_y)$. Conversely, Φ is the *corresponding VOLLVER pool* of $CFMM(\Phi)$. The authors note that $CFMM(\Phi)$ changes every time the Φ pool reserves change. This is due to the expected increase in pool reserve value in a Diamond pool compared to its corresponding CFMM pool over any time period. Specifically, for any time t , and Diamond pool reserves $(R_{x,t}, R_{y,t})$ at that time, $E(f(R_{x,t+1}, R_{y,t+1})) > f(R_{x,t}, R_{y,t})$. The protocol progresses in blocks, with one reserve update possible per block.

For an arbitrageur wishing to move the price of $CFMM(\Phi)$ to p from starting reserves $(R_{x,0}, R_{y,0})$, let this require $\Delta_y > 0$ tokens to be added to $CFMM(\Phi)$, and $\Delta_x > 0$ tokens to be removed from $CFMM(\Phi)$. The same price in Φ is achieved by the following process:

1. Adding $(1 - \beta)\Delta_y$ tokens to Φ and removing $(1 - \beta)\Delta_x$ tokens.
2. Removing $\delta_x > 0$ tokens such that:

$$P(R_{x,0} - (1 - \beta)\Delta_x - \delta_x, R_{y,0} + (1 - \beta)\Delta_y) = p. \quad (4)$$

These δ_x tokens are added to the *vault* of Φ .

Vault tokens are periodically re-entered into Φ through a vault conversion process (effectively an auction process), where the tokens being re-added are in a ratio which approximates the external market price at the time.

When a vault conversion process takes place, all tokens are of the same denomination. Half of the tokens in the vault are auctioned among blockchain participants, with bids denominated in the other pool token. Under enough competition, the expected revenue of such an auction approximates the true value of the tokens being sold. We assume such competition in this paper. With this, the revenue of a vault conversion is approximately the same value as the remaining vault tokens. As such, the ratio of vault tokens to auction revenue tokens represent the true value of the token swap, and can then be added back into the pool.

The main result of [7] is the proving that given a block producer interacts with Φ when the LVR parameter is β , and there is an LVR opportunity of LVR in $CFMM(\Phi)$, the maximum LVR in Φ is $(1 - \beta)LVR$. This result is stated formally therein as follows:

Theorem 1. *For a CFMM pool $CFMM(\Phi)$ with LVR of $L > 0$, the LVR of Φ , the corresponding pool in Diamond, has expectancy of at most $(1 - \beta)L$.*

In this paper, we use the same base functionality of Diamond to restrict the LVR of block producers. Given a block producer wants to move the price of $CFMM(\Phi)$ to some price p to extract maximal LVR LVR , the maximal LVR in Φ of $(1 - \beta)LVR$ is also achieved by moving the price to p . An important point to note about applying LVR rebates as done in [7], is that directly after tokens are placed in the vault, the pool constant drops. This must be considered when calculating the profitability of an arbitrageur extracting LVR from a Diamond pool. We do this when analyzing the profitability of V0LVER in Section 5. Importantly, tokens are eventually re-added to the pool, and over time the expected value of the pool constant is increasing, as demonstrated experimentally in [7].

4. Protocol Description

We now outline the model in which we construct V0LVER, followed by a detailed description of V0LVER.

4.1. Model

In this paper we consider a blockchain in which all transactions are attempting to interact with a single V0LVER pool between tokens x and y . For ease of terminology, we assume block producers execute any and all arbitrage opportunities that exist between the external markets, and the V0LVER pool. In reality, this arbitrage can be performed by arbitrageurs who pay block producers for the right to execute V0LVER transactions, although it is the block producer who profits from this arbitrage given competition among arbitrageurs.

1. A transaction submitted by a player for addition to the blockchain while observing blockchain height H , is finalized in a block of height at most $H + T$, for some known $T > 0$.
2. The token swap has an external market price ϵ , which follows a Martingale process.
3. There exists a population of arbitrageurs able to frictionlessly trade at external market prices, who continuously monitor and interact with the blockchain.

4. Encrypted orders are equally likely to buy or sell tokens at the external market price, distributed symmetrically around the external market price.
5. Users submit market orders, specifying an amount of tokens to sell only.³

4.2. Protocol Building Blocks

This section outlines the terminology and functionalities used in V0LVER, a commit-reveal AMM protocol. This section is intended as a reference point to understand the core V0LVER protocol. Specifically, we describe the possible transactions in V0LVER, the possible states that V0LVER orders/order commitments can be in, and the possible actions of block producers. As in the protocol of Section 3.2, a V0LVER pool Φ with reserves (R_x, R_y) is defined with respect to a CFMM pool, denoted $CFMM(\Phi)$, with reserves (R_x, R_y) , a pricing function $P()$ under the restrictions of Section 3.1, and a pool invariant function $f()$.

4.2.1. Allocation Pools.

Orders in V0LVER are intended to interact with the AMM pool with some delay due to the commit-reveal nature of the orders. Therefore, we need to introduce the concept of allocated liquidity (liquidity set aside specifically for an order, or set of orders) to be used when orders eventually get revealed. To do this, we define an *allocation pool*. For orders of size either $size_x$ or $size_y$ known to be of maximum size max_x or max_y , the allocation pool consists of (λ_x, λ_y) tokens such that:

$$f(R_x, R_y) = f(R_x + max_x, R_y - \lambda_y) = f(R_x - \lambda_x, R_y + max_y). \quad (5)$$

For a given set of user orders and a corresponding allocation pool, let the total tokens being sold by the user orders be δ_x and δ_y , with $\delta_x > \delta_y P(R_x, R_y)$. That is, there are more token x being sold by users than the token y required to match user orders against each other at the pool price $P(R_x, R_y)$. This is an imbalance.

Such an imbalance requires some additional Δ_y tokens from the allocation pool to satisfy the imbalance. The execution price p_e of these orders is such that $(\delta_y + \Delta_y)p_e = \delta_x$, and must satisfy:

$$f(R_x, R_y) = f(R_x + (\delta_x - \delta_y p_e), R_y - \Delta_y). \quad (6)$$

With these two restrictions, we can solve for Δ_y and p_e given our specific pool pricing and invariant functions.⁴ An example of batch settlement against an allocation pool with a Uniswap V2 pool as the corresponding CFMM pool is provided at the end of Section 4.

These restrictions for calculating the execution price and tokens to be removed from the allocation pool are not defined with respect to the tokens in the allocation pool. However, by definition of the allocation pool reserves, there are sufficient tokens in the allocation pool to handle any allowable imbalance (anything up to max_x or max_y).

³We omit a description of how to execute limit orders against allocation pools for ease of notation, leaving a specification of limit order execution as an implementation exercise. As long as limit orders follow the same size restrictions as specified in this paper, the properties of V0LVER outlined in Section 5 should not change.

⁴If $\delta_x < \delta_y P(R_x, R_y)$, we must remove Δ_x tokens from the allocation pool with $\delta_y p_e = \delta_x + \Delta_x$ satisfying $f(R_x, R_y) = f(R_x - \Delta_x, R_y + (\delta_y - \frac{\delta_x}{p_e}))$

4.2.2. Transaction Specifications.

There are three types of transaction in our protocol. To define these transactions, we need an *LVR rebate function* $\beta : \{0, 1, \dots, Z\} \rightarrow [0, 1]$. It suffices to consider $\beta(\cdot)$ as a strictly decreasing function with $\beta(Z) = 0$.

1. **Order.** These are straightforward buy or sell orders indicating a limit price⁵, size, and direction to be traded. Without loss of generality, all orders in our system are executable.
2. **Order commitment transaction (OCT).** These are encrypted orders known to be collateralized by either max_x or max_y tokens. The exact size, direction, price, and sender of an OCT sent from player P_i is hidden from all other players. This is possible using anonymous ZK proofs of collateral such as used in [10, 11, 8]), which can be implemented on a blockchain in conjunction with a user-lead commit-reveal protocol, delay encryption scheme [12, 13] or threshold encryption scheme [14, 18]. OCTs can be added (inserted) to the blockchain at any time.
3. **Update transaction.** These transactions are executed in a block before any unallocated OCT can be allocated liquidity (see Figure 2). Let the current block height be H . Update transactions take as input an *allocation block height* $H_a \leq H$, and pool price p . Given an allocation block height of H'_a in the previous update transaction, valid update transactions require $H_a > H'_a$. All of the unallocated OCTs in blocks $[H'_a + 1, \dots, H_a]$ are then considered as allocated. For any update transaction, we denote by $T_a \in \mathbb{Z}_{\geq 0}$ the number of OCTs being allocated.

Given update transaction inputs of (H_a, p) , implied price of the VOLVER pool Φ is adjusted to p . If this requires $(\Delta_x, \Delta_y) \in \mathbb{R}^2$ to be added to the reserves of $CFMM(\Phi)$, the block producer provides $(1 - \beta(H - H_a))(\Delta_x, \Delta_y)$ to Φ , as is done in Section 3.2.

The producer must then deposit $(T_a \beta(H - H_a) max_y p, T_a \beta(H - H_a) \frac{max_x}{p})$ to an *allocation pool* denoted Φ_{H_a} , with $(T_a(1 - \beta(H - H_a)) max_y p, T_a(1 - \beta(H - H_a)) \frac{max_x}{p})$ being added to Φ_{H_a} from the AMM reserves. As such, the allocation pool contains $(T_a max_y p, T_a \frac{max_x}{p})$ tokens in total.

In other words, if the block producer wants to move the reserves of a VOLVER pool Φ to a particular price p , the amount of tokens extractable from Φ is $(1 - \beta(H - H_a))$ of the tokens extractable by moving the reserves of the corresponding CFMM pool, $CFMM(\Phi)$, to p .

Then, if an allocation pool requires up to $(T_a max_y p, T_a \frac{max_x}{p})$ tokens to trade with orders corresponding to the T_a allocated OCTs, the block producer is forced to provide $\beta(H - H_a)$ of the tokens in the pool, with starting bid and offer prices equal to the pool price set by the block producer. This is used to incentivize the block producer to always choose a pool price equal to the external market price.

4.2.3. Block Producer Action Set.

Every block, a block producer has four possible actions to perform on OCTs and their orders. Orders in our system are batch-settled with other orders allocated at the same time, and against the liquidity in the respective allocation pool.

1. Insert OCTs to the blockchain.

2. Allocate inserted OCTs. For a block producer adding a block at height H to allocate any number (including 0) inserted OCTs with inserted height of at most H_i , the block producer must:
 - a) Submit an update transaction with inputs $(H_a = H_i, p)$, for some $p > 0$.
 - b) Allocate all unallocated OCTs with inserted height less than or equal to H_i .
3. Reveal order corresponding to an allocated OCT. When a decrypted order corresponding to an OCT at height H_a is finalized on the blockchain within T blocks after the corresponding OCT is allocated, it is considered revealed.
4. Execute revealed orders. T blocks after OCTs are allocated, any corresponding revealed orders are executed at a single clearing price for orders allocated at the same time. The final tokens in the allocation pool are redistributed proportionally to the allocating block producer and VOLVER reserves.

4.3. Protocol Outline

Our protocol can be considered as two sub-protocols, a *base protocol* proceeding in rounds corresponding to blocks in the blockchain (see Figure 2), and an *allocation protocol* (Figure 3). As the blockchain progresses through the base protocol, at all heights $H > 0$, block producers have two key choices. The first is how many OCTs in the mempool to insert into the blockchain. The second is whether or not to send an update transaction.

There are two scenarios for an update transaction with inputs (H_a, p) and block height of the previous update transaction H'_a . Either $T_a = 0$ or $T_a > 0$. If $T_a = 0$, the update transaction is equivalent to an arbitrageur operation on a Diamond pool with LVR-rebate parameter of $\beta(H - H_a)$ (see Section 3.2). If $T_a > 0$, the arbitrageur must also deposit $(T_a\beta(H - H_a))max_{yp}, T_a\beta(H - H_a)\frac{max_x}{p}$ to the H_a allocation pool Φ_{H_a} , with $(T_a(1 - \beta(H - H_a))max_{yp}, T_a(1 - \beta(H - H_a))\frac{max_x}{p})$ being added to Φ_{H_a} from the AMM reserves.

After an allocation pool is created for allocated OCTs $\{oct_1, \dots, oct_{T_a}\}$, the orders corresponding to $\{oct_1, \dots, oct_{T_a}\}$ can be revealed for up to T blocks. This is sufficient time for any user whose OCT is contained in that set to reveal the order corresponding to the OCT. To enforce revelation, tokens corresponding to unrevealed orders are burned (unable to be executed or withdrawn). After all orders have been revealed, or T blocks have passed, any block producer can execute revealed orders against the allocation pool at a clearing price which maximizes volume traded. Specifically, given an array of orders ordered by price, a basic smart-contract can verify that a proposed clearing price maximizes volume traded, as is done in [10].

The final tokens in the allocation pool are redistributed to the allocating block producer and VOLVER reserves. Adding these tokens directly to the pool (and not the vault as in the protocol from Section 3.2) allows the pool to update its price to reflect the information of the revealed orders.

4.3.1. Example: Executing Orders Against the Allocation Pool.

This example details how one would batch execute orders against an allocation pool Φ_{H_a} , replicating liquidity in the corresponding CFMM pool of Φ , that is, $CFMM(\Phi)$. Let the total tokens in the VOLVER pool Φ before allocation be (R_x, R_y) , with $CFMM(\Phi)$ the Uniswap V2

pool. As such, $P(R_x, R_y) = \frac{R_x}{R_y} = p_0$. Let the allocated OCTs be selling δ_x and δ_y tokens, with $\delta_y p_0 < \delta_x$. That is, there is an imbalance of tokens at p_0 , with more token x being sold than token y at the price p_0 . We will now derive the execution price p_e for these orders.

Given $\delta_y p_0 < \delta_x$, this means some Δ_y tokens from the allocation pool are required to fill the imbalance. Firstly, given the execution price is p_e , we know $(\delta_y + \Delta_y)p_e = \delta_x$. That is, the execution price equals $\frac{\delta_x}{\delta_y + \Delta_y}$. Secondly, the amount of x tokens added to the allocation pool is $\delta_x - \delta_y p_e$. As the allocation pool provides liquidity equivalent to batch executing the orders against CFMM(Φ), this means the pool invariant function would remain constant if those tokens were traded directly with CFMM(Φ). Specifically:

$$R_x R_y = (R_x + (\delta_x - \delta_y p_e))(R_y - \Delta_y). \quad (7)$$

From our first observation, we know $\Delta_y = \frac{\delta_x}{p_e} - \delta_y$, which we can rewrite as $\frac{1}{p_e}(\delta_x - \delta_y p_e)$. This gives:

$$R_x R_y = R_x R_y + R_y(\delta_x - \delta_y p_e) - R_x \frac{1}{p_e}(\delta_x - \delta_y p_e) - \frac{1}{p_e}(\delta_x - \delta_y p_e)^2. \quad (8)$$

Cancelling the first term on both sides, and dividing by $(\delta_x - \delta_y p_e) > 0$ gives:

$$0 = R_y - R_x \frac{1}{p_e} - \frac{1}{p_e}(\delta_x - \delta_y p_e). \quad (9)$$

Isolating p_e , we get:

$$p_e = \frac{R_x + \delta_x}{R_y + \delta_y}. \quad (10)$$

5. Protocol Properties

The goal of this section is to show that the expected execution price of any user order is the external market price when the order is allocated, excluding at most impact and fees. Firstly, note that an update transaction prior to allocation moves the pool reserves of a V0LVER pool identically to an LVR arbitrage transaction in Section 3.2. If $T_a = 0$, from [7] we know the block producer moves the pool price to the max LVR price which is the external market price, and the result follows trivially.

Now instead, assume $T_a > 0$. Let the reserves of a V0LVER pool Φ before the update transaction be $(R_{x,0}, R_{y,0})$. Given an external market price of ϵ , from Section 3.1 we know the max LVR occurs by moving the pool reserves to some $(R_{x,m}, R_{y,m})$ with $\frac{R_{x,m}}{R_{y,m}} = \epsilon$. Without loss of generality, let $\frac{R_{x,0}}{R_{y,0}} < \frac{R_{x,m}}{R_{y,m}}$. Let the block producer move the pool price to p corresponding to reserves in the corresponding CFMM pool of $(R_{x,p}, R_{y,p})$. For a non-zero $\beta(\cdot)$, this means the tokens in Φ not in the vault (as per the protocol in Section 3.2) are $(R'_{x,p}, R'_{y,p}) = (bR_{x,p}, bR_{y,p})$ for some $b < 1$. This is because some tokens in Φ are removed from the pool and placed in the vault, while maintaining $\frac{R'_{x,p}}{R'_{y,p}} = p$.

There are three payoffs of interest here. For these, recall that by definition of the external market price, the expected imbalance of an encrypted order in our system is 0 at the external market price.

1. **Payoff of block producer vs. AMM pool (update transaction arbitrage):** $(1 - \beta())(R_{x,0} - R_{x,p} + (R_{y,0} - R_{y,p})\epsilon)$.
2. **Payoff of block producer vs. users:** Against a block producer's own orders, the block producer has 0 expectancy. Against other player orders, the block producer strictly maximizes her own expectancy when $(R_{x,p}, R_{y,p}) = (R_{x,m}, R_{y,m})$. This can be seen by considering any other choice of reserves corresponding to a price not equal to ϵ . In such cases, because of our expectation that the imbalance of user orders is 0 only at ϵ , the block producer is either offering below ϵ against an expected buy imbalance, or bidding above ϵ against an expected sell imbalance.
3. **Payoff of user orders vs. AMM pool:** Consider a set of allocated orders executed against the allocation pool, corresponding to the pool receiving δ_x and paying δ_y tokens. By definition of the allocation pool, this (δ_x, δ_y) is the same token vector that would be applied to the CFMM pool with reserves $(bR_{x,p}, bR_{y,p})$ if those orders were batch executed directly against the CFMM. Let these new reserves be $(bR_{x,1}, bR_{y,1})$. Thus the profit of these orders is $b(1 - \beta())(R_{x,p} - R_{x,1} + (R_{y,p} - R_{y,1})\epsilon)$.

Optimal strategy for block producer

Let the block producer account for $\alpha \in [0, 1]$ of the user orders executed against the allocation pool (Payoff 3). The maximum payoff of the block producer against the AMM pool is the maximum of the sum of arbitrage profits (Payoff 1) and profits of block producer orders executed against the pool (α of Payoff 3). Thus, the utility function to be maximized for the block producer is:

$$(1 - \beta())(R_{x,0} - R_{x,p} + (R_{y,0} - R_{y,p})\epsilon) + \alpha(b(1 - \beta())(R_{x,p} - R_{x,1} + (R_{y,p} - R_{y,1})\epsilon)). \quad (11)$$

By taking $\alpha b(1 - \beta())(R_{x,0} - R_{x,p} + (R_{y,0} - R_{y,p})\epsilon)$ from the first term, and adding it to the second term, we get

$$(1 - \alpha b)(1 - \beta())(R_{x,0} - R_{x,p} + (R_{y,0} - R_{y,p})\epsilon) + \alpha b(1 - \beta())(R_{x,0} - R_{x,1} + (R_{y,0} - R_{y,1})\epsilon). \quad (12)$$

We know the second term is maximized for $(R_{x,1}, R_{y,1}) = (R_{x,m}, R_{y,m})$, as this corresponds to LVR. Similarly, the first term is also maximized for $(R_{x,p}, R_{y,p}) = (R_{x,m}, R_{y,m})$. Given $(R_{x,p}, R_{y,p}) = (R_{x,m}, R_{y,m})$, block producers have negative expectancy for $\alpha > 0$, as this reduces the probability that $(R_{x,1}, R_{y,1}) = (R_{x,m}, R_{y,m})$ by increasing the likelihood of an imbalance at p . As such, block producers are strictly incentivized to set $p = \epsilon$, and not submit OCTs to the protocol ($\alpha = 0$) for Payoffs 1 and 3. Now consider the payoff for the block producer against user orders (Payoff 2). We have already argued in the description of Payoff 2 that this is maximized with $(R_{x,p}, R_{y,p}) = (R_{x,m}, R_{y,m})$.

Therefore, moving the pool price p to ϵ is a dominant strategy for the block producer. Given this, we can see that the expected execution price for a client is ϵ excluding impact and fees, with impact decreasing in expectancy in the number of orders allocated. The payoff of the VOLVER pool against the block producer via the update transaction is $(1 - \beta())$ of the LVR

in the corresponding CFMM pool (as by definition, $(R_{x,m}, R_{y,m})$ correspond to the reserves maximizing LVR in the corresponding CFMM pool). Furthermore, the payoff of the V0LVER pool against other user orders is at least 0, as these trades happen on the allocation pool centred around (bid below, and offered above) ϵ .

5.1. Minimal LVR

In the previous section, it is demonstrated that user-level MEV is prevented in V0LVER, with users trading at the external market price in expectancy, excluding impact and fees. However, we have thus far only proved that LVR in a V0LVER pool is $(1 - \beta())$ of the corresponding CFMM pool. As in [7], under competition among block producers, the LVR rebate function has a strong Nash equilibrium at $\beta(0)$, meaning LVR is also minimized.

To see this, we can use a backwards induction argument. Consider the first block producer allowed to send an update transaction with $\beta(H - H_a) = 0$ for a block at height H (meaning $H_a = H'_a + 1$). This block producer can extract all of the LVR, and is required to provide no liquidity to the allocation pool. As LVR is arbitrage, all block producers do this.

A block producer at height $H - 1$ knows this. Furthermore, extracting $(1 - \beta((H - 1) - H_a)) > 0$ of the LVR has positive utility for all block producers, while trading with $\beta((H - 1) - H_a) > 0$ of allocated OCTs around the external market price also has a positive utility (Payoff 2 in Section 5). As such, sending an update transaction at height $H - 1$ is dominant. Following this argumentation, a block producer at height $H - i \geq H_a$ always sends an update transaction as they know the block producer at height $(H + 1) - i$ always sends an update transaction. This means the block producer at height $H'_a + 1$ always sends an update transaction $\forall H'_a$, which corresponds to an LVR rebate function value of $\beta(0)$ in equilibrium.

In reality, frictionless arbitrage against the external market price in blockchain-based protocols is likely not possible, and so LVR extraction has some cost. As such, the expected value for $\beta()$ may be less than $\beta(0)$. Deploying V0LVER, and analyzing $\beta()$ across different token pairs, and under varying costs for block producers makes for interesting future work.

6. Discussion

If a V0LVER pool allows an OCT to be allocated with $\beta() = 0$, V0LVER effectively reverts to the corresponding CFMM pool, with MEV-proof batch settlement for all simultaneously allocated OCTs, albeit without LVR protection for the pool. To see this, note that as $\beta() = 0$, the block producer can fully extract any existing LVR opportunity, without requiring a deposit to the allocation pool. As such, the expected price of the allocation pool is the external market price, with orders executed directly against the V0LVER reserves at the external market price, excluding fees and impact. Importantly, there is never any way for the block producer to extract any value from allocated orders. This is because the settlement price for an OCT is effectively set when it allocated, before any price or directional information is revealed about the corresponding order.

Allocation of tokens to the allocation pool has an opportunity cost for both the V0LVER pool and the block producer, requiring the user to pay a fee to both (or just the block producer, who would then be required to forward some of this fee to the pool). Users doing this anonymously is

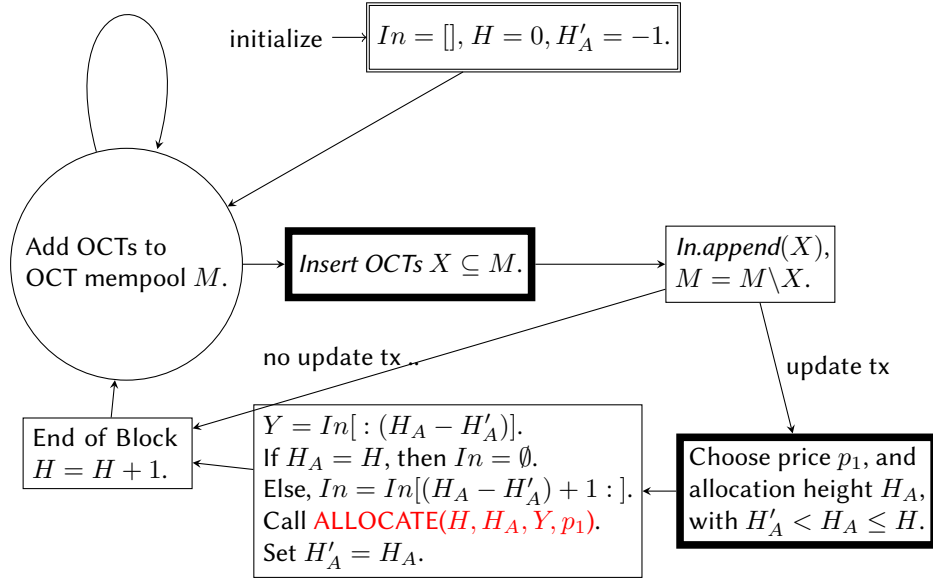


Figure 2: Flow of V0LVER protocol, excluding the allocation protocol (see Figure 3 for the allocation protocol). The double-border rectangle is the initialization state, thin single-bordered rectangles are state updates on-chain, while thick-bordered rectangles are block producer decisions/computations off-chain. The circle state is controlled by the network. Note that In , the array of inserted but unallocated OCTs, is an ordered array of sets of OCTs. For $1 < a \leq \text{len}(In)$, $In[: a]$ returns an ordered sub-array of In elements at indices $[1, \dots, a]$, while $In[a :]$ returns an ordered sub-array of In elements at indices $[a, \dots, \text{len}(In)]$.

important to avoid MEV-leakage to the block producer. One possibility is providing an on-chain verifiable proof of membership to a set of players who have bought pool credits, where a valid proof releases tokens to cover specific fees, as in [11, 10]. Another possibility is providing a proof to the block-producer that the user has funds to pay the fee, with the block-producer paying the fee on behalf of the user. A final option, based on threshold encryption, is creating a state directly after allocation before any more allocations are possible, in which allocated funds are either used or de-allocated. All of these proposals have merits and limitations, but further analysis of these are beyond the scope of this work.

7. Conclusion

V0LVER is an AMM based on an encrypted transaction mempool in which LVR and MEV are protected against. V0LVER aligns the incentives of users, passive liquidity providers and block producers. This is done by ensuring the optimal block producer strategy under competition among block producers simultaneously minimizes LVR against passive liquidity providers and MEV against users.

Interestingly, the exact strategy equilibria of V0LVER depend on factors beyond instantaneous token maximization for block producers. This is due to risks associated with liquidity provision

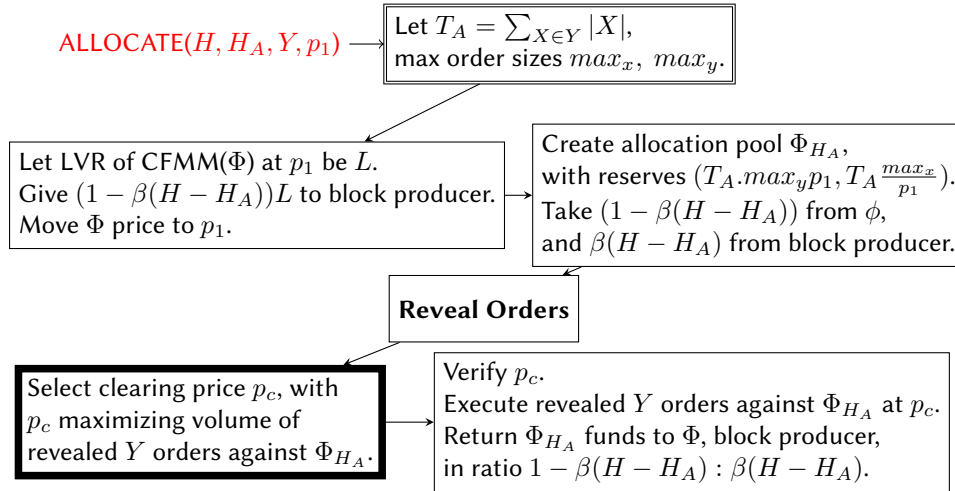


Figure 3: Flow of allocation protocol for VOLLVER pool ϕ , initialized every time the ALLOCATE() function is called in Figure 2. The Reveal Orders state happens by some block after height H . As in the previous figure, the double-border rectangle is the initialization state, thin single-border rectangles are state updates on-chain, while thick-bordered rectangles are block producer decisions/computations off-chain.

and arbitrage costs. On one hand, allocating OCTs after setting the pool price to the external market price, and providing some liquidity to OCTs around this price should be positive expectancy for block producers. Similarly, increasing the number of OCTs should also reduce the variance of block producer payoffs. On the other hand, there are caveats in which all OCTs are informed and uni-directional. Analyzing these trade-offs for various risk profiles and trading scenarios makes for further interesting future work.

8. Acknowledgements

This paper is part of a project that has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement number 814284, and is supported by the AEI-PID2021-128521OB-I00 grant of the Spanish Ministry of Science and Innovation.

References

- [1] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, A. Juels, Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges, <https://arxiv.org/abs/1904.05234>, 2019. arXiv:1904.05234, accessed: 19/04/2023.
- [2] K. Qin, L. Zhou, A. Gervais, Quantifying Blockchain Extractable Value: How dark is the forest?, in: 2022 IEEE Symposium on Security and Privacy (SP), 2022, pp. 198–214. doi:10.1109/SP46214.2022.9833734.
- [3] Flashbots, <https://explore.flashbots.net>, 2022. Accessed: 11/10/2022.

- [4] J. Millionis, C. C. Moallemi, T. Roughgarden, A. L. Zhang, Quantifying Loss in Automated Market Makers, in: Proceedings of the 2022 ACM CCS Workshop on Decentralized Finance and Security, DeFi'22, Association for Computing Machinery, New York, NY, USA, 2022, p. 71–74. URL: <https://doi.org/10.1145/3560832.3563441>. doi:10.1145/3560832.3563441.
- [5] A. Capponi, R. Jia, The Adoption of Blockchain-based Decentralized Exchanges, <https://arxiv.org/abs/2103.08842>, 2021. Accessed: 10/02/2023.
- [6] @thickeythot, <https://dune.com/thickeythot/uniswap-markouts>, 2022. Accessed: 10/02/2023.
- [7] C. McMenamin, V. Daza, B. Mazonra, Diamonds are Forever, Loss-Versus-Rebalancing is Not, <https://arxiv.org/abs/2210.10601>, 2022. doi:10.48550/ARXIV.2210.10601, accessed: 04/03/2023.
- [8] C. McMenamin, V. Daza, Dynamic, private, anonymous, collateralizable commitments vs. mev, <https://arxiv.org/abs/2301.12818>, 2022. doi:10.48550/ARXIV.2301.12818, accessed: 31/03/2023.
- [9] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, M. Virza, Zerocash: Decentralized Anonymous Payments from Bitcoin, in: 2014 IEEE Symposium on Security and Privacy, IEEE Computer Society, New York, NY, USA, 2014, pp. 459–474.
- [10] C. McMenamin, V. Daza, M. Fitz, P. O'Donoghue, FairTraDEX: A Decentralised Exchange Preventing Value Extraction, in: Proceedings of the 2022 ACM CCS Workshop on Decentralized Finance and Security, DeFi'22, Association for Computing Machinery, New York, NY, USA, 2022, p. 39–46. URL: <https://doi.org/10.1145/3560832.3563439>. doi:10.1145/3560832.3563439.
- [11] Tornado Cash, <https://tornadocash.eth.link/>, 2023. Accessed: 31/03/2023.
- [12] J. Burdges, L. De Feo, Delay encryption, in: A. Canteaut, F.-X. Standaert (Eds.), Advances in Cryptology – EUROCRYPT 2021, Springer International Publishing, Cham, 2021, pp. 302–326.
- [13] J. H. Chiang, B. David, I. Eyal, T. Gong, Fairpos: Input fairness in proof-of-stake with adaptive security, <https://eprint.iacr.org/2022/1442>, 2022. Accessed: 23/03/2023.
- [14] A. Asayag, G. Cohen, I. Grayevsky, M. Leshkowitz, O. Rottenstreich, R. Tamari, D. Yakira, Helix: A Fair Blockchain Consensus Protocol Resistant to Ordering Manipulation, IEEE Transactions on Network and Service Management 18 (2021) 1584–1597. doi:10.1109/TNSM.2021.3052038.
- [15] H. Adams, N. Zinsmeister, D. Robinson, Uniswap V2 Core, 2020. URL: <https://uniswap.org/whitepaper.pdf>.
- [16] G. Ramseyer, M. Goyal, A. Goel, D. Mazières, Batch exchanges with constant function market makers: Axioms, equilibria, and computation, <https://arxiv.org/abs/2210.04929>, 2022. doi:10.48550/ARXIV.2210.04929, accessed: 26/03/2023.
- [17] B. Krishnamachari, Q. Feng, E. Grippio, Dynamic Automated Market Makers for Decentralized Cryptocurrency Exchange, in: 2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2021, pp. 1–2. doi:10.1109/ICBC51069.2021.9461100.
- [18] Penumbra, <https://penumbra.zone/>, 2023. Accessed: 23/03/2023.
- [19] E. Budish, P. Cramton, J. Shim, The High-Frequency Trading Arms Race: Frequent Batch Auctions as a Market Design Response *, The Quarterly Journal of Economics 130 (2015) 1547–1621. URL: <https://doi.org/10.1093/qje/qjv027>. doi:10.1093/qje/qjv027.