FairMM: A Fast and Frontrunning-Resistant Crypto Market-Maker

Michele Ciampi[1], Muhammad Ishaq[2(✉)], Malik Magdon-Ismail[3], Rafail Ostrovsky[4], and Vassilis Zikas[2]

[1] The University of Edinburgh, Edinburgh, UK
michele.ciampi@ed.ac.uk
[2] Purdue University, West Lafayette, USA
{ishaqm,vzikas}@cs.purdue.edu
[3] Rensselaer Polytechnic Institute (RPI), Troy, USA
[4] University of California, Los Angeles (UCLA), Los Angeles, USA
rafail@cs.ucla.edu

Abstract. Frontrunning is a major problem in DeFi applications, such as blockchain-based exchanges. Albeit, existing solutions are not practical and/or they make external trust assumptions. In this work we propose a market-maker-based crypto-token exchange, which is both more efficient than existing solutions and offers provable resistance to frontrunning attack. Our approach combines in a clever way a game theoretic analysis of market-makers with new cryptography and blockchain tools to defend against all three ways by which an exchange might front-run, i.e., (1) reorder trade requests, (2) adaptively drop trade requests, and (3) adaptively insert (its own) trade requests. Concretely, we propose novel light-weight cryptographic tools and smart-contract-enforced incentives to eliminate reordering attacks and ensure that dropping requests have to be oblivious (uninformed) of the actual trade. We then prove that with these attacks eliminated, a so-called monopolistic market-maker has no longer incentives to add or drop trades. We have implemented and benchmarked our exchange and provide concrete evidence of its advantages over existing solutions.

Keywords: Front-running · Market maker · Blockchain · Fairness

1 Introduction

Since Bitcoin's introduction in 2008, *crypto-currencies* have become a significant market in global finance[1]. Several tools and platforms have been built to facilitate crypto-currency trading. The early exchanges e.g. Binance, Coinbase, Bittrex, etc. have two undesirable properties. First, they were *custodial*, meaning that traders transfer their assets to the exchange, and trading activity translates

[1] Market capitalization of approx. $2 trillion during all of 2021.

to updating the exchange's internal mapping of traders and their assets. Second, they were *order-book* based i.e. they only match buyers with sellers (collectively, traders), so trading halts when there are no sellers whose ask-prices match the buyer bid-prices. The prices are fully controlled by traders and therefore can be volatile. The later exchanges e.g. Uniswap, Balancer, Curve, etc.—collectively called Decentralized Exchanges or *DEXes*—are *non-custodial*, have their own inventory of assets, and use a market making (MM) algorithm to adjust prices. The latter type of exchanges, colloquially also known as *market makers*, leverage machine learning (ML) to increase liquidity along with additional desirable properties for the market maker (e.g., maximizing profit) or the market itself (e.g., stability by incentivizing traders to report true valuations).

A perennial problem with both types of exchanges (and traditional markets, too) is *frontrunning*, where an adversary reorders trades to gain a price advantage. For example, say a market maker (MM) is selling an asset t at \$100 each, and the pricing function is such that for each unit sold, the price increases by \$100. Say a trader P submits a trade T_P to buy one unit, a (possibly adversarial) MM A sees this pending trade and executes, before processing T_P, a different trade T_A to buy one unit, i.e. the adversary A "front-runs" the trader P. Executing T_A before T_P raises the price to \$200 for P, \$100 more than he would otherwise pay. Trades should be executed in the order they were submitted.

Frontrunning penalizes honest players, and also has a detrimental effect on the health of the underlying crypto-currency blockchain. As an example, adversarial bots might flood the blockchain with frontrunning orders when an opportunity for profit arises, with only few of these orders being executed. This flooding creates a denial of service (DoS) attack. The above effects of frontrunning are worsened in decentralized exchanges. E.g., when the exchange is implemented by a smart contract on a chain like Ethereum, e.g., Uniswap, frontrunning raises transaction fees (*gas price* on Ethereum) as traders compete to get their trades in first. And, in theory, it can also affect the security of the underlying blockchain. Indeed, in DEXes, where the trade ordering is affected by miners through transaction reordering, frontrunning might create incentives for forking as the miners are more likely to pursue a chain on which they make more profit.

Traditional markets mitigate frontrunning through legislation. However, such legislation is tricky to enforce as most frontrunning attacks do not leave an indisputable evidence, which would be necessary to apply fines. As such, several mature markets have embedded certain controlled forms of frontrunning in their allowed operations. The classical example of this principle is embodied in traditional stock exchange markets, where high-profile clients might get so called *privileged access*, allowing them to react faster to market changes. The above solution is unsatisfactory for crypto-currency markets. For starters, legislation lags behind and is typically not tailored to crypto-currencies, making deterrence by regulation much harder. But more importantly, the egalitarian nature of these assets makes preferential frontrunning an undesirable feature for the majority of its users. The main question addressed (to the affirmative) by our work is

Can we leverage the public-observability of blockchain ledgers together with cryptographic and machine learning tools to devise a practical frontrunning resistant market maker?

Informally, solving the above problem requires: (i) ensuring a strict ordering on the trades, and (ii) making sure the system is fast enough for high-frequency trading. The first requirement gets rid of frontrunning attack and the second ensures the solution's practicality. The first requirement can be further broken down into a) preventing traders from frontrunning each other and b) preventing MM from frontrunning. For private-state blockchains (e.g. ZCash, Dash, Monero, etc.), preventing traders from frontrunning is easy because the traders cannot see each other's trades (due to transaction privacy). However, the mainstream blockchains (e.g. Bitcoin, Ethereum, Cardano, etc.) are not private and preventing traders from frontrunning each other requires additional mechanisms.

Several such mechanisms have been proposed, e.g. commit-reveal, encryption, zk-rollups, speed-bumps/retro-active pricing, and commit secret sharing. However they all cause a slowdown of the system and are, therefore, in conflict with the second requirement (the system needs to be fast). Alternative approaches have proposed to simulate MM as a trusted third party (TTP) through secure multi-party computation (MPC), trusted execution environments (TEEs) and zero-knowledge proofs (ZKPs). These do solve the frontrunning problem, but once more, conflict with the second requirement and/or make additional trust assumptions (see also the related research section for a detailed comparison). Indeed, MPC and ZKPs are expensive cryptographic primitives that negatively affect speed of the system and, TEEs place additional demands on application hardware (and assumptions thereof). A viable solution needs to satisfy both of the above requirements, ideally without additional trust assumptions.

This work, FairMM, proposes a market maker (MM) that resolves the frontrunning problem using off-chain communication and inexpensive hash functions. This is done through a combination of techniques from cryptography, game theory, blockchain, and machine leaning for financial economics.

At a high level, FairMM operates as follows: Traders and MM communicate off-chain via secure communication channels (e.g. through TLS), traders form a queue, and a trade is processed as follows: 1) MM issues a ticket to the trader T_i at the front of the queue, this ticket is identified by a cryptographic hash and signed by the MM (think of the hash as a serial number), then, 2) trader T_i may decide to respond with trade (trade) or not trade ($\mathsf{no\_trade}$), 3) MM processes T_i's response and moves to the next trader T_{i+1}. The nature of this ticket hash (or serial number) is such that it incorporates the complete trading history up to that point. If the MM tries to talk to more than one trader at a time, all but one of the traders will get their trade incorporated into the trading history (trading history is linear, there is no way to keep all serial numbers valid without creating branches). MM posts the trading activity to a public bulletin board at regular intervals. The traders can read this bulletin board at any time and if they find any invalid tickets or that their ticket is missing, they can complain to a smart contract. This smart contract, established by the MM before its operation starts, locks a large collateral on behalf of MM. On a valid complaint, the complainant is rewarded (with a sufficiently large but less than the collateral amount) and MM loses all collateral. This ticketing mechanism is extremely efficient to compute.

The above ticketing mechanism already takes care of the worst-case scenario, i.e., reordering attacks. One could plug in any market making algorithm (to adjust asset prices) and obtain a reordering-resilient system. However, the MM can still drop trades, although it will be doing so without the knowledge of subsequent trades. We resolve this problem by carefully choosing a market maker—a *monopolistic profit seeking* market maker—that has economic incentive to not manipulate trading activity in this manner (e.g. by dropping trades obliviously of future trade requests). A monopolistic profit seeking MM uses trade requests as signals to determine where the true value of an asset lies (more buy trades $\implies$ true value of the asset is higher, more sell trades $\implies$ true value is lower). Its core principle is that, because a monopolistic profit seeking MM makes most profit when trading activity happens around the true value of an asset, it has no incentive to manipulate trading requests that would make the signals from trading activity less reliable. See Sect. 5 for formal discussion.

In addition to proving the security of FairMM, we have implemented and evaluated our design. We show that this design is extremely competitive. Concretely, we achieve a throughput of over 200 trades/minute. This is despite strict serialization of trade requests and the fact that we are running off-the-chain part on a relatively weak, consumer laptop (which communicates with an actual Ethereum node in test environment). These figures are about 50% higher than the maximum daily volume of Uniswap [23], arguably the most popular DEX and an order of magnitude higher than P2DEX [33], an order-book exchange implemented using MPC. We are also better than TEX [27], also an order-book exchange, which either does not support high frequency trading or is not frontrunning resilient. While Tesseract [25], another orderbook exchange, reports much higher throughput, it requires both trusted hardware and a consensus group assumption for its security guarantees. We, on the other had, require no such assumption.

In summary, our work makes the following contributions:

- We provide design of a non-custodial frontrunning resilient market-making crypto-currency exchange that does not require sophisticated cryptographic machinery (MPC, ZKPs, etc.), special hardware (TEEs) or additional assumptions (e.g. an additional consensus group in addition to the blockchain).
- We extract a useful abstraction—Σ-trade protocols—for asset exchanges that facilitates modular design of blockchain trading systems.
- We provide an instantiation of the system on Ethereum blockchain and demonstrate that it is very fast, and practical for real world applications.

1.1 Related Works

Market Makers for Crypto-token Markets. New emerging markets, e.g. prediction markets [8] or crypto-token markets, are typically thin and illiquid and often have to be bootstrapped through intelligent market makers to provide liquidity and price discovery [10]. A market maker algorithm aims at maximizing liquidity in the market and/or maximizing its own profit. The *zero-profit competitive market maker* model [1,4,9] considers multiple MMs that compete with each other by lowering their marginal profit to eliminate competition—such a system converges to a zero-profit. The *monopolist* market-maker, has been shown to provide greater liquidity than zero-profit competitive market makers [1,4,9,11]. We adopt the extension by Das [9] (cf. Sect. 5).

Fair Exchange and Blockchains. There is a large amount of literature on fair exchange including early MPC works [2,3,5,6,12], which has been re-ignited with the adoption of blockchains and cryptocurrencies [13,15–17,24,26]. Due to the relevance of these works to ours, we include a detailed review in the full version [35]. However, these works are not suitable for reuse in our design. Informally, the reason is that in our setting, fair exchange is a subroutine of the Market Maker (MM) protocol, and MM needs to know immediately whether a trade will settle or not on the blockchain. Therefore, we designed our own fair exchange protocol, a Σ-trade protocol, that we proved amenable to such composition.

Decentralized Exchanges. Popular decentralized exchanges e.g. Uniswap, Curve, Kyber, etc. [18–23,28,41] do not defend against frontrunning. To our knowledge, Tesseract by Bentov et al. [25] is the first work that addresses frontrunning in the crypto-currency space. It is orderbook based, custodial, and simulates a trusted third party (TTP) through trusted execution environments (TEEs). The assumption here is that since the exchange is a TTP, frontrunning does not happen. Since it relies on players to provide it with blockchain data, there is a check-pointing mechanism on trusted blocks, and if the exchange becomes unavailable, there is a *consensus group* of TEE backed nodes that can enforce/cancel transactions so that traders do not lose funds. In a similar vein, orderbook based P2DEX [33] by Baum et al. simulates TTP through outsourced MPC, their technique is similar to the work by Charanjit et al. [14] for traditional markets. TEX (Trustless Exchange) by Khalil et al. [27] is another orderbook exchange. It uses Zero Knowledge Proofs (ZKPs) for its guarantees. ZKPs are an expensive primitive and, in TEX, there is a trade-off as it either does not fully support high frequency trading or does not provide frontrunning resilience. In contrast to the above works, our construction is a *market maker*, it is *non-custodial*, does away with expensive primitives (MPC, ZKPs), additional requirements on hardware and/or additional check-pointing/consensus mechanisms, and provides frontrunning defense in a high frequency trading environment (as demonstrated by our detailed comparison with Uniswap in Sect. 6.3). Note however, that Tesseract supports cross-chain trading, our work does not.

Fairy by Stathakopoulou et al. [38] solve frontrunning for Byzantine Fault Tolerant (BFT) systems by augmenting Total Order Broadcast (TOB) protocols with *input causality* and *sender obfuscation*. They also require TEEs. Moreover, adapting this work to address frontrunning in crypto-currencies is non-trivial.

GageMPC [31] by Almashaqbeh et al. tackles privacy preserving auctions using non-interactive MPC (NIMPC). This work could be adapted into an exchange, but it is unclear whether it could handle high frequency trading. A^2MM by Zhou et al. [39] optimizes onchain swaps to *mitigate* frontrunning attacks. They study two point arbitrages for two assets. Their analysis holds assuming that all exchanges on a blockchain will be handled by A^2MM. This assumption is too strict for practical applications. Flashbots [36] and Gnosis Protocol V2 [37] both claim to resolve frontrunning. Flashbots requires strong trust (in the players to follow protocol) and is therefore not comparable to our work. Gnosis Protocol V2 claims that it will have a defense against frontrunning when it is built but currently there is no description of how it will be achieved. We refer to Chainlink 2.0 [34] whitepaper for details on existing techniques to achieve strict ordering of transactions. It also proposes a Fair Sequence Service (FSS) for Distributed Oracle Networks (DONs) that should solve this problem in general. However, exactly how such FSS will be implemented is not specified in the whitepaper.

There are some complementary works to ours which studies frontrunning. Flash Boys 2.0 [29] by Daian et al. give evidence that frontrunning is a serious problem on Ethereum. Bartoletti et al. [32] provide a theoretical framework to maximize miner extractable value (MEV), Sobol et al. [30] discuss frontrunning on proof of stake blockchains, and Zhou et al. [40] study sandwich attacks.

Next we dive into technical details of the paper, due to space constraints, we have provided relevant background in the full version [35].

2 Σ-Trade Protocols

A Σ-trade protocol Π is an interactive protocol run by a seller S and potentially many buyers $B_1, \ldots, B_m$ (seller S need not know m) where the exchange of tokens happens on blockchain E (We can think of E as the Ethereum blockchain).

Assume two tokens t_1 and t_2, each buyer wants to buy tokens of type t_2 in exchange of tokens of type t_1. Assume also that, each buyer B_i has an upper bound, denoted with $\mathtt{z}_i$, of type t_1 tokens that he can spend. The amount of t_2 tokens the buyer wants to buy is decided adaptively in the last round of interaction. A Σ-trade protocol Π consists of the following steps:

1. Each buyer B_i creates a smart contract $\mathtt{SC}_i$ on E that locks $\mathtt{z}_i$ tokens of type t_1 (more details on $\mathtt{SC}_i$ are provided later).
2. B_i and S exchange three off-chain messages. First, B_i sends his identities to the seller S. Note that B_i does not yet disclose his desired quantity t_2 tokens. In response, S proposes the exchange rate, `askedPrice`, for the tokens.
3. Let y be the quantity of tokens of type t_2 that the buyer wants to buy s.t. $y \cdot \mathtt{askedPrice} \leq \mathtt{z}_i$. If B_i agrees with `askedPrice`, then B_i sends a *certificate* c. This c can be used by S to invoke $\mathtt{SC}_i$ and withdraw $x = y \cdot \mathtt{askedPrice}$ tokens of type t_1 from B_i's account. However, $\mathtt{SC}_i$ will move the x tokens from B_i's account if S has moved to B_i's account y tokens of type t_2. $\mathtt{SC}_i$ ensures atomic transactions but can only be triggered by the seller.

Any instantiation of Σ-trade protocol can be used in our $\Pi^{\mathtt{trade}}$ protocol. We now show an insantiation of Σ-trade protocol to trade $\check{T}$ for Ξ.

2.1 Selling Tokens for Ethers

For a buyer B_i, denote with $(\mathtt{sk}_i^C, \mathtt{pk}_i^C)$ the signing-verification keys associated with account $C \in \{E, \check{T}\}$, where E represents Ethereum and $\check{T}$, a token on Ethereum. $\varXi$ denotes Ethereum currency. Similarly for seller, $(\mathtt{sk}_S^C, \mathtt{pk}_S^C)$ denote the signing-verification keys associated with account $C \in \{E, \check{T}\}$.

A formal description of the smart-contract and our protocol $\varPi$ is in [35]. Here, we give the intuition. The smart contract $\mathtt{SC}_i$ locks for T_i rounds $\mathtt{z}_i\varXi$ and manages a list of transaction identifiers. Upon receiving an input $(x, y, \mathtt{ID})$ that has been authenticated by both the buyer and the seller, $\mathtt{SC}_i$ moves $x\varXi$ to seller's account if 1) a transaction $\mathtt{trx}$ that moves $y\check{T}$ from the seller's account to the buyer's account has been made and 2) $\mathtt{trx}$ contains the identifier $\mathtt{ID}$ in its payload. In addition, to prevent replay attacks, $\mathtt{SC}_i$ does not allow reusing $\mathtt{ID}$. The same contract $\mathtt{SC}_i$ can be used for multiple trades if $\mathtt{z}_i$ is big enough.

We now describe the protocol. The buyer sends his Ethereum public key to the seller, who replies with the exchange rate, $\mathtt{askedPrice}$ between $\varXi$ and $\check{T}$. If the buyer agrees with $\mathtt{askedPrice}$ and wants to buy $y\check{T}$ tokens, he generates an identifier $\mathtt{ID}$, computes $x = y \cdot \mathtt{askedPrice}$ in $\varXi$, and signs $x||y||\mathtt{ID}$. He sends the signed values (and signature) to the seller. The seller, 1) posts a transaction $\mathtt{trx}$ that pays $y\check{T}$ into the buyer's account, $\mathtt{trx}$ contains $\mathtt{ID}$ in its payload, and 2) signs $x||y||\mathtt{ID}$, and uses the resulting signature, along with signature from the buyer, to invoke $\mathtt{SC}_i$. Note that the seller could post $\mathtt{trx}$ and also sends it to the buyer to indicate that the trade will occur.

3 (Fair) Ordering of Transactions

Our main contribution is the Universally Composable (UC) [7] formalization and realization of the *trade functionality* $\mathcal{F}^{\mathtt{trade}}$. $\mathcal{F}^{\mathtt{trade}}$ formally specifies the only ways in which the market maker can reorder the trades. For simplicity, assume that there are only two assets: $\varXi$ and $\check{T}$. Denote with $\mathtt{price}^{\check{T}\to\varXi}$ (and $\mathtt{price}^{\varXi\to\check{T}}$) the price at which $\mathtt{MM}$ sells $\check{T}$ (or $\varXi$) for $\varXi$ (for $\check{T}$). Assume that trader P_i's trade information is encoded in $\mathtt{trade}_i$. That is, $\mathtt{trade}_i$ describes the type and the amount of assets, the prices, trade direction (sell or buy) and etc. Moreover, assume that all the parties share the procedure $\mathtt{MMalgorithm}$ (the MM algorithm), which on input of a trade outputs the updated prices ($\mathtt{price}^{\check{T}\to\varXi}$ and $\mathtt{price}^{\varXi\to\check{T}}$). At a high level, $\mathcal{F}^{\mathtt{trade}}$ works as follows. Upon receiving a request from a trader P_i, $\mathcal{F}^{\mathtt{trade}}$ sends the prices to P_i, and signals to $\mathtt{MM}$ that P_i wants to trade. If P_i agrees with the prices, he sends trade information, $\mathtt{trade}_i$, to $\mathcal{F}^{\mathtt{trade}}$. Upon receiving $\mathtt{trade}_i$, $\mathcal{F}^{\mathtt{trade}}$ forwards $\mathtt{trade}_i$ to $\mathtt{MM}$ who has two choices: 1) decide not to trade with P_i by sending a command $\mathtt{NO\text{-}TRADE}$ to $\mathcal{F}^{\mathtt{trade}}$, or 2) accept trade with P_i. If $\mathtt{MM}$ does any other action before doing one of these two (e.g., $\mathtt{MM}$ starts trading with a party other than P_i), $\mathcal{F}^{\mathtt{trade}}$ allows that but also sets a special flag $\mathtt{abort}$ to 1. This means that if the traders query $\mathcal{F}^{\mathtt{trade}}$ with the command $\mathtt{getTrades}$ (to get the list of trades accepted by $\mathtt{MM}$), $\mathcal{F}^{\mathtt{trade}}$ would return $\perp$ to denote that $\mathtt{MM}$ has misbehaved. A corrupt $\mathtt{MM}$ can also decide to set

$\Pi^{\mathtt{trade}}$

1) P_i**: creation of a request.** Send $(\mathtt{request}, \mathtt{pk}_i)$ to $\mathtt{MM}$.
2) $\mathtt{MM}$**: waiting for a request.** Upon receiving $(\mathtt{request}, \mathtt{pk}_i)$ from the party P_i do the following.
   – Compute $h' \leftarrow \mathcal{H}(h||\mathtt{pk}_i)$ and set $h \leftarrow h'$ and $\sigma \leftarrow \mathtt{Sign}(\mathtt{sk}_{\mathtt{MM}}, h||\mathtt{pk}_i||\mathtt{e})$.
   – Send $\mathtt{ticket}_1 := (h, \sigma, \mathtt{price}^{\Xi\to\check{T}}, \mathtt{price}^{\check{T}\to\Xi}, \mathtt{pk}_i)$ to P_i and ignore any request that comes from any party $P_j \neq P_i$ for τ rounds.
3) P_i**: finalizing the request.** Upon receiving $\mathtt{ticket}_1 := (h, \sigma, \mathtt{price}^{\Xi\to\check{T}}, \mathtt{price}^{\check{T}\to\Xi}, \mathtt{pk}_i)$ from $\mathtt{MM}$, if the received prices are not satisfactory then send $\mathtt{NO\text{-}TRADE}$. Else create $\mathtt{trade}$ with all the required information with $\mathtt{trade}.p_1 = \mathtt{price}^{\Xi\to\check{T}}$ and $\mathtt{trade}.p_2 = \mathtt{price}^{\check{T}\to\Xi}$ and do the following:
   – If $\mathtt{Ver}(\mathtt{pk}_{\mathtt{MM}}, \sigma, h||\mathtt{pk}_i||\mathtt{e}_i) = 1$ then compute $\sigma_i \leftarrow \mathtt{Sign}(\mathtt{sk}_i, h||\mathtt{trade})$ and send $(\mathtt{trade}, \sigma_i)$ to $\mathtt{MM}$, else ignore the message received from $\mathtt{MM}$
4) $\mathtt{MM}$**: reply to the request of** P_i**.** If $(\mathtt{trade}, \sigma_i)$ is received from P_i within τ rounds such that $\mathtt{Ver}(\mathtt{pk}_i, \sigma_i, h||\mathtt{trade})) = 1$ and $\mathtt{checkTrade}(\mathtt{trade}_i, \mathtt{price}^{\Xi\to\check{T}}, \mathtt{price}^{\check{T}\to\Xi}) = 1$ then do the following.
   – Compute $h' \leftarrow \mathcal{H}(h||\mathtt{trade})$ and set $h \leftarrow h'$.
   – Add $(\mathtt{pk}_i, \mathtt{trade}, \sigma_i)$ to $\mathtt{requests}$.
   – Run $\mathtt{MMalgorithm}(\mathtt{trade}, \mathtt{price}^{\Xi\to\check{T}}, \mathtt{price}^{\check{T}\to\Xi})$ thus obtaining $\mathtt{price}^{\Xi\to\check{T}'}, \mathtt{price}^{\check{T}\to\Xi'}$ and set $\mathtt{price}^{\Xi\to\check{T}} \leftarrow \mathtt{price}^{\Xi\to\check{T}'}, \mathtt{price}^{\check{T}\to\Xi} \leftarrow \mathtt{price}^{\check{T}\to\Xi'}$.

   else do the following
   – Compute $h' \leftarrow \mathcal{H}(h||\mathtt{NO\text{-}TRADE})$ and set $h \leftarrow h'$ and add $(\mathtt{pk}_i, \mathtt{NO\text{-}TRADE}, 0^\lambda)$ to $\mathtt{requests}$.

   Start accepting new requests from any party (i.e., goto step 2).
5) $\mathtt{MM}$**: posting trades to the BB**. If R rounds have passed, post $(h, \sigma, \mathtt{requests}, \sigma^\star, \mathtt{e})$ to the BB, where $\sigma \leftarrow \mathtt{Sign}(\mathtt{sk}_{\mathtt{MM}}, h)$ and $\sigma^\star \leftarrow \mathtt{Sign}(\mathtt{sk}_{\mathtt{MM}}, \mathtt{requests}||\mathtt{e})$. Set $R \leftarrow R + \Delta$, update the epoch number $\mathtt{e} \leftarrow \mathtt{e} + 1$ and reinitialize $\mathtt{requests}$.
6) P_i**: checking honest behavior of the** $\mathtt{MM}$. In each round P_i does the following
   - If no message $(h', \sigma', \mathtt{requests}, \sigma^\star, \mathtt{e}_i)$ has been posted on the BB within the last Δ rounds such that $\mathtt{Ver}(\mathtt{pk}_{\mathtt{MM}}, \sigma', h') = 1$ and $\mathtt{Ver}(\mathtt{pk}_{\mathtt{MM}}, \sigma^\star, \mathtt{requests}||\mathtt{e}_i) = 1$ then output $\perp$, else compute $\mathtt{e}_i \leftarrow \mathtt{e}_i + 1$ and continue as follows.
   - If P_i has not received a new ticket $\mathtt{ticket}_1 := (h, \sigma, \mathtt{price}^{\Xi\to\check{T}}, \mathtt{price}^{\check{T}\to\Xi}, \mathtt{pk}_i)$ during the epoch $\mathtt{e}_i - 1$ then continue, else if $\mathtt{verification}(h_{\mathtt{e}_i-1}, h', h, \mathtt{pk}_i, \mathtt{requests}) = 0$ then send $(h, \sigma, \mathtt{pk}, \mathtt{e}_{i-1})$ to the BB as a proof of cheating of $\mathtt{MM}$ and set $\mathtt{output}_i \leftarrow \perp$.
   - Set $h_{\mathtt{e}_i} \leftarrow h'$.
7) P_i upon receiving $\mathtt{getTrades}$, if $\mathtt{output}_i = \perp$ then return $\perp$ else reinitialize $\mathtt{Trades}$ and do the following.
   - For each message $(h'_j, \sigma'_j, \mathtt{requests}_j, \sigma^\star_j, j)$ with $j \in \{0, \ldots, \mathtt{e}_i - 1\}$ such that $\mathtt{Ver}(\mathtt{pk}_{\mathtt{MM}}, \sigma'_j, h'_j) = 1$ and $\mathtt{Ver}(\mathtt{pk}_{\mathtt{MM}}, \sigma^\star_j, \mathtt{requests}_j||j) = 1$ posted on the BB, if $\mathtt{checkBB}(h_j, h'_j, \mathtt{requests}_j, \mathtt{pk}_{\mathtt{MM}}, j)$=0 then return $\perp$, else for each $(\mathtt{pk}, \mathtt{trade}, \sigma)$ in $\mathtt{requests}_j$ add $\mathtt{trade}$ to $\mathtt{Trades}$.
   - If $\mathtt{verifyPrices}(\mathtt{Trades}) = 1$ then output $(\mathtt{Trades}, \mathtt{e}_i)$ else output $\perp$.

Fig. 1. $\Pi^{\mathtt{trade}}$, the protocol that realizes $\mathcal{F}^{\mathtt{trade}}$.

the output of $\mathcal{F}^{\mathtt{trade}}$ to always be $\perp$. This captures the fact that MM can decide to stop working at his will. Moreover, MM can add any trade of a corrupted party to the list of trades using the command `setAdvTrade`, but this can be done only after MM has concluded any in-progress trades, as specified above.

$\mathcal{F}^{\mathtt{trade}}$ is parametrized by Δ, which denotes the maximum number of rounds per *epoch*. In each *epoch* MM should allow traders to see the entire list of trades. MM can make the list of trades accessible via a special command `setOutput`. If MM does not send this command at least every Δ rounds, $\mathcal{F}^{\mathtt{trade}}$ will return $\perp$ to any honest party who requests trades list.

Note that $\mathcal{F}^{\mathtt{trade}}$ allows the adversarial MM to misbehave (e.g., by completely reordering the trades) but this misbehavior will be notified to the honest parties. Moreover, the MM cannot modify the trades (e.g., change the quantity that a party P_i is willing to sell/buy). Therefore, even if the adversary reorders the trades (at the cost of being detected), all the trades will be consistent with the prices that $\mathcal{F}^{\mathtt{trade}}$ sent to the traders. The market maker still has the power to choose the parties he wants to trade with first, however, this choice has to be made obliviously of the trade information of the honest party. Luckily, we can also argue that for a relevant class of market-making algorithms, this does not constitute an additional useful power. We finally note that $\mathcal{F}^{\mathtt{trade}}$ does not allow any real exchange of assets. However, if the output of $\mathcal{F}^{\mathtt{trade}}$ is posted on a blockchain and if the trades are defined properly according to the language of the blockchain, then the MM can use the trades to trigger events on the blockchain that move the assets according to what is described by $\mathcal{F}^{\mathtt{trade}}$. We can also disincentivize any malicious behavior of the adversary by means of the compensation paradigm over the blockchain. Indeed, given that in our protocol all the honest parties can detect a malicious behavior without using any private state, the same can be done by a smart contract.

To simplify the description of our protocol, we make use of the procedures `checkTrade` and `checkPrices`. `checkTrade` takes as input `trade`, $\mathtt{price}^{\check{T}\rightarrow\Xi}$ and $\mathtt{price}^{\Xi\rightarrow\check{T}}$, and outputs 1 if the description of a trade `trade` is consistent with the prices defined by $(\mathtt{price}^{\check{T}\rightarrow\Xi}, \mathtt{price}^{\Xi\rightarrow\check{T}})$. `checkPrices` takes as input a list of trades and verifies that trade prices are consistent with `MMalgorithm`. These procedures, and $\mathcal{F}^{\mathtt{trade}}$ are formally specified in [35].

3.1 Our Protocol: How to Realize $\mathcal{F}^{\mathtt{trade}}$

Assume all parties have access to a bulletin board BB, all parties know the MM's public key, and the procedure `MMalgorithm` is public. Our protocol realizes $\mathcal{F}^{\mathtt{trade}}$ as follows. MM maintains a hash chain (that starts with a value $h_{\mathtt{start}}$), all parties know $h_{\mathtt{start}}$. Whenever MM receives a request from a trader P_i, he adds to the hash chain the public identity of P_i, signs the new head (say h_i), the public key of P_i and the current prices. We call this set of information a *ticket*. The MM then hands over the ticket to the trader. The trader checks that the signature is valid under the MM's public key, and if so, P_i defines the trade $\mathtt{trade}_i$, signs it thus obtaining σ_i, and sends $(\mathtt{trade}_i, \sigma_i)$ to MM (σ_i guarantees that MM cannot change $\texttt{trade}_i$). MM, upon receiving $\texttt{trade}_i$ and its signature, checks if $\texttt{trade}_i$ is well formed (i.e., the prices used to describe $\texttt{trade}_i$ are consistent with what MM sent in the previous round). If so, MM adds to the hash chain $\texttt{trade}_i$, adds $\texttt{trade}_i$ with σ_i to a list `requests`, run `MMalgorithm` on $\texttt{trade}_i$ and the current prices to get the new prices, and waits to receive next trade request.

In every epoch (at most Δ rounds) MM publishes to the bulletin board[2] the head h of the hash chain and the list `requests`, all authenticated with his signing key. If MM that does not post such authenticated information within Δ rounds then all the traders will understand this as an abort and output $\perp$. Each honest party that has access to the BB now: 1) checks that each trade in `requests` is either `NO-TRADE` or a correctly signed trade; 2) checks that all the prices are consistent with `MMalgorithm` and that the hash chain that starts at $h_{\texttt{start}}$ and finishes at h can be constructed using the trades in `requests`; and, 3) checks if the hash value h_i (received as part of the ticket) is part of the hash chain.

We observe that anyone (even traders who did not trade e.g. third parties) can check if the first and the second conditions hold. If either the first or the second condition does not hold, then all the honest traders output $\perp$. The third condition can be checked only by a trader who received a ticket. If a trader detects that the third condition does not hold, he can post his ticket on the bulletin board. At this point all the other parties who see BB can also determine that MM misbehaved and output $\perp$. Intuitively, our protocol realizes $\mathcal{F}^{\texttt{trade}}$ because once MM sends a ticket to a trader, he also commits to a set of trades. As long as MM cannot generate collisions for the hash function, he cannot include new trades in the hash chain. This protocol, $\Pi^{\texttt{trade}}$, is formally specified in Fig. 1. In the protocol, MM maintains $h \leftarrow 0^\lambda$, an initially empty list `requests` and the integers R, τ and Δ. Δ represents the maximum number of rounds after which MM has to post the trades on the BB, τ represents the timeout (e.g. number of seconds, or rounds) before which a party has to reply to MM (to avoid DoS attack) and R is initialized to Δ. Let also $\texttt{SP}^{\check{T}\rightarrow\Xi}$ and $\texttt{SP}^{\Xi\rightarrow\check{T}}$ be the starting prices. MM also maintains an integer called *epoch index* denoted with e, MM initializes $\texttt{price}^{\Xi\rightarrow\check{T}} \leftarrow \texttt{SP}^{\Xi\rightarrow\check{T}}$ and $\texttt{price}^{\check{T}\rightarrow\Xi} \leftarrow \texttt{SP}^{\check{T}\rightarrow\Xi}$ and $\texttt{e} = 0$. Each party maintains and initially empty list `Trades`, $h_0 \leftarrow 0^\lambda$ and a view of the current epoch index which we denote by $\texttt{e}_i$.

The protocol uses utility procedures to check misbehavior. A formal description of these procedures is presented in the full version [35], a summary follows:

– `verifyPrices` takes as input a list of trades and checks each trade price is computed according to `MMalgorithm`.
– `verification` takes as input the ticket received by a trader, the head of the hash chain and the list of trades posted at the end of an epoch to the BB by MM, and checks whether the ticket appears in the hash chain and its consistency of trades list with hash chain.

[2] Publishing can be done cheaply e.g. by only posting the hash on the blockchain and providing hash-preimages on demand.

- `checkBB` checks BB for valid tickets and runs `verification` for each of them. If `verification` outputs 0, the procedure outputs 0 as well.

In the full version [35] we formally prove the following theorem:

Theorem 1. *Assuming that unforgeable signatures, and collision resistent hash functions exist, $\Pi^{\mathtt{trade}}$ realizes $\mathcal{F}^{\mathtt{trade}}$ in the $(\mathcal{F}_{RO}, BB)$-hybrid world.*

4 Combining $\mathcal{F}^{\mathtt{trade}}$ with Σ-Exchange Protocols

We observe that if, in the realization of $\mathcal{F}^{\mathtt{trade}}$, we replace the BB with a blockchain that supports smart contracts, then a smart contract can act as a party registered to $\mathcal{F}^{\mathtt{trade}}$ that can query $\mathcal{F}^{\mathtt{trade}}$ with the command `getTrades`. We can program this smart contract in such a way that if the output of $\mathcal{F}^{\mathtt{trade}}$ is $\perp$ then the MM is penalized. In our final protocol the traders and MM run a Σ-trade protocol Π, and in parallel, invoke $\mathcal{F}^{\mathtt{trade}}$ with the same information as input i.e. the prices, quantity and the type of the trades used in the execution of Π. Once that the output of $\mathcal{F}^{\mathtt{trade}}$ is generated, we can rely on a smart contract to check that the trades are consistent with the transactions generated by Π. If this is not the case then MM can be penalized. More precisely, to punish a misbehaving $\mathtt{MM}$ we require $\mathtt{MM}$ to create a smart contract $\mathtt{SC}_{\mathtt{penalize}}$ which locks a collateral z. $\mathtt{SC}_{\mathtt{penalize}}$, if queried by any party, inspects the output of $\mathcal{F}^{\mathtt{trade}}$ and if it is $\perp$ then $\mathtt{SC}_{\mathtt{penalize}}$ burns the collateral of $\mathtt{MM}$. Otherwise $\mathtt{SC}_{\mathtt{penalize}}$ checks whether the trades from $\mathcal{F}^{\mathtt{trade}}$ are consistent with the transactions generated by $\mathtt{MM}$ on the Ethereum blockchain with respect to the wallet addresses $(\mathtt{pk}^{\Xi}_{\mathtt{MM}}, \mathtt{pk}^{\check{T}}_{\mathtt{MM}})$. If they are not, $\mathtt{SC}_{\mathtt{penalize}}$ burns the collateral. We note that this contract is expensive to execute (in terms of gas cost). However, if $\mathtt{MM}$ and the traders follow the protocol nobody will ever invoke it. On the other hand, if $\mathtt{MM}$ misbehaves then a trader will detect it (from the output of $\mathcal{F}^{\mathtt{trade}}$) and will invoke $\mathtt{SC}_{\mathtt{penalize}}$. We incentivize the honest invocations of $\mathtt{SC}_{\mathtt{penalize}}$ by transferring a small portion of the locked collateral to the calling party before burning the rest of it.

To finish exposition, we need to introduce yet another smart contract, $\mathtt{SC}_{\mathtt{account}}$. This contract, too, is created by $\mathtt{MM}$. It checks if the transactions that pay the $\mathtt{MM}$'s account exceed a certain value Y. If this is the case, then the contract blocks additional payment towards $\mathtt{MM}$. Hence it bounds the amount of commodities that $\mathtt{MM}$ can trade, We do it to prevent a situation where profit of the $\mathtt{MM}$ exceeds the collateral and thus makes it rational to misbehave (and get penalized). Observe that no malicious (even irrational) $\mathtt{MM}$ can steal money from the traders. The worst that $\mathtt{MM}$ can do is to frontrun the traders (by letting $\mathcal{F}^{\mathtt{trade}}$ output $\perp$) or avoid posting transactions that allow the settling of the trades. Both these types of misbehavior is caught by $\mathtt{SC}_{\mathtt{penalize}}$ and $\mathtt{MM}$ loses collateral. Thus, if we set Y to be smaller than the collateral of $\mathtt{SC}_{\mathtt{penalize}}$, then it is not a viable strategy for a rational $\mathtt{MM}$ to be penalized by means of $\mathtt{SC}_{\mathtt{penalize}}$.

The formal description of our final protocol $\Pi^{\mathtt{full}}$ is in the full version [35]. We describe the case when traders only want to buy $\check{T}$ for Ξ. $\Pi^{\mathtt{full}}$ combines the functionality $\mathcal{F}^{\mathtt{trade}}$ and the Σ-trade protocol. We specify $\mathtt{SC}_{\mathtt{penalize}}$

in [35]. $\texttt{SC}_{\texttt{penalize}}$ acts like a party registered to $\mathcal{F}^{\texttt{trade}}$ who, when queried sends `getTrades` to $\mathcal{F}^{\texttt{trade}}$ and decides whether the MM misbehaved. Let T be the number of rounds for which $\texttt{SC}_{\texttt{penalize}}$ has locked the collateral, we can claim the following:

Theorem 2. *If there is at least one honest party P_i then, within the first T rounds one of the following occurs with overwhelming probability:*

1. *the $\mathcal{F}^{\texttt{trade}}$ outputs $\perp$ and the collateral locked in $\texttt{SC}_{\texttt{penalize}}$ by* `MM` *is burned;*
2. *the $\mathcal{F}^{\texttt{trade}}$ is not $\perp$ but there is not a perfect correspondence between the trades contained in the output of $\mathcal{F}^{\texttt{trade}}$ and the transactions that appear on the blockchain E with respect to* `MM`*'s public keys. Moreover, the collateral locked in $\texttt{SC}_{\texttt{penalize}}$ is burned;*
3. *the $\mathcal{F}^{\texttt{trade}}$ is not $\perp$, there is a perfect correspondence between the trades contained in the output of $\mathcal{F}^{\texttt{trade}}$ and the transactions that appear on the blockchain E with respect to* `MM`*'s public keys and all the collateral remains locked in $\texttt{SC}_{\texttt{penalize}}$ for T rounds.*

For appropriate parameters in the smart contracts, and assuming the market-maker maximizes his amount of $\varXi$, we can argue that the first two cases in Theorem 2 happen with negligible probability. Indeed, let α be the gas cost to run $\texttt{SC}_{\texttt{penalize}}$ with the input `detected`, let `reward` be reward that could be given to a party calling $\texttt{SC}_{\texttt{penalize}}$, let z be the locked collateral in $\texttt{SC}_{\texttt{penalize}}$ and let Y be the maximum amount of $\varXi$ that `MM` can earn at $\texttt{pk}^{\varXi}_{\texttt{MM}}$. If there is at least one honest party P_i, $\texttt{reward} > \alpha$, and $z > Y$ then for every rational market-maker the probability of occurrence of the first two cases of Theorem 2 is negligible.

5 Incentive Compatibility of Market Maker (MM)

We use myopic-greedy market maker from [11] in our construction. Here we provide an overview, see [35] for details and proofs. Let market maker's distribution $p_t(v)$ quantify market maker's information on the true value V after t trades, our market maker has the following properties:

– The market discovers the originally unknown true value of the commodity based on trades with traders who arrive with imperfect information. Empirically, the speed of this convergence is illustrated in [11] and follows the standard $1/t$ convergence for Bayesian updates.
– The market maker uncertainty converges to 0. The market maker recovers the true value in expectation, and also becomes more certain of it. Again, this convergence is standard for Bayesian updates.
– In equilibrium, the market maker spread that produces maximum single step profit monotonically increases with the variance of its distribution, which converges to zero. Hence the bid-ask spread converges to a minimum possible for a profit maximizing market maker. A market maker who knows V can always make more expected profit than a market maker who does not.

The last bullet above is essentially the intuition behind why an optimal market maker has no incentive to manipulate prices. The maximum profit is made when the market maker knows the true value V. Hence the market maker is incentivized to discover the true value V as quickly as possible. The only information available on the V is through the un-manipulated trader signals x_t.

We now present the main theorem (proved in [35]):

Theorem 3 (Incentive compatibility). *A rational profit-seeking market maker has no incentive to manipulate the price given knowledge that some trader wishes to place a trade and the direction (buy/sell) of the trade being known.*

The following lemma states that it is suboptimal for the market maker in our setting to ignore trades without knowledge of other trades.

Lemma 1. *A rational profit-seeking market maker which receives sequential trades, has no incentive to disregard completed trades, even when the direction of the following trade is known.*

6 Evaluation

We implemented $\Pi^{\texttt{trade}}$ to trade Ether and ERC20 tokens on Ethereum (see [35] for implementation details). Table 1 lists the gas costs. Note that the cost of executing one trade is the sum of the costs of *execute* methods of the *SellerContract* and the *BuyerContract*.

6.1 Experiment Setup

To measure throughput, we ran several experiments on a consumer laptop equipped with Core i7-10510U 1.80 GHz CPU and 8 GB of RAM running Ubuntu 20.04. Recall that in our fair trade protocol $\Pi^{\texttt{trade}}$ (see Fig. 1), the buyer P_i first sends its public keys to the seller `MM`. Then the seller responds by sending a ticket and the current prices. Both of these messages can be computed very cheaply. Concretely creating the first message takes less than 50 ms (for each party) in our setup. Then the buyer either responds with `NO-TRADE` or `trade`. This is still cheap and can be done in less than 50 ms. Now the seller must respond to the trade offer. If this offer is `NO-TRADE`, the buyer needs to perform very little work (concretely less than 50 ms). However, if the offer is `trade`, the seller must verify and create signatures, perform balance checks on appropriate assets and create/broadcast a transaction for the trade. These operations are slow (especially the ones that involve communicating with an Ethereum node). Concretely, it takes ≈350 ms to prepare this message. Lastly, we observed that the typical round trip time from buyer → seller → buyer is less than 100 ms.

Our goal was to observe the system's throughput in the following adversarial scenarios. The first is the *Standard DoS attack*. Here, a malicious buyer floods the system with ticket requests and then stops responding, slowing the system down. To this end, we performed the following experiment: n buyers connect to

Table 1. Gas costs of seller and buyer contracts.

Methods		Gas	USD[b]
Contract	Method	47gwei/gas[a]	3,284.20 usd/eth[a]
BuyerContract	claimExpiry	31,619	4.88
	execute	67,984	10.50
SellerContract	execute	33,456	5.16
Deployments			
BuyerContract		1,082,529	167.09
SellerContract		836,341	129.09

[a]Prices taken from https://coinmarketcap.com/ on 2021-09-10.
[b]USD cost is a bad measure of contract complexity. We list it to be consistent with other work.

the seller. The seller responds (with ticket and prices) to them in the order they connect. Upon receiving the ticket (and prices) from the seller, an honest buyer will execute a trade (i.e., the `trade` scenario). On the other hand, a corrupt buyer will stop responding. After a timeout τ, the seller will assume a `NO-TRADE` response, execute the `NO-TRADE` scenario, and move to the next buyer. We ran experiments with $n = 300$ buyers, repeating 5 times and reporting the average measurement. Note that relatively few repetitions of the experiments are not a concern because of low variance of the measured values.

The other attack scenario is a *Worst-case Throughput attack*. The setting remains the same as above with one difference: the malicious buyer now waits until just before the timeout and then responds with a `trade` response. This strategy is more effective at slowing down the system than the standard DoS attack. The reason why it is the case is discussed in the next section.

6.2 Analysis of Results

The results of the experiments for *Standard DoS attack* are summarized in Fig. 2, and for *Worst-case Throughput attack*, in Fig. 3. We observe that in Standard DoS attack (cf. Fig. 2) with no corruptions, throughput is over 200 trades/min. Recall that the gas cost of a trade is $101K$, Assuming *block gas limit* is $12M$ (i.e. the current limit) and block generation delay of 15 s, Upper bound on throughput is 475 trades/min. This upper bound assumes no other application (except ours) competes for block space. Keeping this in mind, achieving over 200 trades/min is an excellent throughput. This number is higher than Uniswap's [23] average throughput/min on its highest daily volume (cf. Sect. 6.3). Recall also that this throughput is achieved on a consumer laptop. A high-end server (typical machine for such use-cases) will yield higher throughput.

Interestingly, at low values of τ, the throughput of the system goes up with the number of corruptions. This is not an anomaly. If a malicious trader does not respond within the timeout τ, the seller assumes a `NO-TRADE` response which takes about one-seventh of the time it takes to execute `trade` response. This means at low values of τ ($\tau <$ execution-timet of `trade`) and some corruptions, some (i.e. the corrupt) trades are cheaper to execute compared to when there are no corruptions (because all honest players trigger the `trade` scenario). This effect disappears as soon as the value of the timeout τ goes near and above the execution-time of the `trade` scenario. While setting a low timeout τ may seem a good idea to defend against malicious parties, it should not be less than the typical round-trip time (100 ms in our trials), otherwise it will cause timeouts for honest players. Note also that a trader needs to setup a smart contract and register with the market maker before commencing trading. Therefore, Sybil attack is not trivial and repeat offenders may be blacklisted.

A better attack strategy would be for a malicious buyer to wait until just before the timeout (for maximum slowdown) and then respond with `trade` response; to trigger the more expensive (in running time) scenario for seller. This strategy removes the above mentioned advantage. Concretely, a malicious seller would wait until he has just enough time left for one round-trip (100 ms in our setup). Thus the amount of time he should wait, `delayBudget`, can be computed as $\texttt{delayBudget} = \tau - \texttt{RoundTripTime}$. The negative effect of such attack is seen in Fig. 3. The throughput has gone down for all values of τ. Importantly though, observe that the x-axis in Fig. 3 starts at $\tau = 200$. This is because at $\tau = 100$, the `delayBudget` of the adversary is 0 i.e., he has to respond immediately and there is no longer a difference between an honest buyer and a malicious buyer.

In conclusion, the choice for value of τ should be the typical round-trip time (with some noise). This prevents throughput-loss even against a determined adversary who wants to pay (via `trade` responses) to slow down the system. Finally, consider that in real life some honest sellers may also respond with `NO-TRADE` e.g., if the prices are not favorable. Hence, the value of 205 trades per minute at $\tau = 100$ should be considered the lower bound.

6.3 Comparison with Uniswap

A summary of FairMM and Uniswap is presented in Table 2. See [35] for our analysis of Uniswap gas costs. At the time of this writing, the throughput values in v2 are higher than v3 e.g. the highest daily volume 3 times more for v2 (251K txns) than v3 (71K txns). So, we compare against v2.

Second, our trade execution time is bounded by the round trip time of the network, about 350 ms. In contrast, Uniswap trades are executed by the miners as part of mining a block. At the time of this writing, etherscan shows high fees transactions (ones that get picked up the soonest) take about 30 s. One can safely say that a trade in Uniswap takes at least 15 s (half of the value on etherscan). This is much larger than the approx. 0.3 s in our system.

First, Uniswap (or any existing market maker, centralized or decentralized) has no defense against front-running attacks without additional trust/hardware assumptions. Our construction resolves this long-standing problem by ensuring that the market maker cannot reorder trades without getting caught.

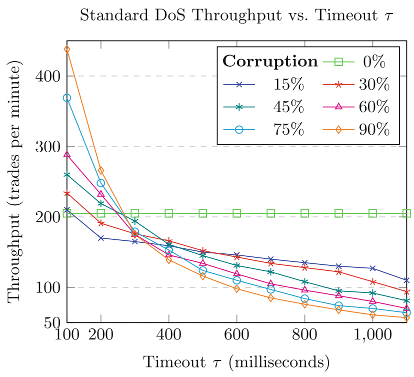

Fig. 2. Standard DoS Attack Throughput (at 0% to 90% corruption). Values average of 5 runs. Note that x-axis starts at ms, this is the typical RTT, and any $\tau < 100$ may cause timeouts for honest players.

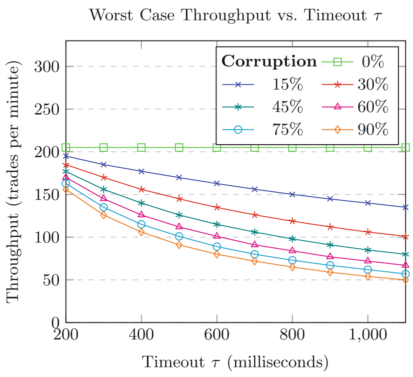

Fig. 3. Worst-case Throughput (at 0% to 90% corruptions). Note: x-axis starts at 200 ms because malicious player needs a budget of at least RTT (100 ms here) to respond without risking timeout.

Third, in Uniswap and similar systems, miners are free to reorder trades. This gives them a profit opportunity e.g. including favorable trades first. On the contrary, in our system the trade order is fixed before the corresponding transactions are broadcast to the blockchain, nullifying miners' influence. Moreover—

Table 2. Comparison Summary

Feature	FairMM	Uniswap
Front running resilience	**Yes**	No
Gas price auctions	**No**	Yes
Miner influence	**No**	Yes
Trade execution (seconds)	≈**0.30**	≥15
Average trade cost l(K)	≈**101**	≈141[a]
Max trade cost (K)	≈**101**	≈1,316[b]
Max throughput[c]	≈**475**	≈340

[a]Based on average cost of 1M trade transactions (block 12,162,664 to 12,231,464). Trades are calls to swap methods of V2Router02.
[b]Txn: https://etherscan.io/tx/0xa87b492f2945d2a99ca1f8e2d9530599c040f00c3257f989f9c2822e20b2ed5e). There may be more expensive transactions outside our dataset.
[c]in trades/minute. Theoretical upper bound on throughput based on average trade cost, assuming 12M block gas limit on Ethereum network.

because of the above mentioned miners' influence—traders on Uniswap have an incentive to pay high *gas price* to get their trade included sooner. In fact, since the traders can see other traders' activity, they can actively compete with one another. Such trading behavior induces the, so called, *gas price auction*s attack. Gas price auctions needlessly raise transaction cost for everyone (not just the traders). Transactions in our system are merely moving the funds and may be mined in any order. There is no incentive to pay higher than usual gas price.

Fourth, gas cost in Uniswap is variable. We observed an average gas cost of $141K$. It can be much higher depending on the trade e.g. over $1,316K$ for txn 0xa87b492f2945d2a99ca1f8e2d9530599c040f00c3257f989f9c2822e20b2ed5e. Recall that Uniswap is specifically designed and optimized for Ethereum. On the other hand, our system design is general and lacks aggressive optimizations. Yet, the gas cost of our system is constant at $101K$. Notwithstanding, even if the gas cost of Uniswap transactions were much lower than ours, Uniswap's transactions would still be more costly in Ethers because of the gas price auctions mentioned above.

Finally, based on the average trade gas cost and assuming a block gas limit of $12M$, the maximum throughput of Uniswap is $\approx$ 340 trades per minute. This is less than our upper bound of 475. Concretely, highest daily volume[3] on Uniswap has been $\approx 251K$ transactions. On average, this means about 174 trades per minute. Importantly, this throughput is achieved in a scenario where all trade data is locally available. Our construction on the other hand, communicates with the traders in real time. The fact that this communication happens sequentially—on first come first served basis—negatively affects our throughput. Despite this, we achieve at least 200 trades per minute (higher than the highest volume Uniswap). We stress that this throughput was achieved on a mid-range consumer machine. A computationally powerful server will increase throughput further. Therefore, we do not see it as a major problem in practice.

References

1. Glosten, L.R., Milgrom, P.R.: Bid, ask and transaction prices in a specialist market with heterogeneously informed traders. J. Financ. Econ. **14**(1), 71–100 (1985)
2. Yao, A.C.-C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS. IEEE Computer Society Press, pp. 162–167, October 1986
3. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC, pp. 218–229. ACM Press, May 1987
4. Glosten, L.R.: Insider trading, liquidity, and the role of the monopolist specialist. J. Bus. **62**(2), 211–235 (1989)
5. Asokan, N., Shoup, V., Waidner, M.: Optimistic fair exchange of digital signatures. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 591–606. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054156
6. Cachin, C., Camenisch, J.: Optimistic fair secure computation. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 93–111. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44598-6_6

[3] https://etherscan.io/address/0x7a250d5630b4cf539739df2c5dacb4c659f2488d#analytics.

7. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press, October 2001
8. Wolfers, J., Zitzewitz, E.: Prediction markets. J. Econ. Perspect. **18**(2), 107–126 (2004)
9. Das, S.: A learning market-maker in the Glosten-Milgrom model. Quant. Fin. **5**(2), 169–180 (2005)
10. Pennock, D., Sami, R.: Computational aspects of prediction markets. In: Algorithmic Game Theory. Cambridge University Press (2007)
11. Das, S., Magdon-Ismail, M.: Adapting to a market shock: optimal sequential market-making. In: Proceedings of the Advances in Neural Information Processing Systems (NIPS), pp. 361–368 (2008)
12. Küpçü, A., Lysyanskaya, A.: Usable optimistic fair exchange. In: Pieprzyk, J. (ed.) CT-RSA 2010. LNCS, vol. 5985, pp. 252–267. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11925-5_18
13. Bentov, I., Kumaresan, R.: How to use bitcoin to design fair protocols. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 421–439. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_24
14. Jutla, C.S.: Upending stock market structure using secure multi-party computation. Cryptology ePrint Archive, Report 2015/550 (2015). https://eprint.iacr.org/2015/550
15. Banasik, W., Dziembowski, S., Malinowski, D.: Efficient zero-knowledge contingent payments in cryptocurrencies without scripts. In: Askoxylakis, I., Ioannidis, S., Katsikas, S., Meadows, C. (eds.) ESORICS 2016, Part II. LNCS, vol. 9879, pp. 261–280. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45741-3_14
16. Kiayias, A., Zhou, H.-S., Zikas, V.: Fair and robust multi-party computation using a global transaction ledger. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 705–734. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_25
17. Campanelli, M., et al.: Zero-knowledge contingent payments revisited: attacks and payments for services. In: Thuraisingham, B.M., et al. (eds.) ACM CCS 2017. ACM Press, pp. 229–243 (2017)
18. Warren, W., Bandeali, A.: Ox: an open protocol for decentralized exchange on the Ethereum blockchain (2017)
19. AirSwap: AirSwap (2018)
20. Ether Delta: EtherDelta (2018)
21. IDEX: IDEX (2018)
22. Kyber: Kyber (2018)
23. Uniswap: Uniswap Exchange Protocol (2018)
24. Bitcoin Wiki: Zero Knowledge Contingent Payment (2018)
25. Bentov, I., et al.: Tesseract: real-time cryptocurrency exchange using trusted hardware. In: Cavallaro, L., et al. (eds.) ACM CCS 2019, pp. 1521–1538. ACM Press, November 2019
26. Fuchsbauer, G.: WI is not enough: zero-knowledge contingent (service) payments revisited. Cryptology ePrint Archive, Report 2019/964 (2019). https://eprint.iacr.org/2019/964
27. Khalil, R., Gervais, A., Felley, G.: TEX - a securely scalable trustless exchange. Cryptology ePrint Archive, Report 2019/265 (2019). https://eprint.iacr.org/2019/265
28. Curve: Curve (2020)

29. Daian, P., et al.: Flash Boys 2.0: frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In: 2020 IEEE Symposium on Security and Privacy, pp. 910–927. IEEE Computer Society Press, May 2020
30. Sobol, A.: Frontrunning on automated decentralized exchange in proof of stake environment. Cryptology ePrint Archive, Report 2020/1206 (2020). https://eprint.iacr.org/2020/1206
31. Almashaqbeh, G., et al.: Gage MPC: bypassing residual function leakage for non-interactive MPC. Cryptology ePrint Archive, Report 2021/256 (2021). https://eprint.iacr.org/2021/256
32. Bartoletti, M., Chiang, J.H., Lluch-Lafuente, A.: Maximizing extractable value from automated market makers. In: CoRR abs/2106.01870 (2021)
33. Baum, C., David, B., Frederiksen, T.: P2DEX: privacy-preserving decentralized cryptocurrency exchange. Cryptology ePrint Archive, Report 2021/283 (2021). https://eprint.iacr.org/2021/283
34. Breidenbach, L., et al.: Chainlink 2.0: next steps in the evolution of decentralized oracle networks (2021)
35. Ciampi, M., et al.: FairMM: a fast and frontrunning-resistant crypto market-maker. Cryptology ePrint Archive, Report 2021/609 (2021). https://ia.cr/2021/609
36. Flashbots: Flashbots (2021)
37. Gnosis: Introducing Gnosis Protocol V2 and Balancer-Gnosis-Protocol (2021)
38. Stathakopoulou, C., et al.: Adding fairness to order: preventing front-running attacks in BFT protocols using TEEs. In: 40th International Symposium on Reliable Distributed Systems, SRDS 2021, Chicago, IL, USA, 20–23 September 2021, pp. 34–45. IEEE (2021)
39. Zhou, L., Qin, K., Gervais, A.: A2MM: mitigating frontrunning, transaction reordering and consensus instability in decentralized exchanges. In: CoRR abs/2106.07371 (2021)
40. Zhou, L., et al.: High-frequency trading on decentralized on-chain exchanges. In: 2021 IEEE Symposium on Security and Privacy (SP), pp. 428–445 (2021)
41. Bancor: Bancor Network