FairTraDEX: A Decentralised Exchange Preventing Value Extraction

Conor McMenamin[1,2], Vanesa Daza[1,3], Matthias Fitzi[4], and Padraic O'Donoghue

[1] Department of Information and Communication Technologies, Universitat Pompeu Fabra, Barcelona, Spain
[2] NOKIA Bell Labs, Nozay, France
[3] CYBERCAT - Center for Cybersecurity Research of Catalonia
[4] IOHK

Abstract. We present FairTraDEX, a decentralized exchange (DEX) protocol based on frequent batch auctions (FBAs), which provides formal game-theoretic guarantees against extractable value. FBAs when run by a trusted third-party provide unique game-theoretic optimal strategies which ensure players are shown prices equal to the liquidity provider's fair price, excluding explicit, pre-determined fees. FairTraDEX replicates the key features of an FBA that provide these game-theoretic guarantees using a combination of set-membership in zero-knowledge protocols and an escrow-enforced commit-reveal protocol. We extend the results of FBAs to handle monopolistic and/or malicious liquidity providers. We provide real-world examples that demonstrate that the costs of executing orders in existing academic and industry-standard protocols become prohibitive as order size increases due to basic value extraction techniques, popularized as maximal extractable value. We further demonstrate that FairTraDEX protects against these execution costs, guaranteeing a fixed fee model independent of order size, the first guarantee of it's kind for a DEX protocol. We also provide detailed Solidity and pseudo-code implementations of FairTraDEX, making FairTraDEX a novel and practical contribution.

Keywords: Extractable Value · Decentralized Exchange · Incentives · Blockchain

1 Introduction

One of the most prominent and widely-used classes of protocols being run on smart-contract enabled blockchains is that of decentralised-exchange (DEX) protocols. DEX protocols allow a specific set of players, whom we call *clients*, to

email — (primary) *conor.mcmenamin@upf.edu*
vanesa.daza@upf.edu — *matthias.fitzi@iohk.io* — *padraicodonoghue@gmail.com*

This Technical Report 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

exchange one token for another in the presence of *market-makers* (MMs), who provide liquidity to clients, usually in exchange for a fee. Interacting with a blockchain-based DEX requires a client or MM to first interact with the players who add transactions to the blockchain, known as miners or block producers. These interactions typically reveal a player's intention to trade to the block producer before the transaction is confirmed on the blockchain, and in doing so, present the block producer with what has become known as a miner-extractable value (MEV) opportunity. MEV, first coined in [16], refers to any expected profits the miner of a block can extract from other players interacting with the blockchain. This extraction is performed by manipulating the ordering of, injecting, and/or censoring transactions in prospective blocks. This has been generalised to expected extractable value (EEV) [26] (defined in Appendix B), as non-miner players can also perform many of these attacks.

A significant advancement in DEX protocols was the advent of *automated market makers* (AMMs), with Uniswap [40] being the most prominent of which. Projects like Flashbots [18] (a direct spin-off to [16]) have identified that AMMs are the main source of recorded EEV ($> 98\%$, as seen in the chart labelled "Extracted MEV Split by Protocol" in [18], of the \$665M in EEV identified by Flashbots since August, 2020). Furthermore, Flashbots only observes basic forms of EEV, meaning in reality (and as stated by the Flashbots team [19]), this amount of EEV is a lower bound for the total amount of value being extracted from clients and MMs alike through participation in DEX protocols. To this point, a peer-reviewed analysis in [37] identified \$540.54M in extracted value up to August 2021, indicating the current number provided by Flashbots is significantly lower than the actual amount of extracted value. In [28], it has been further highlighted that in Uniswap V3, liquidity providers are losing more to EEV attacks (impermanent loss in this case) than they are collecting in fees. It is clear that the long-term viability of existing DEX protocols is not plausible.

Although many attempts have been made academically to address this significant source of EEV [13,4,14,27,20,2], no satisfactory solution has been found. The protocols presented in these works remain vulnerable to basic EEV attacks in the case where all transactions are eventually added to the blockchain (censorship-resistant). The overriding trend in these papers is a purely cryptographic approach to an economic/game-theoretic problem, resulting in relatively straightforward game-theoretic exploits. When describing these protocols in Section 2, we also describe some of the value extraction attacks to which the respective protocols are vulnerable. Therefore, there is a clear gap, both in literature and in practice, to provide a DEX protocol which definitively eliminates all sources of EEV. In this paper, we provide such a protocol. We outline FairTraDEX, a DEX protocol based on an existing auction process called *frequent batch auctions* (FBAs) [9] and zero-knowledge (ZK) tools for set-membership. FBAs have been proven to provide a strict Nash Equilibrium in which clients trade at the market fair price for a particular token swap, although are intended to be run by a trusted third party. FairTraDEX is, to the best of our knowledge, the first DEX protocol outlining specific practical conditions under which EEV is prevented in a censorship-resistant blockchain by replicating the guarantees of an FBA without the need for a trusted third party. We formally prove this security against EEV by utilising results from ZK literature and game-theory.

1.1 Our Contribution

We first introduce a width-sensitive frequent batch auction (WSFBA), an idealised commit-reveal exchange protocol between clients and MMs, based on FBAs [9]. WSFBAs are an important improvement on basic FBAs with respect to decentralised systems. WSFBAs ensure clients submit market orders in the presence of monopolistic MMs, compared to a standard FBA in which clients are required to submit limit orders at the fair price plus some trade fee. The requirement for clients to submit limit orders leads to worse order execution as trade probability is decreased, while also placing a significant burden on clients to choose the fair price. This burden is removed in an WSFBA, providing an "obvious optimal" for clients, as coined in [38]. Furthermore, in the case of competing MMs, a WSFBA provides equivalent equilibria to [9]. Importantly, this equilibrium involves all client orders being traded in the auction in which the orders are submitted, removing any strategies which involve unfilled orders needing to being revealed and resubmitted in proceeding auctions.

We then describe FairTraDEX, a blockchain-based implementation of a WSFBA. In FairTraDEX, order commitments are recorded on-chain (to enforce the corresponding escrow punishment). We utilise ZK set-membership proofs to allow clients to commit to their orders anonymously. As such, in FairTraDEX, every client must initially register to the protocol, depositing an escrow. Then, whenever a client wants to commit to an order, the client only has to prove that they are a member of the set of players who registered in the protocol. Given enough registrations, the probability a client's ZK set-membership proof and committed order relates to the actual order contents approaches 0 (we formalise this notion in Section 5) such that no other player in the system can see the committed order and use it to infer anything about what the order is. To definitively hide a client's order-information, orders are committed, including the ZK-proof, by using a relayer, a third-party who receives a fee for including relayed transactions in the blockchain (see Appendix B.2 for further details).

We provide an extensive Ethereum virtual-machine (EVM) compatible proof-of-concept for FairTraDEX [31] including a comparison of protocol running costs with previous solutions in Section 6, which remain constant with respect to order size, price and direction. When compared to potentially percentage-point slippages and EEV-attack costs required to trade on current DEXs, also highlighted in Section 6, FairTraDEX's formal guarantees of protocol-level EEV prevention and up-front, fixed and explicit costs set a new standard for DEX protocols.

2 Related Work

The main works aimed at protecting DEX users from EEV either focus on preventing front-running of orders [13,4], the fair ordering of transactions based on their delivery time [27], or on hiding client trade information until the trade has been committed to the blockchain [20,2,14]. Of these works, the closest to our proposal are [13,4,20,14]. All of these works critically depend on honesty from MMs, auction operators and/or the block proposers. In Appendix A, we provide high-level descriptions of the DEX protocols while in this section, we briefly outline how, when all players in the respective DEX protocols are rational, EEV opportunities exist.

In [13], the MM is always allowed to see orders from clients and can choose to abort them. It is argued that MMs are happy to trade against all orders, including informed orders. Furthermore, it is assumed that clients are independent, with random information. This is not true in real-world trading environments, and as such, Theorem 3 may not hold in practice.

In P2DEX [4], clients must publicly deposit the tokens with which to trade in the same time frame as the order matching takes place, exposing clients to standard identity- and directional-based EEV exploits. Separating token deposit and identity revelation from a client's commitment to a specific auction are important advancements used in FairTraDEX to protect against EEV. Furthermore, at least one of the servers in charge of fairly executing orders is required to be honest-by-default. If the servers, a minority subset of players in the P2DEX protocol, are rational and monopolised/colluding, the servers can front-run orders. In FairTraDEX, such value-extraction is prevented by keeping all order information hidden until every order in a particular settlement round has been committed. Furthermore, the set of servers act as a point-of-failure as server participation is required to finalise order-settlement. No subset of players in FairTraDEX can prevent the matching of correctly-revealed orders, while in P2DEX, any majority of servers can prevent order-matching.

Similar commit-reveal protocols to FairTraDEX for blockchain-based token-exchange are proposed in [20,14]. The protocol in [20] is exposed to several game-theoretic exploits which contradict its protection against front-running. These include the necessity to reveal order direction a-priori, and the non-trivial handling of the linkability between commitments and account-balances. The protocol also depends on an operator who does not participate in token-exchange, gains exclusive access to order information, and is depended on for protocol completion. In [14], clients commit their own orders to the blockchain, revealing their identities, and corresponding token balances/execution patterns which can be used by a basic professional MM to skew prices and extract value from the client. Both of [20,14] aim to protect unmatched orders from being revealed, but to do so, both protocols depend on a third party selected before the auction begins to execute order-matching (no one else in the ecosystem can finalise the auction, a single point of failure). Both protocols assume that the operator does not reveal unexecuted order information.

3 Preliminaries

This section introduces the terminology and definitions necessary to understand the main results of the paper. By *negl*() we denote any function $f : \mathbb{N} \rightarrow \mathbb{R}$ that decreases faster than any (positive) polynomial p. More formally, $\forall\ p\ \exists\ \lambda_0 \in \mathbb{N} : \forall \lambda > \lambda_0 : f(\lambda) < \frac{1}{p(\lambda)}$. For protocol correctness, we must assume that some of the involved players may be malicious trying to force the protocol into incorrect execution, and without any direct benefit for themselves. However, for the game-theoretic part of the analysis, we assume that all players are rational. Accordingly, the analysis of our protocol is based on two security parameters, a cryptographic security parameter κ used to bound the probability that the protocol execution is incorrect, and a game-theoretic extractable-value parameter ψ used to bound the extractable value of any player. The following are the crucial terms and definitions needed for the remainder of the paper, with supplementary terminology and definitions contained in Appendix B.

- *Notional Value*: The value of a set of tokens expressed in some common reference token. In this paper, we use the symbol ₿ as the reference token in which we measure notional, and with which we reason about utility.
- *Market-Implied Fair Price* (MIFP, denoted y): As in [9], the MIFP of a token/token swap is a publicly observable signal which is perfectly informative of the fundamental fair price of the underlying token/token swap. Moreover, a random order of fixed notional generated by a player in the system is equally likely to buy or sell tokens at the MIFP, distributed symmetrically around the MIFP. Unless otherwise stated, observing the MIFP has a prohibitive cost for players in our system.
- *Market*: A market in a DEX between two tokens A_{tkn} and B_{tkn} consists of two limit orders, a *bid* and *offer*. When the market is quoted from token A_{tkn} to B_{tkn}, the offer price indicates the quantity of token A_{tkn} a player must sell for 1 token B_{tkn}, while the bid price indicates the quantity of token A_{tkn} a player receives for 1 token B_{tkn}. In this paper, we represent such a market as *bid* @ *offer*, with $0 < bid \leq offer$.
- *Reference Price* (y_{ref}): For a market *bid* @ *offer*, the reference price y_{ref} is the price such that $\frac{bid}{y_{ref}} = \frac{y_{ref}}{offer}$, i.e., y_{ref} is the geometric mean of *bid* and *offer*.
- *(Market) Width* (w): For a market *bid* @ *offer*, the width is calculated as $w = \frac{offer}{bid}$ (as such $w \geq 1$).
- *Multiplicative Market-Impact Coefficient (δ)*: If the pre-trade MIFP for particular swap is y, the expected post-trade MIFP given a buy order is $\delta\ y$ for some $\delta \geq 1$, while the expected post-trade MIFP given a sell order is $\frac{y}{\delta}$. Unless otherwise stated, a swap from A_{tkn} to B_{tkn} with multiplicative market-impact coefficient δ corresponds to buy orders of B_{tkn} having a multiplicative market-impact coefficient on y_B of $\sqrt{\delta}$ and $\frac{1}{\sqrt{\delta}}$ on y_A. Given our definition of the MIFP, this impact function implies an upward drift in y if $\delta > 1$. However, our use of δ is intended to highlight that impact must

be considered, with the exact choice of δ for a particular token pair being a complex process and beyond the scope of this paper.

- *Client*: Any player in a DEX protocol who, for an MIFP y, there exists some *minimum client utility* $f_{mcf} > 1$ such that client buyers (sellers) have positive expected utility to trade for or below $\sqrt{f_{mcf}}\, y$ (at or above $\frac{y}{\sqrt{f_{mcf}}}$).
- *Market Maker* (MM): A player in a DEX protocol with large supplies of all tokens, who has positive expected utility trading with clients on markets of any width $w > 1$ with reference price equal to the MIFP. MMs can observe the MIFP.
- *Strict Nash Equilibrium* (SNE) [35]: Consider a set of non-cooperative players P_1 ,..., P_n, with strategies (series' of actions) str_1 ,..., str_n describing the actions which each player takes throughout a particular protocol. These strategies form a strict Nash Equilibrium if any individual player deviation from these strategies strictly reduces that player's utility.

In creating a DEX protocol, an idealised goal would be to ensure that there exists an SNE in which clients trade at the MIFP in expectancy. However, this is unrealistic as MMs are a key component in liquidity provision. Therefore a more realistic, yet still desirable, goal would be to ensure that there exists an SNE where clients can trade at the MIFP in expectancy in exchange for some pre-determined fee, payable to the MMs, which is bounded by the clients' utility gain from the swap. In existing AMMs and DEX protocols, this realistic goal remains unachieved, as explained in Section 2. FairTraDEX however, achieves this goal.

Note that MMs differ from liquidity pools in AMMs. The decision logic of AMM liquidity pools is public and deterministic, and any adjustments to liquidity pools must be queued publicly in the mempool, exposing it to EEV attacks. MMs, however, make private trading decisions and communicate them on-chain. One possible action is to add liquidity to an AMM, or in the case of FairTraDEX, add a market to an auction. Following the analysis of [28] and the losses being incurred by liquidity providers in AMMs, players currently providing liquidity in AMMs, although acting honestly, do not fit our rational player model. In FairTraDEX, by ensuring following the protocol forms an SNE, honesty and rationality are equivalent. If players deviate from the protocol in FairTraDEX, this strictly decreases their expected utility, which is further discussed in Appendix G.6.

3.1 Zero-Knowledge Primitives

The aim of this section is to outline the *non-interactive zero-knowledge* (NIZK) tools for set membership as used in this paper, such as those stemming from papers like [33,5,24,7,25]. We generalise these formal works, allowing for the adoption of any secure NIZK set-membership protocol into FairTraDEX, as we only require a common functionality that is shared by all of them. Further elaboration on these protocols is deferred to Appendix C.

The zero-knowledge proofs used in FairTraDEX allow, for a given set of commitments *Com* to user-generated secrets, that any user knowing the secret corresponding to a commitment $com \in Com$ can prove the knowledge of a secret corresponding to a commitment in the set, without revealing which secret, or commitment. Moreover, we require that more than one proof relating to the same commitment is identifiable by a verifier.

To participate in FairTraDEX, clients privately generate two bit strings, the *serial number* S and *randomness* r, with $S,\ r \in \{0,1\}^{\Theta(\kappa)}$. To describe FairTraDEX we define a commitment scheme h, a set membership proof scheme *SetMembership*, an NIZK proof of knowledge scheme *NIZKPoK* and a NIZK signature of knowledge scheme (*NIZKSoK*). We do not specify which instantiation of these schemes to use, as the exact choice will depend on several factors, such as efficiency, resource limitations and/or the strength of the assumptions used.

– $h(m)$: A deterministic, collision-resistant function taking as input a string $m \in \{0,1\}^*$, and outputting a string $com \in \{0,1\}^{\Theta(\kappa)}$.
– $SetMembership(com, Com)$: Compresses a set of commitments *Com* and generates a membership proof π that *com* is in *Com* if $com \in Com$.
– $NIZKPoK(r, S, Com)$: For a set of commitments *Com*, returns a string S and NIZK proof of knowledge if the person running $NIZKPoK()$ knows an r producing a proof when running $SetMembership(h(S||r), Com)$. In FairTraDEX, this revelation identifies to a verifier when a proof has previously been provided for a particular, albeit unknown, commitment as the prover must reproduce S. This is used in FairTraDEX, in conjunction with an escrow, to enforce the correct participation of both, clients and MMs.
– $NIZKSoK(m)$: Returns a signature of knowledge that the person who chose m can also produce NIZKPoK.

4 Width-Sensitive Frequent Batch Auctions

In this section we outline the properties of an idealised variation of an FBA which we define as a width-sensitive FBA. Width-sensitive FBAs maintain the desirable properties of FBAs with respect to optimal strategies for MMs and clients [9], while also adding important protections for clients in a decentralised setting where monopolistic MMs may exist. The important assumption with regard to the guarantees of an FBA is the presence of at least two non-cooperative MMs. In a decentralised setting, this can be seen as insufficient. One of the most desirable properties of FBAs in the presence of 2 non-cooperative MMs is the fact that clients submit market orders. We envisage clients as relatively uninformed players for whom choosing the correct/fair price to trade has an implicit cost. Market orders remove this burden, providing clients with an "obvious optimal" as advocated in [38]. To reach a similar equilibrium in the presence of a monopolistic MM, we must amend the basic FBA protocol. In this section, we define a *width-sensitive* FBA (WSFBA) to handle monopolistic MMs, while retaining the desirable properties of an FBA in the presence of two or more non-cooperative MMs.

In the presence of a single rational MM, we need to utilise the value gained by clients for exchanging token. That is, recall from Section 3, clients in our protocol observe a positive utility of at least the minimum client fee f_{mcf} for exchanging tokens. In a WSFBA, this fee is translated to a market width, and input with clients orders as a maximum market width on which clients are willing to trade. This allows us to prove submitting market orders remains a SNE. Conversely in an FBA, if MMs cooperate/are replaced by a monopolistic MM, submitting market orders is a strictly dominated strategy for clients, with clients now required to submit a limit price. WSFBAs avoid this degradation of user experience, and the corresponding reduced probability of execution and quality of liquidity this has on FBAs.

We let D represent the net trade imbalance of clients in a particular instance of a WSFBA in terms of Ᵽ. A positive D indicates a client buy imbalance (more client buyers than sellers of the swap), while a negative D indicates a client sell imbalance. We require a finite bound on the absolute imbalance, which we denote $Q_{not} < \infty$, for the existence of optimal MM strategies. As in [9], we assume that $|D| \leq Q_{not}$, and in-keeping with the notion of an MIFP, D is symmetric around 0 at the MIFP. This Q_{not} is used as the lower-bound on the notional of a MM's bid and offer in WSFBA. Importantly, this ensures client orders submitted to the auction are executed (used in the proof of Theorem 1). We now define a WSFBA.

Definition 1. *A width-sensitive frequent batch auction (WSFBA) involves MMs submitting markets to the TTP with total notional on the bid and offer of at least Q_{not}. Clients and MMs privately submit limit and market orders to the TTP including a requested maximum width from the tightest MM, above which the order is not executed. Orders are collected until a specified deadline. After this deadline, client orders with requested width greater than or equal to the tightest MM width, along with a randomly-selected market from the tightest provided markets, are settled at a single clearing price which maximises the total notional traded, and then minimises the net trade imbalance.*[5] *If there is more supply at the clearing price than demand, sell orders at the highest price at or below the clearing price are pro-rated based on size such that supply equals demand at the clearing price. Similarly, if there is more demand than supply at the clearing price, buy orders at the lowest price at or above the clearing price are pro-rated based on size such that demand equals supply at the clearing price. Any limit buy orders below/sell orders above the clearing price are not executed.*

The key differences between a conventional FBA and a WSFBA are the specification of MM widths by clients, the minimum MM notional requirement on the bid and offer, and the requirement for the clearing price to minimise the imbalance over all prices which maximise the notional traded. Minimising imbalance is a small optimisation which produces a reasonable and precise clearing price when MMs do not show width 1 markets as in an FBA. A precise algorithm for verifying a given clearing price satisfies these proprieties is included both in

[5] As Q_{not} is greater than the absolute client order imbalance, the clearing price must lie between the MM bid and offer

Algorithm 4 and [31], and described in Appendix F. The other amendments are intended to protect clients against monopolistic MMs, and are discussed in the proceeding section.

4.1 Properties of Width-Sensitive Frequent Batch Auctions

In Theorem 1, we identify an SNE for WSFBAs, and show that it is equivalent to the SNE of an FBA. The case of a single monopolistic MM is more complex. First, we observe that an MM in a WSFBA always shows a market with reference price equal to the MIFP. In the proceeding lemmas and theorems, proofs omitted from the main body of the paper are included in Appendix D.

Lemma 1. *For an MM in a WSFBA between A_{tkn} and B_{tkn} with MIFP equal to $y_{A\rightarrow B} = \frac{y_B}{y_A}$ and a client order of notional $X Ƀ > 0$, she strictly maximizes her expected utility by showing a market with reference price $y_{ref} = y_{A\rightarrow B}$ for any fixed width $w \geq 1$.*

This result is independent of the choice of width and market-impact coefficient. However, it assumes that the MM trades with the client on either the bid or the offer. With respect to a WSFBA without notional restrictions and a monopolistic MM, if clients submit market orders, there are fringe cases (large imbalances) which incentivize MMs to show markets far from the MIFP. Removing these restrictions from a WSFBA makes for interesting future work.

Recall clients have a strictly positive utility to exchange tokens described by the minimum client fee f_{mcf}, which is equivalent to being strongly incentivized to trade on a market with reference price y_{ref} and width $w \leq f_{mcf}$. With this in mind, we can now apply the main result of [9] to a WSFBA.

Theorem 1. *For a WSFBA, the strict Nash equilibria strategies given the number of non-cooperative MMs submitting markets being N are:*

- $N = 1$*: Clients submit market orders of requested width f_{mcf} and the MM shows a market of width at most f_{mcf} with reference price equal to the MIFP.*
- $N \geq 2$*: Clients submit market orders of requested width greater than 1 and MMs show a market of width 1 with reference price equal to the MIFP.*

Theorem 1 identifies that clients always submit market orders, and in settings where it is unclear whether there is a single monopolistic MM, or many non-cooperative MMs, it can be seen that clients always submit market orders with requested width f_{mcf}.

5 FairTraDEX

In Section 4 we constructed a WSFBA using a TTP to enforce correct player balances, order sizes, revelation of orders, correct calculation of the clearing price and the settlement of orders. In a decentralised setting with rational players, such a TTP does not exist. However, we do have access to censorship-resistant public bulletin boards in the form of blockchain-protocols. As discussed in the Section 1, these bulletin boards have many caveats such as the ordering of transactions based on transaction send time not being preserved (transaction re-ordering attacks). However, if we are able to bound the delay of updates being added to such a bulletin board (transactions being confirmed on the blockchain), we can implement a WSFBA in such a setting.

In this section we construct the FairTraDEX protocol as a sequence of algorithms. We then provide a series of results regarding the incentive compatibility of these algorithms with the goal of proving FairTraDEX instantiates a WSFBA, and that following the protocol is an SNE.

5.1 System Model

1. All players $P_1, ..., P_n$ are members of a blockchain-based distributed ledger, and a corresponding PKI.
2. The ledger is represented by a linear blockchain with its state progressing by having new blocks sequentially appended. For simplicity, we assume instant finality of blocks meaning that such an appended (valid) block cannot be replaced at any later point in time.
3. A transaction submitted by a player for addition to the blockchain (either directly or relayed) while observing blockchain height H, is included (and thus finalised) in a block of height at most $H + T$, for some known $T > 0$, given that the transaction remains valid for sufficiently many intermediate ledger states.
4. The public NIZK parameters are set-up in a trusted manner.

We do not make any assumptions regarding transaction ordering in blocks. Specifically, the order in which transactions are executed is at the discretion of the block proposer.

If block producers are participating as MMs/clients, we need to adjust T. Let $0 < \alpha < 1$ bound the fraction of blocks produced over chains of length greater than T by a MM responding to/the set of clients requesting trades in a particular instance of a FBA (we need to consider all clients in a request phase, as they may all have the same direction, and as such, some positive expectancy to preventing a MM revelation). We need to increase T by a factor of $\frac{1}{1-\alpha}$ (similar to the methodology behind the Chain Quality property in [21,36]). Moreover, our property can be seen as a 'block-based' variant of the time-based *liveness* property defined in [21,36]. An example for instant finality is Algorand [12] which stands in contrast to, e.g., Bitcoin which only guarantees eventual finality, while example of a public NIZK parameter setup is a Perpetual Powers of Tau ceremony, as used in Zcash [39].

5.2 FairTraDEX Algorithms

Each player P_i owns (has exclusive access to) a set of token balances bal_i which are stored as a global variable. For a token tkn, $bal_i(tkn)$ is the amount of token tkn that P_i owns. Keeping the notation from Section 3.1, outputs included in round brackets () are known only to the player running the algorithm, with all other outputs posted to the public bulletin board, updating existing variables/balances where appropriate. Algorithm outputs are not signed, so players observing the output of an algorithm instance can only infer information about the player running the algorithm from public outputs and any corresponding global variable updates.

We now outline FairTraDEX as a set of algorithms: Setup(), Register(), CommitClient(), CommitMM(), RevealClient(), RevealMM() and Resolution(). A FairTraDEX instance is initialised by running Setup(), and proceeds indefinitely in rounds of three distinct, consecutive phases: *Commit*, *Reveal* and *Resolution*, each of length T blocks (see Section 5.1). For readability, we provide here the intuition to the algorithms of FairTraDEX, with a detailed explanation of each algorithm provided in Appendix E.

Players in the underlying blockchain protocol can enter FairTraDEX as clients by running an instance of Register(), which for a given client deposits an escrow $escrow_{client}$, and generates private information $(S,\ \ r \in \{0,1\}^{O(\kappa)})$ which is used in CommitClient() to prove that the client indeed deposited an escrow, without revealing which deposit.

In the Commit phase, all players can run any number of CommitClient() and/or CommitMM() instances. CommitClient() generates a client order, commits to that order publicly and proves in ZK that the player deposited an escrow. If such a proof cannot be generated, or a proof has already been generated for the same S, no order can be committed. A correctly run CommitMM() instance generates a market for a prospective MM, commits to that market publicly and deposits an escrow $escrow_{MM}$.

In the Reveal phase, players can run any number of RevealClient() and/or RevealMM() instances. RevealClient() publishes an order generated through CommitClient(), returning the escrow corresponding to the CommitClient() instance, and as such the Register() instance, to the client. RevealMM() publishes a market corresponding to a CommitMM() instance, and returns the corresponding escrow. Both Reveal phase algorithms assert that the client and MM have sufficient token balances to submit their order and market respectively. These assertions are also ensured in the Commit phase, but must be rechecked to ensure correct balances at the point of token transfer.

In the Resolution phase, any number of Resolution() instances can be run. The first correct Resolution() instance selects the tightest market from the set of revealed markets, *revealedMkts*, for inclusion in order settlement, and any tie-breaks settled using $h(revealedMkts)$, as a random seed. The clearing price which maximises notional traded, and then minimises the notional imbalance of the remaining market and orders is computed. A precise algorithm for verifying the clearing price is included in Appendix G.5, Algorithm 4, and described in Appendix F. Orders and markets are then settled based on this clearing price. Finally, the arrays tracking active commitments, orders and markets *clientCommits*, *MMCommits*, *revealedOrders*, *revealedMkts* are cleared, so unsuccessfully revealed commitments during this round cannot be used to run

RevealClient() or RevealMM() in future rounds. This effectively destroys the deposited escrows of such commitments.

5.3 Properties of FairTraDEX

We now argue that FairTraDEX possesses all of the necessary properties to instantiate a WSFBA, and discuss the practical implications of these properties. As the Register() and CommitClient() algorithms are constructed analogously to the Mint() and Spend() functions in [33], we can make use of the results as provided therein. These can be translated informally as:

1. Linkability: Consider a player P_j, a set of registrations *RegIDs* to which P_j does not know the privately committed values, and a valid ZK signature of knowledge π and serial number S corresponding to some $regID_i \in RegIDs$. P_j in has no advantage in linking π and S to the corresponding $regID_i$ over probability $\frac{1}{|RegIDs|} + negl(\kappa)$.
2. Double-spending: Given a set of registrations *RegIDs*, and any number of valid $(\pi,\ S)$ pairs corresponding to elements in *RegIDs*, it is computationally infeasible to generate a new serial number S' and corresponding valid proof of registration $\pi\ '$ in *RegIDs*.

Given that all players in the system are registered as clients, by definition of MIFP, the expected trade imbalance implied by their orders is 0. However, in reality, we cannot always expect this level of client participation, with less client registrations typically resulting in a greater advantage for rational players in predicting the implied trade imbalance of committed orders.

To account for this in our analysis, we introduce n_ψ denoting the minimal number of registrations required to guarantee that EEV is $negl(\psi)$. Note that in certain blockchain systems, as the total number of players may be unknown to players within the system, precisely defining n_ψ may not be possible (for example, a player registering for the second time observes a smaller relative increase than a player registering for the first time). In that sense, our analysis demonstrates that the listed desirable properties can be achieved under a sufficient level of registration (n_ψ registrations), but not necessarily that a client can detect whether this level is met in a given auction instance. In practice, a client's decision whether or not to commit to an order in FairTraDEX will be based on heuristics involving the number n_c of observed client registrations, noting that non-negligible EEV may be tolerable if the total expected participation fees are less than f_{mcf}.

When players are rational, however, running these algorithms might not be an SNE. Only if following a protocol is an SNE can we be sure that rational players correctly follow the protocol. Towards proving FairTraDEX forms an SNE for all rational clients and MMs, we prove a serious of Lemmas that we use to prove the main result of the section, Theorem 2. Due to space restrictions, we provide here an intuition for these Lemmas, while formally stating and proving them in Appendix D. We first prove that some player in the blockchain protocol runs a Resolution() instance every round. Then, we prove a serious of Lemmas demonstrating that given a rational client (resp. MM) runs an instance of Register() (resp. CommitMM()), that same player correctly runs CommitClient() and RevealClient() (resp. RevealMM()) in the proceeding phases. Finally, we show that it is indeed an SNE for a client (resp. MM) to run Register() (resp. CommitMM()).

With these results in hand, we have it that rational clients and rational MMs correctly execute all algorithms as outlined by FairTraDEX. We now show that with at least n_ψ Register() calls, the optimal strategy for a client is to submit market orders, while the optimal strategy for a MM with MIFP $y_{A\rightarrow B}$ is to show a market *bid* @ *offer* with *bid* $\approx y_{A\rightarrow B} \approx$ *offer* in the case where there are at least 2 non-cooperative MMs, and of width $w \leq f_{mcf}$ otherwise.

Theorem 2. *Consider an instance of FairTraDEX between A_{tkn} and B_{tkn} with MIFP $y_{A\rightarrow B}$ and at least n_ψ previously called instances of Register(). For N non-cooperative MMs, the following strategies form strict Nash equilibria:*

– $N = 1$*: Clients run Register(), followed by CommitClient() producing market orders of width f_{mcf}. The MM runs CommitMM() producing a market of width at most f_{mcf} with reference price equal to $y_{A\rightarrow B}$ in size Q_{not}. Clients and MMs then run RevealClient() and RevealMM() respectively.*
– $N \geq 2$*: Clients run Register(), followed by CommitClient() producing market orders of width greater than 1. MMs run CommitMM() producing markets of width 1 with reference price equal to $y_{A\rightarrow B}$ in size of at least Q_{not}. Clients and MMs then run RevealClient() and RevealMM() respectively.*

Although providing width- markets may seem prohibitive for MMs, the unique guarantees of FairTraDEX ensure that no players external to the protocol can extract value from players within the protocol. As player value is being retained within the FairTraDEX protocol, fees can be introduced to compensate MMs. Given the potential value retention of FairTraDEX (see Section 6, Table 2), these fees can be substantial while still ensuring FairTraDEX provides clients with best-in-class liquidity.

Remark 1. To minimise expected absolute trade imbalances in a DEX auction, existing protocols, including FairTraDEX, require the hiding/mixing of order-information. Consider how FairTraDEX compares to previous DEXs aimed at ensuring client privacy [4,14,20]. In these previous protocols, each order commit reveals the same, and in some cases more information per-order than a Register() call in FairTraDEX. EEV protection guarantees in these previous protocols which require n_ψ orders per auction are achieved in FairTraDEX for every order in every auction when n_ψ players are registered to participate in the protocol. This is an n_ψ factor improvement in EEV protection/block-space requirements per auction.

More than this, these previous protocols face liveness issues when players are concerned about EEV. The first players entering one of these previous protocols must choose to do so without any guarantees of protection against EEV attacks based on information leaked from order commitment (trade direction, identity, trading patterns, etc.).

5.4 Smart Contract Implementation of FairTraDEX

A blockchain-based pseudo-code implementation of FairTraDEX, and code description, are provided in Appendix G, while a Solidity implementation of FairTraDEX is provided in [31]. We outline here the key differences between the algorithmic description of Section 5.2, and the blockchain-based implementations of Appendix G and [31]. As a blockchain-based implementation under the model of Section 5.1 involves a PKI for message sending, all public algorithm outputs must now be signed using the PKI. These messages must now be included in blockchain transactions, with a transaction fee required to ensure the transaction gets added to the blockchain.

For a player to publish a transaction to a blockchain-based smart contract without revealing her identity, she must utilise a relayer (for details on relayers, see Appendix B.2). Otherwise, the transaction fee is payable from her account, revealing sensitive information such as trading patterns and account balances. Furthermore, this relayer must be rewarded on-chain for relaying the transaction. This reward is added by the client when depositing her escrow, and retrievable by the first relayer publishing the transaction to the blockchain. Furthermore all checks, such as those for the previous use of serial numbers in CommitClient(), or the recording of the tightest MM width in Resolution(), are explicitly encoded in the provided implementations.

6 Cost-Benefit Analysis of FairTraDEX

The aim of this section is to demonstrate the contributory significance of FairTraDEX vs. current state-of-the-art protocols as introduced in Section 2. In Table 1 we include an overview of the gas costs for running FairTraDEX compared to the previous blockchain-based attempts to implement batch auctions of [20,14], with numbers taken from the respective papers. It can be seen that FairTraDEX has a slightly greater upfront gas cost for clients, but a lesser cost for MMs.

To demonstrate the benefits of FairTraDEX, Table 2 compares specific swaps that allow for EEV attacks in existing state-of-the-art protocols. We perform our analysis on ETH/USDC swaps, as this is the highest volume pool on Uniswap, which at time of writing had pool sizes of 120k ETH and 185M USDC, an indicative MIFP of 1 ETH equal to 1,540 USDC [40]. Furthermore, we use a gas cost of 7 gwei [17].

Consider 3 buy ETH orders of 10k, 500k and 10M USDC from 2 different players who are known to need to trade at any price. P_1 has large quantities of both ETH and USDC, and buys or sells ETH pseudo-randomly, while P_2 only owns USDC/only buys ETH. We take the estimated impact for each order to be 0, 0.15% and 1% respectively, numbers taken from the Uniswap V3 API [40] (these are more realistic impacts than those implied by the constant product impact [1] of 0, 0.54% and 11.1% respectively, using starting pool sizes of). Although this is a simplification of order impact, true impact is likely some multiple/factor of this impact. Protocol fees incentivizing MMs to provide liquidity are omitted as they are not considered in the provided academic protocols. After gas costs, this fee should be approximately equal for all protocols (the Uniswap fee for this pool is 0.3%).

	FairTraDEX [6]	Uniswap	[14]	[20]
Register	112,800	-	87,000	-
Commit Client	344,500	-	52,000	276,150
Commit MM (per order)	24,300	-	52,000	276,150
Reveal (per order)	172,000	190,000	171,000	48,750
Settle (per order) [7]	45,500	-	122,500	54,000
Total Client	674,800	190,000	432,500	378,900
Total MM	266,100	-	397,500	649,050
Total Client (USDC)	7.27	2.05	4.66	4.08
Total MM (USDC)	2.87	-	4.09	7

Table 1: Comparison of gas costs in batch-auction implementations. [2] Costs provided for FairTraDEX are amortised over 128 client orders and 8 markets. [3] We add an estimated cost for token transfer from smart contract to player of 40,000 to the figures provided in [20] to standardise the costs therein with those of FairTraDEX and [14].

	FairTraDEX	Uniswap	[14]	[20]
P_1-10,000	0	50	0	0
P_2-10,000	0	50	0	0
P_1-500,000	0	3000	750	0
P_2-500,000	0	3000	750	750
P_1-10,000,000	0	150,000	100,000	0
P_2-10,000,000	0	150,000	100,000	100,000

Table 2: Comparison of execution costs in USDC of batch-auction implementations.

When P_1 submits an order in FairTraDEX or [20], no information is gained about the direction of the trade. However, in [14], direction is revealed. As such, any blockchain participant can front run that impact on all other markets, and thus the MIFP for any MM responding to the order will be the impacted MIFP. When P_2 submits an order in either of [14,20] the direction is known, and the MIFP is impacted in the same way as for P_1, **before** any player interacts with P_2, giving P_2 a worse price. Using estimated price impacts of 0, 0.15% and 1%, Table 2 demonstrates the costs of executing these swaps, excluding transaction fees, in these protocols, and Uniswap. For Uniswap, we must also add the recommended slippage, an additional 0.5% of the order size, as it is always in a block producers interest to give Uniswap players worst execution. It can be seen that these costs become increasingly more significant as order size increases, dominating the differences in gas costs of Table 1.

Although Table 2 can be seen as simplifying how orders are handled, it demonstrates two crucial motivators for our work. Firstly, any information revealed about clients before a trade is agreed can, is and will continue to be used against clients. Furthermore, this cost is not necessarily paid to the MM. As orders are committed in public, any blockchain participant can use the committed information to front run the impact on the MIFP before the client or MM has an opportunity to trade, extracting money from the DEX protocol. Secondly, as the effects of these value-extraction techniques increase super-linearly in order-size, a protocol with the value-extraction guarantees of FairTraDEX is needed to allow typically large clients to utilise the benefits of DEXs, and blockchain protocols in as a whole, at a fixed cost, as demonstrated in Table 1, without incurring the prohibitive execution costs of previous solutions, as demonstrated in Table 2.

7 Conclusion

We provide FairTraDEX, a blockchain-based DEX protocol based on WSFBAs in which we formally prove the strategies of rational participants have strict Nash equilibria in which all trades occur at the MIFP plus or minus bounded upfront costs (specified market widths) which approach 0 in the presence of non-cooperative MMs. This is an attractive alternative to existing mainstream protocols such as AMMs where rational players effectively and systematically prevent such an equilibrium from happening. Compared to previous blockchain-based attempts to implement EEV-proof DEXs, FairTraDEX is the first to practically allow for indistinguishable client-order submissions by decoupling order submission from escrow deposit and order revelation. The FairTraDEX benefits formalised in Section 5.3, summarised in Remark 1, and demonstrated in Section 6 provide important improvements on previous protocols regarding EEV protection, setting a new standard for EEV protection in DEXs.

As stated in the comparisons of Section 6, protocol fees are omitted for all protocols. Given the total retention of value within the FairTraDEX protocol (no extractable value), fees in line with the utility gained by clients for exchanging their tokens can be charged to incentivise the long-term participation of MMs in FairTraDEX. These fees should reflect the need to incentivize MMs while retaining the unique client-side benefit of trading at the MIFP in expectation, which is proven to occur in FairTraDEX. Analysis of these fees makes for interesting future work.

References

1. Angeris, G., Kao, H.T., Chiang, R., Noyes, C., Chitra, T.: An analysis of uniswap markets. In: Cryptoeconomic Systems Journal (2019)
2. Asayag, A., Cohen, G., Grayevsky, I., Leshkowitz, M., Rottenstreich, O., Tamari, R., Yakira, D.: Helix: A fair blockchain consensus protocol resistant to ordering manipulation. IEEE Transactions on Network and Service Management **18**(2), 1584–1597 (2021). https://doi.org/10.1109/TNSM.2021.3052038

3. Baric, N., Pfitzmann, B.: Collision-free accumulators and fail-stop signature schemes without trees. In: Proceedings of the 16th Annual International Conference on Theory and Application of Cryptographic Techniques. p. 480–494. EUROCRYPT'97, Springer-Verlag, Berlin, Heidelberg (1997)
4. Baum, C., David, B., Frederiksen, T.: P2DEX: Privacy-preserving decentralized cryptocurrency exchange. In: Sako, K., Tippenhauer, N.O. (eds.) Applied Cryptography and Network Security. pp. 163–194. Springer International Publishing (2021)
5. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474. IEEE Computer Society, New York, NY, USA (2014)
6. Benaloh, J., de Mare, M.: One-way accumulators: A decentralized alternative to digital signatures. In: Helleseth, T. (ed.) Advances in Cryptology — EUROCRYPT '93. pp. 274–285. Springer Berlin Heidelberg, Berlin, Heidelberg (1994)
7. Benarroch, D., Campanelli, M., Fiore, D., Gurkan, K., vKolonelos, D.: Zero-knowledge proofs for set membership: Efficient, succinct, modular. In: Borisov, N., Diaz, C. (eds.) Financial Cryptography and Data Security. pp. 393–414. Springer Berlin Heidelberg, Berlin, Heidelberg (2021)
8. Boneh, D., Bünz, B., Fisch, B.: Batching techniques for accumulators with applications to iops and stateless blockchains. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019. pp. 561–586. Springer International Publishing, Cham (2019)
9. Budish, E., Cramton, P., Shim, J.: The High-Frequency Trading Arms Race: Frequent Batch Auctions as a Market Design Response *. The Quarterly Journal of Economics **130**(4), 1547–1621 (07 2015). https://doi.org/10.1093/qje/qjv027, `https://doi.org/10.1093/qje/qjv027`
10. Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology. p. 61–76. CRYPTO '02, Springer-Verlag, Berlin, Heidelberg (2002)
11. Chase, M., Lysyanskaya, A.: On signatures of knowledge. In: Proceedings of the 26th Annual International Conference on Advances in Cryptology. p. 78–96. CRYPTO'06, Springer-Verlag, Berlin, Heidelberg (2006), `https://doi.org/10.1007/11818175_5`
12. Chen, J., Micali, S.: Algorand: A secure and efficient distributed ledger. Theoretical Computer Science **777**, 155–183 (2019)
13. Ciampi, M., Ishaq, M., Magdon-Ismail, M., Ostrovsky, R., Zikas, V.: Fairmm: A fast and frontrunning-resistant crypto market-maker. Cryptology ePrint Archive, Report 2021/609 (2021), `https://ia.cr/2021/609`, retrieved: 02/02/2022
14. Constantinides, T., Cartlidge, J.: Block auction: A general blockchain protocol for privacy-preserving and verifiable periodic double auctions. In: 2021 IEEE International Conference on Blockchain (Blockchain). pp. 513–520. IEEE Computer Society, United States (2021). https://doi.org/10.1109/Blockchain53845.2021.00078
15. Costello, C., Fournet, C., Howell, J., Kohlweiss, M., Kreuter, B., Naehrig, M., Parno, B., Zahur, S.: Geppetto: Versatile verifiable computation. In: 2015 IEEE Symposium on Security and Privacy. pp. 253–270. IEEE, United States (2015). https://doi.org/10.1109/SP.2015.23
16. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning, transaction reordering, and consensus in-

stability in decentralized exchanges. https://arxiv.org/abs/1904.05234 (2019), retrieved: 19/01/2022
17. Etherscan: https://etherscan.io/gastracker
18. Flashbots: https://explore.flashbots.net
19. Flashbots: https://explore.flashbots.net/faq
20. Galal, H.S., Youssef, A.M.: Publicly verifiable and secrecy preserving periodic auctions. In: Bernhard, M., Bracciali, A., Gudgeon, L., Haines, T., Klages-Mundt, A., Matsuo, S., Perez, D., Sala, M., Werner, S. (eds.) Financial Cryptography and Data Security. FC 2021 International Workshops. pp. 348–363. Springer Berlin Heidelberg, Berlin, Heidelberg (2021)
21. Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Advances in Cryptology - EUROCRYPT 2015. pp. 281–310. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)
22. Gjøsteen K., Raikwar M., W.S.: Pribank: Confidential blockchain scaling using short commit-and-proof nizk argument. In: Topics in Cryptology – CT-RSA 2022. CT-RSA 2022. Springer International Publishing, Cham (2022). https://doi.org/10.1109/SP.2015.23
23. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. In: Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing. p. 291–304. STOC '85, Association for Computing Machinery, New York, NY, USA (1985). https://doi.org/10.1145/22145.22178, https://doi.org/10.1145/22145.22178
24. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016. pp. 305–326. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
25. Gurkan, K., Jie, K.W., Whitehat, B.: Community proposal: Semaphore: Zero-knowledge signaling on ethereum (2020), https://github.com/appliedzkp/semaphore, retrieved: 25/01/2022
26. Judmayer, A., Stifter, N., Schindler, P., Weippl, E.: Estimating (miner) extractable value is hard, let's go shopping! (2021), https://ia.cr/2021/1231
27. Kelkar, M., Zhang, F., Goldfeder, S., Juels, A.: Order-fairness for byzantine consensus. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology - CRYPTO 2020. pp. 451–480. Springer International Publishing, Cham (2020)
28. Loesch, S., Hindman, N., Richardson, M.B., Welch, N.: Impermanent loss in uniswap v3 (2021). https://doi.org/10.48550/ARXIV.2111.09192, https://arxiv.org/abs/2111.09192
29. Massacci, F., Ngo, C.N., Nie, J., Venturi, D., Williams, J.: Futuresmex: Secure, distributed futures market exchange. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 335–353. IEEE, United States (2018). https://doi.org/10.1109/SP.2018.00028
30. McMenamin, C., Daza, V., Pontecorvi, M.: Achieving State Machine Replication without Honest Players, p. 1–14. Association for Computing Machinery, New York, NY, USA (2021), https://doi.org/10.1145/3479722.3480986
31. McMenamin, C., O'Donoghue, P.: https://github.com/MEVProof/Contracts (2022)
32. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) Advances in Cryptology — CRYPTO '87. pp. 369–378. Springer Berlin Heidelberg, Berlin, Heidelberg (1988)
33. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed e-cash from bitcoin. In: 2013 IEEE Symposium on Security and Privacy. pp. 397–411. IEEE Computer Society, United States (2013). https://doi.org/10.1109/SP.2013.34

34. Nguyen, L.: Accumulators from bilinear pairings and applications. In: Proceedings of the 2005 International Conference on Topics in Cryptology. p. 275–292. CT-RSA'05, Springer-Verlag, Berlin, Heidelberg (2005), https://doi.org/10.1007/978-3-540-30574-3_19
35. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V.: Algorithmic Game Theory. Cambridge University Press, Cambridge (2007)
36. Pass, R., Seeman, L., abhi shelat: Analysis of the blockchain protocol in asynchronous networks. In: Coron, J.S., Nielsen, J.B. (eds.) Advances in Cryptology – EUROCRYPT 2017. pp. 643–673. Springer International Publishing, Cham (2017)
37. Qin, K., Zhou, L., Gervais, A.: Quantifying blockchain extractable value: How dark is the forest? (2021). https://doi.org/10.48550/ARXIV.2101.05511, https://arxiv.org/abs/2101.05511
38. Roughgarden, T.: Transaction fee mechanism design for the ethereum blockchain: An economic analysis of eip-1559. https://arxiv.org/pdf/2012.00854 (2020), retrieved: 18/05/2021
39. of Tau, P.P.: https://zkproof.org/2021/06/30/setup-ceremonies/
40. Uniswap: https://app.uniswap.org/
41. Zhang, Y., Katz, J., Papamanthou, C.: An expressive (zero-knowledge) set accumulator. In: 2017 IEEE European Symposium on Security and Privacy (EuroS P). pp. 158–173. IEEE, United States (2017). https://doi.org/10.1109/EuroSP.2017.35

A Extended Related Work

Estimating (Miner) Extractable Value is Hard [26]

This paper paper attempts to formalise extractable value and generalise it beyond value extractable be miners. We also believe it is necessary to model the decision of all rational players based on *expected extractable value* (EEV) that can generated by particular orderings of transactions/blocks by any player in the system, and not just the miner. The approach taken is to consider EEV as the maximum of all non-protocol strategies, with protocols considered secure if the EEV of following the protocol is strictly dominated by following the protocol strategy, which is further formalised in [30]. In our paper, we also consider an additional case of EEV not necessarily considered in [26] which is prevalent in commit-reveal protocols such as [13]. In such protocols, honest behaviour usually involves sending a valid second transaction (the reveal transaction in a commit-reveal protocol), but where players can extract value in expectancy by not sending these transactions. However, we believe the definition of EEV in [26] can be extended to include these attacks. As such, we also move away from the legacy use of MEV, and focus instead on the prevention of the more general EEV.

Publicly Verifiable and Secrecy Preserving Periodic Auctions [20]

This protocol also attempts to implement an FBA, and as such has many similarites to FairTraDEX. As in FairTraDEX, the protocol progresses in rounds of Commit, Reveal and Resolution phases. In [20], there is a designated *operator* who is in charge of settling the auction. Players commit to orders in the Commit phase, as well as providing cryptographic information, which is used to prove correct settlement in the Resolution phase. Unlike FairTraDEX, these commit messages are sent by players directly to the blockchain, revealing identity and trade direction. In the Reveal phase, players encrypt their orders using the operator's public key, and send the encryptions to the operator. In the Resolution phase, the operator then chooses a clearing price which intersects the buy and sell liquidity, maximising the notional to be traded. The operator then publishes a list of all matched orders to the blockchain, along with a range proof which is used to verify the correct execution of orders, while not revealing any information about unexecuted orders other than that already revealed in the commit phase.

Block Auction [14]

This protocol attempts to implement an FBA, and is the most similar to FairTraDEX. It is an improvement on [20], with a direct comparison of the two protocols forming the main basis of the justification of [14]. As in [20], the protocol is overseen by an *operator* who is in charge of receiving orders privately from players and correctly executing the auction. As in FairTraDEX, the protocol progresses in rounds of Commit, Reveal and Resolution phases. In the Commit phase, players commit to orders and publish these commitments to the blockchain. Although not revealing the trade direction as is the case in [20], these commit messages are sent by players directly to the blockchain, and as such reveal identity In the Reveal phase, players encrypt their orders using the operator's public key, and send these encryptions to the operator. The operator then publishes all executed orders, while revealing nothing about unexecuted orders. The validity of execution depends on all players who submitted orders verifying that their order should not have been executed given the list of executions.

P2DEX [4]

The P2DEX protocol is an off-chain MPC protocol run by servers where players can submit orders to exchange tokens from one blockchain to another (although it also appears applicable to one blockchain with many tokens). The orders are encrypted using a threshold secret-sharing scheme with each server receiving a unique share. The protocol has mechanisms to identify double-spending of player funds sent to the servers, and deviation (failure/ misbehaviour) of servers, as the MPC matching protocol is publicly verifiable. As such, all players in the blockchain can verify that a set of orders have been matched correctly, or some of the servers deviated from the protocol. The exchange depends on all servers participating in a secret-sharing protocol to match orders, with at least one server being honest/not colluding with other servers. As with [20,14], an emphasis is placed on not revealing unmatched orders.

FairMM [13]

Clients submit orders to a single (monopolistic) MM in an off-chain Σ-protocol. Clients contact the MM with a size, direction (communicated on-chain) and price (communicated off-chain). If the MM accepts, the MM then publishes the trade on the blockchain, otherwise aborts. Client orders are sequentialized so only one order can be executed at a time, preventing the MM from reordering client orders.

FuturesMEX [29]

The FuturesMEX protocol is an MPC version of a DEX with claims of anonymity. In [29], client token balances are kept privately by owners in an off-chain database, so the protocol has limited applicability to a blockchain-based setting. Furthermore, orders are submitted publicly to all participants before being settled. For smaller clients with less connectivity, this is equivalent to showing the order to more-connected counterparties before it is executed. This is a typical source of EEV in existing blockchain protocols (through front-running attacks), and something which is protected against in FairTraDEX.

B Terminology and Useful Definitions

This section contains additional financial and game-theoretical terms used in this paper. Although not mandatory for all readers, this section serves as a useful reference point towards understanding the results and discussions that follow.

- *Decentralized Exchange* (DEX): A distributed marketplace which allows players to swap one token for another.
- *Limit Order*: Specifies an amount of tokens to be bought (sold), and a maximum (minimum) price at which to buy (sell) these tokens. This price is known as the *limit price*.
- *Market Order*: Specifies an amount of tokens to be sold, but no limit price. Market orders are to be executed immediately at the best available price based on the liquidity of buy orders.
- *Direction*: With respect to an order on a market quoted from token A_{tkn} to B_{tkn}, if the order is trying to buy token B_{tkn}, the direction is *buying*, while if the order is trying to sell token B_{tkn}, the direction is *selling*.
- *Forward Price*: This is the price at which a seller delivers a token to the buyer at some predetermined date. In any exchange protocol without instantaneous delivery, the forward price at expected delivery time is the price at which trades should happen. The difference between current (spot) price and forward price is known as *carry*, and can be due to storage/opportunity costs, interest rates, etc. In this paper, we set carry to 0 for complexity and ease-of-notation purposes.

The following definition of expected extractable value is translated from [26] using the terminology of this paper.

Definition 2. *The expected extractable value EEV_i, describes the total value in value units, which is transferred to player P_i in expectation using a certain strategy which produces a transaction, sequence of transactions, or blocks that later become part of the main chain with some probability.*

B.1 Frequent Batch Auctions

As stated in Section 1, FairTraDEX is based on an FBA [9]. FBAs are used in many of the largest centralised exchanges [8]. As FBAs were initially intended for a centralised setting, we consider them being run by a trusted third party (TTP) who enforces the correct participation of all players. In FairTraDEX, the key TTP functionalities needed to instantiate an FBA are replicated using ZK set-membership proofs, incentivisation and a blockchain as a censorship-resistant bulletin-board. We define an FBA here using the terminology of our paper.

Definition 3. *A frequent batch auction (FBA) (sometimes referred to as a periodic auction) involves clients and MMs privately submitting either limit or market orders to the TTP. These orders are collected until a specified deadline. After this deadline, the orders are settled at the clearing price. A single clearing price is chosen which maximises the total notional traded based on the specified sizes and prices of all orders. If there is more supply (quantity of tokens being sold) at the clearing price than demand (quantity of tokens being bought), all sell orders offered at the highest price at or below the clearing price are pro-rated based on size such that supply equals demand at the clearing price. Similarly, if there is more demand than supply at the clearing price, all buy orders bid at the lowest price at or above the clearing price are pro-rated based on size such that demand equals supply at the clearing price. Any limit buy orders below/sell orders above the clearing price are not executed.*

There are two key differences between this definition and the specification in [9]:

1. In our definition, if an order is not fulfilled, it is revealed with any tokens not being sold returned to the seller. This does not affect the game-theoretic guarantees of the paper, as the results in [9] only depend on the hiding of order information while players are submitting orders to the auction.
2. As every order in our auction must be submitted independently for each auction, there is no time priority applied when pro-rating orders in case of a supply-demand imbalance. This is a sub-case of the FBAs as defined in [9], and consequently, our protocol retains the same game-theoretic guarantees.

[8] FCA `https://www.fca.org.uk/publications/research/periodic-auctions`, CBOE `https://www.cboe.com/europe/equities/trading/periodic_auctions_book/`, ESMA `https://www.esma.europa.eu/sites/default/files/library/esma70-156-1035_final_report_call_for_evidence_periodic_auctions.pdf`

Game-Theoretic Guarantees of a Frequent Batch Auction In this section we investigate the properties of an FBA between rational MMs and rational clients, where MMs do not know the desired trade direction of the clients.

We first restate, using the terminology from this paper, the main result from [9] which applies to our game-theoretically equivalent definition of an FBA. To do this, we let D represent the net trade imbalance of clients in a particular instance of an FBA in terms of ₿. A positive D indicates a client buy imbalance (more client buyers than sellers of the swap), while a negative D indicates a client sell imbalance. We require a finite bound on the absolute imbalance, which we denote $Q_{not} < \infty$, for the existence of optimal MM strategies. As in [9], we assume that $|D| \leq Q_{not}$, and in-keeping with the notion of an MIFP, D is symmetric around 0 at the MIFP.

Theorem 3. *[9] For an FBA with at least two non-cooperative MMs, there is a strict Nash equilibrium where clients only submit market orders and MMs show a market of width 1 (bid = offer) centred around the MIFP in size greater than* Q_{not}.

This is a useful result in the case of at least two non-cooperative MMs, with clients receiving a game-theoretic guarantee that they can exchange one token for another at the MIFP in expectancy in an FBA. Furthermore, as MM liquidity is greater than the net client trade size, the implicit impact to these trades in [9] is bounded by the width, which is 1. As clients have a strictly positive utility for exchanging tokens, this is equivalent to clients always having positive expectancy to participate in an FBA. However, it is also shown in this equilibrium that MMs have 0 expected utility. A basic adjustment to the protocol in that setting would then be to charge clients a fee for the service and pro-rate these fees to the MMs to ensure the long-term participation of MMs.

B.2 Relayers

A fundamental requirement for transaction submission in blockchains is the payment of some transaction fee to simultaneously incentivise block producers to include the transaction, and to prevent denial-of-service/spamming attacks. However, in both the UTXO- and account-based models, this allows for the linking of player transactions, balances, and their associated transaction patterns. With respect to DEX protocols, if clients are required to deposit money into a UTXO/account before initiating a trade, any other player in the system can infer who the client is, what balances the client owns, what transactions the client usually performs, etc., and use this information to give the client a worse price.

To counteract this, we utilise the concept of *transaction relayers*[9]. In the smart-contract encoding of FairTraDEX (App. G.5), clients must publicly register to a smart contract, and in doing so, deposit some escrow. In addition to this

[9] Ox `https://0x.org/docs/guides/v3-specification`, Open Gas Station Network `https://docs.opengsn.org/`, Rockside `https://rockside.io/`, Biconomy `https://www.biconomy.io/`

escrow, we also require the clients to deposit a relayer fee. When the client wishes to submit a transaction anonymously to the blockchain, the client publishes a proof of membership in the set of registered clients to the relayer mempool, as well as the desired transaction and a signature of knowledge cryptographically binding the membership proof to the transaction, preventing tampering. As the relayer can verify the proof of membership, the relayer can also be sure that if the transaction is sent to the FairTraDEX contract, the relayer will receive the corresponding fee. With this in mind, a relayer observing the client transaction includes it in a normal blockchain transaction, with the first relayer to include the transaction receiving the fee. As such, relayers are a straightforward extension of the standard transaction-submission model. Furthermore, if the proof of membership is NIZK and the message is broadcast anonymously (using the onion routing (Tor) protocol[10] for example), the relayer can only infer that the player sending the transaction is a member of the set of clients.

C Background on Zero-Knowledge Primitives

Proving membership has been traditionally solved using cryptographic accumulators [6], where the prover P computes a value (the accumulator) and, based on it, a set of short membership proofs that the verifier V can easily verify. Three are the approaches to construct set membership proofs: Merkle trees [32], RSA accumulators [3,8], and pairing-based accumulators [34,41].

Each approach has its own benefits for public parameters, accumulator or witness size or need of trusted setup. The exact choice depends on the resource constraints of the system. We direct interested readers to [7] for a nice review of the main features of each of the approaches.

When the prover P does not want to reveal the value of x, the membership proof should not leak any information on the value of x. At a high level, the general approach is to guarantee privacy using zero knowledge proofs. Zero knowledge proofs [23] are powerful cryptographic primitives that allow a prover P to prove knowledge of the truth of some statement without revealing the statement contents, to some honest verifier V who needs to be convinced of the truth of the statement provided by the prover. Special mention for its applicability should be made of zero-knowledge proofs that are also non-interactive, that is, proofs that only depend on the prover's private information about the statement and publicly available information[11]. As such, proofs do not depend on interaction with the verifier. The main features in a NIZK argument are completeness, soundness, and zero-knowledge. Completeness guarantees that if the statement is true, the prover behaving honestly can convince the verifier that the statement is true, while soundness ensures that a dishonest prover cannot convince an

[10] `https://www.torproject.org/`

[11] This public information can come in many forms, but in [33,5,24,7], it must be generated honestly in a process known as a *trusted setup*. If a prover knows the private information used to generate public proof parameters, the knowledge extraction property cannot exist.

honest verifier. Zero-knowledge maintains that the only information learned by the verifier is that the statement is true. However, in practice it is required that the prover knows a witness for the statement, that is, a zero-knowledge proof of knowledge. In this case, soundness is not enough and is required that a prover cannot produce a valid proof unless she knows a witness, even if the prover has seen an arbitrary number of simulated proofs. This is what is known, as simulation intractability. Furthermore, NIZK arguments are interesting for constructing other cryptographic primitives, such as signature of knowledge (SoK) [11].

In literature, there are several constructions that add the privacy layer using zero knowledge proofs for set membership based on RSA Accumulators or Merkle Trees are [10,33,5]. In these works the prover proves statements about values that are committed, that is, they follow what is known as a commit-and-prove zero knowledge proof. More recent approaches propose new commit-and-prove for set-membership based on SNARKs [15,7] or Bulletproofs [22].

D Proofs

Lemma 1. *For an MM in a WSFBA between A_{tkn} and B_{tkn} with MIFP equal to $y_{A\to B} = \frac{y_B}{y_A}$ and a client order of notional $X₿ > 0$, she strictly maximizes her expected utility by showing a market with reference price $y_{ref} = y_{A\to B}$ for any fixed width $w \geq 1$.*

Proof. Let us define the market in terms of y_{ref} and w as described in Section 3, namely, $\frac{y_{ref}}{\sqrt{w}}$ @ $\sqrt{w}y_{ref}$. In the cases of a client buyer and client seller of the swap, we convert client notional orders into the tokens being sold, mark the trades to their respective MIFPs using a multiplicative market-impact coefficient for the token swap of δ, then reconvert the tokens into notional.

If the client is a buyer of the swap, the client is selling A_{tkn}, with trade size of $\frac{X₿}{y_A}$ in A_{tkn}. The trade occurs on the token swap offer of $\sqrt{w}y_{ref}$, resulting in the sale of $\frac{X₿}{y_A}\frac{1}{\sqrt{w}y_{ref}}$ token B_{tkn}s by the MM. Finally, the B_{tkn}s bought by the client have an expected per-token value of $\sqrt{\delta}y_B$, while the notional acquired by the MM (A_{tkn}s) has an expected value of $\frac{X₿}{\sqrt{\delta}}$. This is an expected net profit for the MM measured in notional of:

$$\frac{X₿}{\sqrt{\delta}} - \frac{X₿}{y_A\sqrt{w}y_{ref}}\sqrt{\delta}y_B = \frac{X₿}{\sqrt{\delta}} - \frac{\sqrt{\delta}X₿}{\sqrt{w}}\frac{y_{A\to B}}{y_{ref}}. \tag{1}$$

If the client is a seller of the swap, the client is selling B_{tkn}, with trade size of $\frac{X₿}{y_B}$ in B_{tkn}. The trade occurs on the token swap bid of $\frac{y_{ref}}{\sqrt{w}}$, resulting in the sale of $\frac{X₿}{y_B}\frac{y_{ref}}{\sqrt{w}}$ token A_{tkn}s by the MM. Finally, the A_{tkn}s bought by the client have an expected per-token value of $\sqrt{\delta}y_A$, while again, the notional acquired by the MM (B_{tkn}s) has an expected value of $\frac{X\text{₿}}{\sqrt{\delta}}$ This is an expected net profit for the MM of:

$$\frac{X\text{₿}}{\sqrt{\delta}} - \frac{\sqrt{\delta}X\text{₿}}{\sqrt{w}}\frac{y_{ref}}{y_{A\to B}}. \tag{2}$$

We know the expected buying and selling of X₿ notional at $y_{A\to B}$ are equally likely by the definition of an MIFP as a perfectly-informed signal. Therefore, the total expected profit is:

$$X\text{₿}\Big(\frac{1}{2}(\frac{1}{\sqrt{\delta}} - \frac{\sqrt{\delta}}{\sqrt{w}}\frac{y_{A\to B}}{y_{ref}}) + \frac{1}{2}(\frac{1}{\sqrt{\delta}} - \frac{\sqrt{\delta}}{\sqrt{w}}\frac{y_{ref}}{y_{A\to B}})\Big). \tag{3}$$

To find the maximum with respect to y_{ref}, we take the first derivative of this formula, and let it equal to 0:

$$\frac{y_{A\to B}}{y_{ref}^2} - \frac{1}{y_{A\to B}} = 0. \tag{4}$$

Solving for y_{ref} gives $y_{ref} = y_{A\to B}$, which is equivalent to the MM strictly maximizing her expected profits by letting $y_{ref} = y_{A\to B}$.

Theorem 1. *For a WSFBA, the strict Nash equilibria strategies given the number of non-cooperative MMs submitting markets being N are:*

- $N = 1$*: Clients submit market orders of requested width f_{mcf} and the MM shows a market of width at most f_{mcf} with reference price equal to the MIFP.*
- $N \geq 2$*: Clients submit market orders of requested width greater than 1 and MMs show a market of width 1 with reference price equal to the MIFP.*

Proof. We now investigate each of the cases described in the theorem statement in terms of the number of non-cooperative MMs N.

$N = 1$: Consider first the strategy of a client. Although clients are not necessarily aware of the MIFP, let us consider their strategies taking the MIFP p as a variable with an arbitrary distribution. For buy orders, the strategy of submitting a limit order with price p less than $\sqrt{f_{mcf}}y$ is dominated by all prices greater than p and less than or equal to $\sqrt{f_{mcf}}y$. For limit sell orders, this limit is $\frac{y}{\sqrt{f_{mcf}}}$. As such, the equilibrium for clients involves submitting orders equivalent to a market of width equivalent to at least f_{mcf} with reference price equal to the MIFP. If a client knows a MM submits a market of width less than or equal to f_{mcf} with reference price equal to the MIFP, this strategy is further dominated by submitting a market order with requested width f_{mcf}, as market orders strictly increase the client's probability of trading. Furthermore, any strategy for a client which involves trading on a price outside $[\frac{y}{\sqrt{f_{mcf}}}, \sqrt{f_{mcf}}y]$ is strictly dominated by not trading. As such, the only possibilities for equilibria can occur on a market of $\frac{y}{\sqrt{f_{mcf}}}@\sqrt{f_{mcf}}y$. Furthermore, the submission of market orders (increasing probability of trading) with requested width f_{mcf} is strictly dominant if the MM shows a market with reference price equal to the MIFP in sufficient size to fill all clients' orders. As $Q_{not} > |D|$, this would be the case given the appropriate reference price. In Lemma 1, we have seen that a MM trading on a market against a random client shows a market with reference price equal to the MIFP. Therefore clients submit market orders with requested width f_{mcf}. Moreover, this strategy does not require the client to know the MIFP..

Consider now the MM strategy. As only the tightest market in every auction is included for settlement, the MM only submits one market. Any MM order bidding above/offering below the MIFP has negative expected utility, and as such, no rational MM does this this. Also, by definition, the MM must show a market in size $Q_{not} \geq |D|$, meaning the MM has sufficient notional on the bid and offer to trade all client orders and as such the clearing price must be inside the provided MM market. Next, we have seen in Lemma 1 that a MM trading on a market against a random client shows a market with reference price equal to the MIFP. Furthermore, from Equation 3 we can see that the expected utility of a MM is strictly increasing in width. Any strategy involving a market with width greater than f_{mcf} cannot be an equilibrium as clients strictly prefer to trade on markets of lesser width, as argued above. Therefore, the MM maximises her expectancy against a random client by showing a market of width f_{mcf} with reference price equal to the MIFP. Against multiple clients, a positive notional imbalance at the MIFP is decreasing in price (resp. a negative notional imbalance is increasing as price decreases), which may cause the MM to provide a market of width less then f_{mcf}.

Consider the strategy of a MM providing a market of width less than or equal to f_{mcf} with reference price equal to the MIFP, and the strategy of clients submitting market orders with requested width f_{mcf}. We have shown that any player deviation strictly decreases that player's expectancy, making this a strict Nash Equilibrium.

$N \geq 2$: As MMs in a standard FBA provide markets of width 1 when the width is not a restriction, applying the requested width adjustments of a WSFBA, further incentivising tighter markets, does not change the unique equilibrium of Theorem 3. Similarly, as there is a unique clearing price when a width-1 market is submitted, it must also minimise the imbalance over prices that maximise total notional traded. The restriction on the notional of markets in a WSFBA is in line with the inequality of Theorem 3. Furthermore, any requested width > 1 in this equilibrium ensures a client's order trading through the MIFP is included in the final auction settlement, with the maximum allocation occurring when a client submits a market order.

Lemma 2. *At least one player runs Resolution() in every round.*

Proof. Consider a Resolution phase where at least one player has not called a CommitClient() or CommitMM() instance in the preceding Commit phase. This player is indifferent to the settlement of orders, and as such the only payoff for that player by running Resolution() correctly is the receipt of $resBounty \in \mathbb{R}^+$.

Consider instead the case where all players in the system called at least one instance of CommitClient() and/or CommitMM() in the preceding Commit phase. In this case, all players have an additional payoff for receipt of the tokens currently locked in the protocol. As at least one of the buyers or sellers of the swap must receive a non-zero amount of tokens, the receipt of which having at worst 0 utility. This, in addition to the receipt of *resBounty* makes the calling of Resolution() positive expectancy for at least one player in the system.

Lemma 3. *A rational client who correctly runs an instance of Register() also runs correct corresponding instances of CommitClient() and RevealClient().*

Proof. By correctly running Register(), a player deposits $escrow_{client}$. The only way to receive $escrow_{client}$ back is to correctly run a RevealClient() instance, which itself can only be run after having run a CommitClient() instance in the previous phase. By construction, $escrow_{client}$ is greater than any incurrable losses by running CommitClient() and RevealClient(), with maximal losses occurring where the client's order is settled for tokens that have 0 notional (price goes to 0). As the client's initial deposit had notional value strictly less than $escrow_{client}$, so must the incurred loss. The result follows.

Lemma 4. *A rational MM who correctly runs an instance of CommitMM() also runs a correct instance of RevealMM() in the proceeding phase.*

Proof. By correctly running CommitMM(), a player deposits $escrow_{MM}$. The only way to receive $escrow_{MM}$ back is to correctly run a RevealMM() instance in the proceeding. By definition, as the MM's bid and offer has notional value greater than or equal to the total notional in the auction, Q_{not}, so must the incurred loss of not revealing a market. Therefore, players running CommitMM() always correctly run RevealMM().

Lemma 5. *Consider an instance of FairTraDEX between A_{tkn} and B_{tkn}. A rational client P_i with $bal_i(Ƀ) > escrow_{client}$ runs an instance of Register().*

Proof. Consider such a client P_i with minimum client utility f_{mcf} to exchange one token for another. To execute the swap, P_i must first call Register(). Given P_i calls Register(), we know from Lemma 3 that P_i also calls CommitClient() and RevealClient(). We know from Lemma 2, Resolution() will be run in every round, meaning P_i either trades or the tokens are returned. Given a trade occurs, P_i realises the utility of trading given the restrictions of the order generated by P_i in CommitClient(), which can be chosen to be any value with positive utility (buy below the MIFP/sell above the MIFP). If no trade occurs, P_i's realised utility (of 0) does not change. Therefore, P_i runs Register().

Lemma 6. *Consider an instance of FairTraDEX between A_{tkn} and B_{tkn}, and at least 1 previously called instance of Register(). Any rational MM P_i with $bal_i(Ƀ) \geq escrow_{MM}$ and $bal_i(A_{tkn}),\ \ bal_i(B_{tkn}) > 0$ runs an instance of CommitMM().*

Proof. Consider such a player P_i. Given Register() was called by some player P_j, P_i knows P_j must call CommitClient() and RevealClient(). Furthermore, P_i

knows the total order size is bounded by Q_{not} (Section 4). Given MIFP $y_{A\rightarrow B}$ and the definition of a MM, there is some market $market \leftarrow (bid@offer)$ at which P_i observes positive utility to trade with P_j. Therefore, P_i submits *market* to the auction. We must now ensure P_i submits the order by calling CommitMM(). By the calculation of order settlement, if P_i submits *market* through the necessary Register() and CommitClient() calls and P_i submits a market order with finite requested width, no trade happens. As such, P_i runs CommitMM().

With these lemmas in hand, we have it that rational clients and rational MMs correctly execute all algorithms as outlined by FairTraDEX. This can be expressed concisely in the following corollary. In this corollary, and the theorem that follows, we assume clients and MMs satisfy the token-balance requirements as described in Lemmas 5 and 6 respectively.

Corollary 1. *Rational clients and MMs always follow the FairTraDEX protocol.*

Observation 5 *It can be seen that FairTraDEX with at least n_ψ previous Register() calls implements a WSFBA when all players follow the protocol. In the Commit phase, CommitClient() specifies a client order which is committed to, while CommitMM() specifies a market which is also committed to. As clients commit to these orders and sign this commitment using a NIZKSoK, nothing is revealed about the client's order, as there are are least n_ψ Register() calls. This is equivalent to privately submitting the order.*

CommitMM() and RevealMM() ensure MMs provide the equivalent of at least Q_{not} notional on the bid and offer. Furthermore, as MMs are indistinguishable (see Section 3), a MM commitment reveals nothing about the liquidity on the bid or offer[12].

When the Commit phase is finished, no further orders or markets can be submitted for that auction round, and as such, the clearing price is predetermined. During the Reveal phase, RevealClient() and RevealMM() reveal the orders corresponding to CommitClient() and CommitMM() instances from the previous phase, which are settled in the Resolution phase according to clearing price rules which maximise the amount of notional to be traded, as is in a WSFBA.

Theorem 2. *Consider an instance of FairTraDEX between A_{tkn} and B_{tkn} with MIFP $y_{A\rightarrow B}$ and at least n_ψ previously called instances of Register(). For N non-cooperative MMs, the following strategies form strict Nash equilibria:*

– $N = 1$*: Clients run Register(), followed by CommitClient() producing market orders of width f_{mcf}. The MM runs CommitMM() producing a market of width at most f_{mcf} with reference price equal to $y_{A\rightarrow B}$ in size Q_{not}. Clients and MMs then run RevealClient() and RevealMM() respectively.*

[12] MMs are also allowed to participate as clients if privacy is a concern. CommitMM() provides professionals with a functionality to efficiently provide liquidity in a decentralised setting. It is possible to introduce a RegisterMM() function analogous to Register(), allowing MMs to relay markets in ZK. We believe this has negligible benefit for professionals who already have price and market-size hidden through the commitment scheme.

– $N \geq 2$*: Clients run Register(), followed by CommitClient() producing market orders of width greater than 1. MMs run CommitMM() producing markets of width 1 with reference price equal to* $y_{A\to B}$ *in size of at least* Q_{not}*. Clients and MMs then run RevealClient() and RevealMM() respectively.*

Proof. From Corollary 1, we know the running of FairTraDEX is a strictly dominant strategy for clients and MMs. Furthermore, from Observation 5, we have seen that FairTraDEX with at least n_ψ previously called instances of Register() implements a WSFBA. Given FairTraDEX is a WSFBA, the statement follows by applying Theorem 1.

E FairTraDEX Algorithms

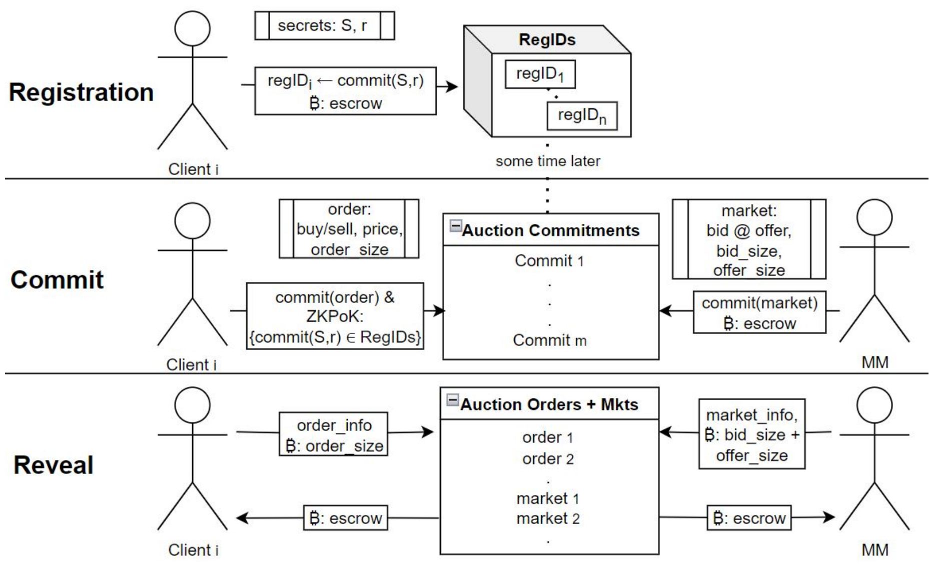

Fig. 1: FairTraDEX phases before order settlement. ₿ : indicates the transfer of some tokens, but not necessarily in the same denomination.

We now describe in detail the algorithms which together form FairTraDEX, and then describe the main differences between FairTraDEX and a WSFBA.

– Setup(κ, Q_{not}) → [*params*, y_A, *RegIDs*, *MMCommits*, *clientCommits*, *revealedOrders*, *revealedMkts*]: For a given cryptographic-security parameter κ, output the necessary public cryptographic and ZK parameters in *params*. Set $y_A \in \mathbb{R}^+$ as the indicative price of token A_{tkn} (used to convert restrictions based on escrows into token amounts). Choose $escrow_{client} \in \mathbb{R}^+$ such that the notional of any client order is bounded by $escrow_{client}$, and $escrow_{MM} \leftarrow$

$c \cdot Q_{not}$, for some $c > 1$, with Q_{not} as described in Section 4. Choose a bounty $resBounty \in \mathbb{R}^+$ to reward players for successfully calling Resolution(). Add $escrow_{client}, escrow_{MM}, Q_{not}, resBounty$ to *params*. Set *RegIDs*, *clientCommits*, *MMCommits*, *revealedOrders*, *revealedMkts* $\leftarrow []$.

– Register$(params, P_i, RegIDs) \rightarrow [(S, r), regID, RegIDs]$: If $\{bal_i(\text{฿}) \geq escrow_{client}\}$, set $bal_i(\text{฿}) \leftarrow bal_i(\text{฿}) - escrow_{client}$. Then, randomly generate $S,\ r \in \{0,1\}^{O(\kappa)}$, and compute $regID \leftarrow h(S, r)$. Add *regID* to *RegIDs*.
– CommitClient$(params,\ regID, RegIDs,\ S,\ r,\ clientCommits,\ y_A) \rightarrow [(order),$ $\pi,\ S,\ com,$ $clientCommits]$: If $\{regID \in RegIDs\}$: select a token $tkn \in \{A_{tkn}, B_{tkn}\}$, a trade price $p \in \mathbb{R}^+$, and a trade size amount $size \in \mathbb{R}^+$. If token $tkn = A_{tkn}$, set $y \leftarrow y_A$. Otherwise, set $y \leftarrow y_A \cdot p$ (used to verify client order is less than the escrow given the indicative price for A_{tkn}). If $\{size \leq bal_i(tkn), \frac{escrow_{client}}{y}\}$: select a minimum trade width $w \geq 1$. Set $order \leftarrow (tkn, size, p, w)$ and $com \leftarrow h(order)$. Finally, generate the signature of knowledge $\pi \leftarrow NIZKSoK[com]\{(regID,\ r) :$ MemVerify $(RegIDs,\ regID) = 1\ \ \&\ \ regID = h(\ S, r)\ \}$. If no proof corresponding to S has been computed before, add S to *clientCommits*. Otherwise, output $\perp$.
– CommitMM$(params, P_i, MMCommits,\ y_A) \rightarrow [(market), com, MMCommits]$: If $\{bal_i(\text{฿}) \geq escrow_{MM}\}$: select a bid size $size_{bid} \in \mathbb{R}^+$, offer size $size_{offer} \in \mathbb{R}^+$ and prices $0 \leq bid \leq offer$ such that $Q_{not}\ \leq size_{bid} {\cdot} y_A \leq escrow_{MM}, (bal_i(A_{tkn}) {\cdot} y_A) \wedge Q_{not}\ \leq size_{offer} \cdot y_A \cdot offer \leq escrow_{MM}, (bal_i(B_{tkn}) \cdot y_A \cdot offer)$ (ensures the bid, offer prices, and sizes fall within the bounds of the minimum size required by a WSFBA and the escrow). Set $market \leftarrow (bid,\ size_{bid},\ offer,\ size_{offer})$ and compute $com \leftarrow h(market)$. Finally, set $bal_i(\text{฿}) \leftarrow bal_i(\text{฿}) - escrow_{MM}$ and add *com* to *MMCommits*. Otherwise, output $\perp$.
– RevealClient$(params, \pi, S, r, order\ \leftarrow\ (tkn, size, p,\ w,\ y_A)\ , clientCommits,$ $revealedOrders,\ P_i\ ) \rightarrow$ $[revealedOrders]$: If token $tkn = A_{tkn}$, set $y \leftarrow y_A$. Otherwise, set $y \leftarrow y_A \cdot p$. If $\{\pi \in clientCommits\ \wedge\ \pi = NIZKSoK[h(order)]\{(regID, r)$: MemVerify$(RegIDs, regID) = 1\ \ \&\ \ regID = h(S, r)\ \} \wedge size \leq bal_i(tkn), \frac{escrow_{client}}{y}$ (Repeat checks from CommitClient()): set $bal_i(tkn) \leftarrow bal_i(tkn) - size$, $bal_i(\text{฿}) \leftarrow bal_i(\text{฿}) + escrow_{client}$ (return escrow), and add *order* to *revealedOrders*. Otherwise, output $\perp$.
– RevealMM$(params, com, market \leftarrow (bid,\ size_{bid},\ offer,\ size_{offer}), y_A, MMCommits,$ $revealedMkts, P_i) \rightarrow [revealedOrders]$: If $\{com \in MMCommits,\ com = h(market)$ $\wedge\ Q_{not}\ \ \leq\ size_{bid} \cdot y_A\ \leq\ escrow_{MM}, (bal_i(A_{tkn}) \cdot y_A)\ \wedge\ Q_{not}\ \ \leq\ size_{offer} \cdot y_A \cdot offer\ \leq\ escrow_{MM}, (bal_i(B_{tkn}) \cdot y_A \cdot offer)$ (Repeat checks from CommitMM()): set $bal_i(A_{tkn}) \leftarrow bal_i(A_{tkn}) - size_{bid}$, $bal_i(B_{tkn}) \leftarrow bal_i(B_{tkn}) - size_{offer}$, $bal_i(\text{฿}) \leftarrow bal_i(\text{฿}) + escrow_{MM}$ (return escrow), and add the orders $(A_{tkn}, size_{bid}, bid, any)$, $(B_{tkn},\ size_{offer},\ offer, any)$ to *revealedMkts*. Otherwise, output $\perp$.
– Resolution$(params, revealedOrders, revealedMkts, clientCommits, MMCommits, P_i)$ $\rightarrow [CP,$ $imbalance, w_{tight}\ clientCommits, MMCommits, revealedOrders, revealedMkts]$: If this is not the first time Resolution() was called, output $\perp$. Otherwise,

calculate the tightest market width w_{tight} of a market in *revealedMkts*. Remove all markets in *revealedMkts* except one with width equal to w_{tight}, chosen using *revealedMkts* as a random seed to $h()$ [13]. Remove all orders in *revealedOrders* with requested width greater than w_{tight}. Calculate the clearing price *CP* which first maximises notional of orders traded from *revealedOrders*$\cup$ *revealedMkts*, and then minimises the imbalance *imbalance*.
If *CP* does not maximise notional traded, and then minimise imbalance, output $\perp$. Otherwise, if *imbalance* > 0 at *CP*, pro-rate all buy orders with the lowest bid price above *CP* based on order size. If *imbalance* < 0, pro-rate all sell orders with the highest offer price below *CP* based on order size. Then, settle all other buy orders with price greater than or equal to *CP*, and all other sell orders with price less than or equal to *CP*. Set *clientCommits*, *MMCommits*, *revealedOrders*, *revealedMkts* $\leftarrow []$, and $bal_i(\text{₿}) \leftarrow bal_i(\text{₿}) + resBounty$.

E.1 FairTraDEX vs. WSFBA

The main differences between FairTraDEX and a WSFBA are as follows:

- Escrows are used to enforce the correct revelation of players who commit to orders or markets. Escrows are only returned to players if orders are revealed and correspond to a valid commit. Furthermore, escrows are chosen large enough to ensure the reclamation of escrows has strictly higher utility than not, ensuring rational players follow the protocol.
- An algorithm involving deposits and/or withdrawals updates the set of balances for all players, identifying the player calling the algorithm.
- FairTraDEX separates the depositing of client escrow and client order commitments. This is a key functionality necessary to preserve client anonymity and the guarantees of a WSFBA. If a client deposits an escrow in the same instance as committing to an order, that information can be used to identify the player, and imply information about the player's order. By separating the two, commitment does not require the update of global variables that can be used to identify the client.
- Set-membership proofs in ZK in the CommitClient() algorithm are used to prove that a player committing to a client order has deposited a client escrow. As FairTraDEX separates the deposit and commitment steps, these proofs allow a client who deposited an escrow to generate one (and only one, as ZK proofs reveal *S*) order per escrow, while only revealing that the order corresponds to a deposited escrow. As the number of deposited escrows increases, the probability that an order commitment matches any particular escrow approaches 0. This replicates the anonymous order submission of a WSFBA.

[13] Given all markets are revealed, the final value of *revealedMkts*, and as such $h(revealedMkts||*)$, is unpredictable in the presence of two or more non-cooperative MMs. We prove in Lemma 4 that all MMs running CommitMM() also run RevealMM(). The blockchain based implementation of this function is described in App. G.3

- Tokenised incentives are used to ensure some player in the blockchain calculates the clearing price, and settles orders correctly.

F Clearing Price Verification

The protocol in Algorithm 4 checks that a given clearing price CP clears the highest notional with respect to A_{tkn}. To do this, it checks the imbalance and total notional that would be settled at CP. Note, that the statements that follow are true given some volume trades at the proposed clearing price, which is asserted in the protocol (line 92).

If the imbalance is 0 (line 97), it can be seen that this price must maximise the volume trade while minimising the absolute value of the imbalance. Higher prices reduces the buying notional/ strictly increases the selling notional, while lower prices strictly reduces the selling notional/increases the buying notional, which creates an imbalance without increasing the notional traded.

If the imbalance is positive (line 99), this implies there will be buy orders (partially) unfilled at or above CP. We can see that the only prices at which more notional can trade must be greater than CP. Thus, the contract checks the next price increment above (line 100), and verifies the total notional traded at that price is less than at CP, or equal, but with a larger absolute imbalance (line 109). If the notional traded is lower at this higher price, the clearing price is correct as a lower price reduces the value of the selling notional, and increases the value of the buying notional. If the notional traded is the same, but the absolute value of the imbalance is higher, this imbalance must be a sell imbalance by the same reasoning (buying notional decreases, selling notional increases). If the absolute value of the imbalance is higher (although negative), the imbalances at all price points above that price are increasing (buying notional decreases, selling notional increases).

The same holds for sell imbalances at a proposed clearing price (line 105). Checking the price point below (line 106) and ensuring the notional is lower, or that the notional traded is the same but with a large absolute imbalance (line 109) guarantees that the proposed clearing price is valid.

G Encoding of FairTraDEX

The following is an overview of the blockchain-based encoding of FairTraDEX, as described algorithmically in Section 5 and encoded in Appendix G.5.

For an arbitrary bit-string $m \in \{0,1\}^*$, $\mathit{relay}(m)$ indicates broadcasting m to the relay transaction mempool, where m is included as a transaction in the blockchain if and only if it gives the including relayer access to a relayer fee f_r.

G.1 Register

Clients randomly generate $S,\ r \in \{0,1\}^{n(\kappa)}$, and compute $\mathit{regID} \leftarrow h(S,r)$. Then, the client sends a $\langle \mathit{CLIENT\text{-}REGISTER},\ \mathit{regID} \rangle$ to the blockchain using the blockchains PKI. Upon addition to the blockchain, this deposits an escrow of $\mathit{escrow}_{\mathit{client}}$ and a relayer fee of f_r to the blockchain (line 22), with *regID* added to *clients* (line 23).

G.2 Commit

A client wishing to submit an order of the form $\mathit{order} \leftarrow (\mathit{tkn}, \mathit{size}, p, w)$, first generates a commitment to the order $\mathit{com} \leftarrow h(\mathit{order})$. The client then generates a NIZK signature of knowledge $\pi \leftarrow \mathrm{NIZKSoK}(\mathit{com})\{(\ \mathit{regID}, r) : \mathrm{MemVerify}(\mathit{RegIDs}, \mathit{regID}) = 1\ \ \&\ \ \mathit{regID} = h(S, r)\ \}$ on the commitment. The client then relays a message of the form $\langle \mathit{COMMIT},\ \mathit{com},\ S,\ \pi\rangle$ to the relayer mempool, which is then sent to the smart contract by a relayer. The transaction is only valid if $\mathrm{Verify}(\pi)$ returns 1, and as such, a relayer cannot tamper with *com*. The contract first checks that the maximum auction notional has not been reached ($\mathit{currAucNotional} < Q_{\mathit{not}}$, line 24).

Furthermore, a valid $\langle \mathit{COMMIT}, \mathit{com}, S, \pi\rangle$ message must not reveal a serial number S which has previously been added to *blacklistedSNs* (initialised line 18). Serial numbers in *blacklistedSNs* correspond to client commits that were not revealed according to the protocol during a previous Reveal phase. The escrow corresponding to serial numbers of *blacklistedSNs* are effectively burned by the protocol, with clients permanently losing access to them. If S is not in *blacklistedSNs*, the order commitment is recorded in *clientCommits* (line 26), and the relayer who relays the transaction to the blockchain receives the fee (line 27).

MMs who wish to participate generate a market of the form $\mathit{market} \leftarrow (\mathit{bid},\ \mathit{size}_{\mathit{bid}},\ \mathit{offer},\ \mathit{size}_{\mathit{offer}})$, and submit a transaction directly to the blockchain of the form $\langle \mathit{COMMIT}, h(\mathit{market})\rangle$. This transaction deposits an escrow of $\mathit{escrow}_{\mathit{MM}}$ to the smart contract. The commitment is then recorded in *MMCommits* (line 30).

Client and MM Commit transactions are collected until the Commit phase deadline, *requestDeadline* (line 11), has passed (line 31).

G.3 Reveal

A client who committed to trade through a $\langle \mathit{COMMIT}, \mathit{com}, S, \pi\rangle$ transaction in the Commit phase submits a Reveal transaction directly to the blockchain of the form $\langle \mathit{CLIENT\text{-}REVEAL},\ \mathit{tkn}, \mathit{size}, p, w,\ S, r, \mathit{regID}\ \mathit{regIDNew}\rangle$ (line 34). If the client intends to ren-enter the protocol as a client, *regIDNew* is a commitment to a new serial number and randomness. Otherwise, it is the null value.

This $\langle \mathit{CLIENT\text{-}REVEAL}, *\rangle$ transaction reveals the token being sold, token amount to sell, and requested width of the order. The p either reveals a limit price at which the client is selling, that the order is the market order if $p = \mathit{mkt}$, or that the client is withdrawing their escrow if $p = \mathit{withdraw}$. The contract checks:

- $S \in \mathit{clientCommits}$ to verify a commitment corresponding to that serial number has been recorded.

– $\mathit{regID} = h(S, r)$ to ensure that client was indeed the same player that generated the *regID* and that the client order is the same as that committed in the Commit phase $h(\mathit{tkn}, \mathit{size}, p, w) = \mathit{clientCommits}[S].\mathit{com}$.

If the transaction is valid, the order is added to *revealedBuyOrders* or *revealedSellOrders*, depending on direction. If the token being sold by the client is A_{tkn}, the effective order size for clearing price calculation and trade size allocation is the minimum of *size* and $\mathit{escrow}_{\mathit{client}}/y_A$, the maximum token A_{tkn} order size allowable (line 37), with the order recorded in *revealedOrders* (line 38). If the token being sold by the client is B_{tkn}, the order size is the minimum of *size* and $\mathit{escrow}_{\mathit{client}}/(y_A \cdot \mathit{offer})$ (line 45), with the order recorded in *revealedOrders* (line 46).

Finally, if *regIDNew* is the null value, the escrow is returned (line 41 or 48), while if it is not, it corresponds to re-entering the protocol as a new client (saving on an additional transaction to re-enter as a client).

A MM who committed to a market through a $\langle \mathit{COMMIT}, \mathit{com} \rangle$ transaction in the commit phase submits $\langle \mathit{MM\text{-}REVEAL}, \mathit{market} \leftarrow (\mathit{bid},\ \mathit{size}_{\mathit{bid}},\ \mathit{offer},\ \mathit{size}_{\mathit{offer}}) \rangle$ (line 56). The contract verifies:

– The market *market* matches the previously commitment from the commit phase ($h(\ \mathit{market}) = \mathit{MMCommits}[\mathit{MM}].\mathit{com}$) (line 56).
– For the bid, $Q_{\mathit{not}}/y_A \leq \mathit{size}_{\mathit{bid}}$, which verifies the MM has provided the minimum notional required by a WSFBA (line 57).
– For the offer, $Q_{\mathit{not}}/(y_A \cdot \mathit{offer}) \leq \mathit{size}_{\mathit{offer}}$, which again verifies the MM has provided the minimum notional required by a WSFBA (line 57).

Any MM not revealing a market (line 84) loses their escrow (line 85) and is prevented from participating in the Resolution phase. Otherwise, from the set of all valid revealed markets *revealedMkts*, the tightest market is selected, including a tie-breaking procedure for more than one market with width equal to the tightest width. The tie-breaker used in our implementation of FairTraDEX takes, for a MM *MM* (identified by a unique public key) and submitted market *market*, the market corresponding to the largest value of $h(h(\mathit{revealedMkts})\ ||\ \mathit{MM}\ ||\ \mathit{market})$ (lines 64-80). Given the tightest market $\mathit{market} \leftarrow (\mathit{bid},\ \mathit{size}_{\mathit{bid}},\ \mathit{offer},\ \mathit{size}_{\mathit{offer}})$ after tie-breaks, the two implicit limit orders are added to the set of revealed orders *revealedBuyOrders* and *revealedSellOrders*. As was the case with client orders, the effective order size for clearing price calculation and trade size allocation of the bid is the minimum of $\mathit{size}_{\mathit{bid}}$ and $\mathit{escrow}_{\mathit{MM}}/y_A$, while the effective offer size is the minimum of $\mathit{size}_{\mathit{offer}}$ and $\mathit{escrow}_{\mathit{MM}}/(y_A \cdot \mathit{offer})$.

Reveal transactions are collected until the Reveal deadline, *revealDeadline* (line 11), has passed (line 31).

G.4 Resolution

Once the protocol enters the Resolution phase, any player in the system can propose a clearing price by submitting a $\langle CP, * \rangle$ message. Players submitting such a message must deposit a token amount (which we set as *resBounty*, although any significantly large value to prevent invalid calls to the smart contract will do). This deposit, along with a bounty is returned to the player if *CP* is a valid clearing price.

The orders are then settled based on the clearing price *CP* (lines 114-134). If the quantity of A_{tkn} being sold is greater than the quantity being bought, the sell orders at the highest sell price below the clearing price are pro-rated based on the quantity of A_{tkn} being sold. If the quantity of token A_{tkn} being bought is greater than the quantity being sold, the buy orders at the lowest buy price above the clearing price are pro-rated based on the quantity of A_{tkn} being bought. Remaining unexecuted order balances and escrows are returned to the owners.

G.5 Protocol Encoding

In the following, for arrays containing array objects, the array objects are uniquely identifiable by the first item in the array (i.e. client identifier, serial number, ZKProof).

Algorithm 1 Register

```
1: $players \leftarrow generatePopulation()$
2: $RegIDs \leftarrow []$
3: $clientCommits \leftarrow []$
4: $MMCommits \leftarrow []$
5: $phase \leftarrow$
6: $w_{tight} \leftarrow any$
7: $revealedBuyOrders,\ revealedSellOrders, revealedMkts \leftarrow []$
8: $lastPhaseChange \leftarrow 0$    ▷ tracks the block number of last step update
9: $f_r \leftarrow getRelayerFee()$
10: $minTickSize \leftarrow getMinimumTickSize()$
11: $requestDeadline, revealDeadline \leftarrow T$    ▷ Deadline for responses equal to the maximal reveal delay described in the Threat Model
12: $escrow_{client} \leftarrow Y$    ▷ Escrow required to show each market, in line with the Threat Model
13: $Q_{not} \leftarrow getMaxAuctionNotional()$
14: $c \leftarrow random(\mathbb{R}_{>1})$
15: $escrow_{MM} \leftarrow c \cdot Q_{not}$    ▷ Escrow required to show each market, some amount greater than $Q_{not}$
16: $y_A \leftarrow getTokenAIndicativePrice()$
17: $currAucNotional \leftarrow 0$
18: $blacklistedSNs \leftarrow []$    ▷ Tracks revealed serial numbers that misbehaved

19: function Initialise()
20:     $phase \leftarrow Commit$

21: upon $\langle CLIENT\text{-}REGISTER, regID\rangle$ from $player \in players$
    with $player.balance(₿) > escrow_{client} + f_r$ do    ▷ register player as a client
22:     $player.transfer(escrow_{client} + f_r, ₿, protocolContract)$    ▷ Add client deposit to the contract account
23:     $clients.append(regID)$
```

Algorithm 2 Commit

```
24: upon $relay(\langle COMMIT, com, S, \pi \rangle)$ from $player \in players$ with
    $currAucNotional < Q_{not} \wedge$ Verify$(\pi, com) = 1 \wedge phase = Commit \wedge \neg(S \in blacklistedSNs)$
    do
25:     $currAucNotional \leftarrow currAucNotional + escrow_{client}$
26:     $clientCommits.append([S, com])$
27:     $protocolContract.transfer(f_r, ₿, player))$        ▷ Reward relayer

28: upon $\langle COMMIT, com \rangle$ from $player \in players$ with $player.balance(₿) > escrow_{MM} \wedge$
    $\neg(player \in MMCommits)$ $phase = Commit$ do        ▷ Allow only one market per player address
29:     $player.transfer(escrow_{MM}, ₿, protocolContract)$   ▷ Transfer escrow to the protocol contract
    account
30:     $MMCommits.append([player, com])$

31: upon $phase = Commit \wedge Blockchain.height() = lastPhaseChange + requestDeadline$ do
32:     $phase \leftarrow Reveal$
33:     $lastPhaseChange \leftarrow Blockchain.height()$
```

Algorithm 3 Reveal

```
34: upon ⟨$CLIENT\text{-}REVEAL,\ tkn, size, p, w,\ S, r, regID\ , regIDNew$⟩ from $player \in players$
    with $S \in clientCommits\ \wedge\ regID = h(S, r)\ \wedge\ h(tkn, size, p = clientCommits[S].com$
    $\wedge\ phase = Reveal$ do
35:     if $p \neq withdraw$ then
36:         if $tkn = A_{tkn}\ \wedge\ player.balance(A_{tkn}) \geq size$ then
37:             $size \leftarrow minimum(size, escrow_{client}/y_A)$
38:             $revealedBuyOrders.append([player,\ size, p, w])$ ▷ Add client order to array of orders
    to trade
39:             $player.transfer(size, A_{tkn}, protocolContract)$
40:             if $regIDNew = \varnothing$ then
41:                 $protocolContract.transfer(escrow_{client}, ₿,\ player))$
42:             else
43:                 $clients.append(regIDNew)$
44:         else if $tkn = B_{tkn}\ \wedge\ player..balance(B_{tkn}) \geq size$ then
45:             $size \leftarrow minimum(size, escrow_{client}/(y_A \cdot p))$
46:             $revealedSellOrders.append([player,\ size, p, w])$ ▷ Add client order to array of orders
    to trade
47:             if $regIDNew = \varnothing$ then
48:                 $protocolContract.transfer(escrow_{client}, ₿,\ player))$
49:             else if $player.balance(₿) > f_r$ then
50:                 $player.transfer(f_r, ₿, protocolContract)$
51:                 $clients.append(regIDNew)$
52:     else                                                        ▷ Client wants to withdraw
53:         $protocolContract.transfer(escrow_{client}, ₿,\ player))$
54:     $clients.remove(regID)$
55:     $clientCommits.remove(S)$

56: upon ⟨$MM\text{-}REVEAL, market \leftarrow (bid,\ size_{bid},\ offer,\ size_{offer})$⟩ from $MM \in$
    $MMCommits$ with $h(market) = MMCommits[MM].com \wedge phase = Reveal$ do
57:     if $(Q_{not}/(y_A \cdot offer) \leq size_{offer} \leq player.balance(B_{tkn}))$
    $\wedge\ (Q_{not}/y_A \leq size_{bid} \leq player..balance(A_{tkn}))$ then ▷ Check MM has provided the minimum
    required liquidity, $Q_{not}$
58:         $revealedMkts.append(MM, market)$
59:         $MMCommits.remove(MM)$

60: upon $phase = Reveal\ \wedge\ len(MMCommits) = 0\ \wedge\ len(clientCommits) = 0$ do ▷ All reveals
    published
61:     $phase \leftarrow Resolution$
62:     $lastPhaseChange \leftarrow Blockchain.height()$

63: upon $phase = Reveal\ \wedge\ Blockchain.height() = lastPhaseChange + revealDeadline$ do
64:     $tieBreaker \leftarrow 0$
65:     $tightMkt \leftarrow ()$
66:     $tieBreakSeed \leftarrow h(revealedMkts)$ ▷ Generate seed for tie-breaks before revealed markets is
    changed
67:     for $MM \in revealedMkts$ do ▷ Select the unique market corresponding to the tie-breaker in
    the proceeding If statement
68:         if $(Q_{not}/(y_A \cdot MM.offer) \leq MM.size_{offer} \leq MM.balance(B_{tkn}))$
    $\wedge\ (Q_{not}/y_A \leq MM.size_{bid} \leq MM.balance(A_{tkn}))$ then ▷ Check MM *still* has provided the
    minimum required liquidity
69:             $protocolContract.transfer(escrow_{MM}, ₿,\ MM))$
70:             if $w_{tight} = any\ \vee\ (\frac{offer}{bid} < w_{tight})$
    $\vee\ (\frac{offer}{bid} = w_{tight} \wedge h(tieBreakSeed||MM||MM.market) > tieBreaker)$ then
71:                 $w_{tight} \leftarrow \frac{offer}{bid}$
72:                 $tieBreaker \leftarrow h(tieBreakSeed||MM||MM.market)$
73:                 $tightMkt \leftarrow [MM, market]$
74:         $revealedMkts.remove(MM)$
75:     $size_{bid} \leftarrow minimum(tightMkt.size_{bid}, escrow_{MM}/y_A)$
76:     $size_{offer} \leftarrow minimum(tightMkt.size_{offer}, escrow_{MM}/(y_A \cdot tightMkt.offer))$
77:     $revealedBuyOrders.append([player \leftarrow tightMkt.MM,$
    $size \leftarrow size_{bid}, p \leftarrow tightMkt.bid, w \leftarrow any])$ ▷ Add tightest market to set of orders to be settled
78:     $revealedSellOrders.append([player \leftarrow tightMkt.MM,$
    $size \leftarrow size_{offer}, p \leftarrow tightMkt.offer, w \leftarrow any])$
79:     $tightMkt.MM.transfer(size_{bid}, A_{tkn}, protocolContract)$
80:     $tightMkt.MM.transfer(size_{offer}, B_{tkn}, protocolContract)$
81:     for $S \in clientCommits$ do ▷ Add all clients who did not reveal order to blacklist,
    preventing further commitments
82:         $blacklistedSNs.append(S)$
83:         $clientCommits.remove(S)$
84:     for $MM \in MMCommits$ do                        ▷ MMs who did not reveal market in time
85:         $MMCommits.remove(MM)$ ▷ Remove from protocol without adding to $revealedOrders$,
    effectively burning escrow
86:     $phase \leftarrow Resolution$
87:     $lastPhaseChange \leftarrow Blockchain.height()$
```

Algorithm 4 Resolution: Clearing Price Verification

```
88: upon $\langle CP, volumeSettled, imbalance\rangle$ from $player \in players$ with
    $player.balance(₿) > resBounty \land phase = Resolution$ do
89:     $player.transfer(resBounty, ₿, protocolContract)$   ▷ To prevent Sybil attacks, player must deposit funds which are returned if $CP$ is valid
90:     $revealedSellOrders \leftarrow revealedSellOrders[revealedSellOrders..width() > w_{tight} \lor revealedSellOrders.width() = any]$   ▷ Remove sell orders that cannot trade due to requested width
91:     $revealedBuyOrders \leftarrow revealedBuyOrders[revealedBuyOrders..width() > w_{tight} \lor revealedBuyOrders.width() = any]$
92:     Assert($volumeSettled > 0 \lor$ minimum$(revealedSellOrders.p) <$ maximum$(revealedBuyOrders.p)$)   ▷ If the indicated clearing price is below the lowest offer/above highest bid, all of the proceeding checks pass.
93:     $buyVolume \leftarrow sum(revealedBuyOrders[revealedBuyOrders.p \geq CP].size)$
94:     $sellVolume \leftarrow sum(revealedSellOrders[revealedSellOrders.p \leq CP].size)$
95:     Assert(minimum$(buyVolume/CP, sellVolume) = volumeSettled$))
96:     Assert($(buyVolume/CP) - sellVolume = imbalance$ )
97:     if $imbalance = 0$ then   ▷ We are done
98:         $SettleOrders(CP, buyVolume, sellVolume)$
99:     if $imbalance > 0$ then   ▷ As the auction is bid at $CP$, check if next price increment above clears higher volume OR smaller imbalance
100:        $priceToCheck \leftarrow CP + minTickSize$
101:        $buyVolumeNew \leftarrow (buyVolume - sum(revealedBuyOrders[CP \leq revealedBuyOrders.p < priceToCheck].size))/CP$
102:        $sellVolumeNew \leftarrow sellVolume + sum(revealedSellOrders[CP < revealedSellOrders.p \leq priceToCheck].size)$
103:        Assert((minimum$(buyVolumeNew, sellVolumeNew) < volumeSettled) \lor$
            (minimum$(buyVolumeNew, sellVolumeNew) = volumeSettled \land$
            $imbalance \leq |buyVolumeNew - sellVolumeNew|$))   ▷ If the next price clears less volume, or clears the same volume with a larger imbalance, the proposed CP is valid
104:        $SettleOrders(CP, buyVolume, sellVolume)$
105:    if $imbalance < 0$ then   ▷ As the auction is offered at $CP$, check if next price increment below clears higher volume OR smaller imbalance
106:        $priceToCheck \leftarrow CP - minTickSize$
107:        $buyVolumeNew \leftarrow (buyVolume + sum(revealedBuyOrders[CP > revealedBuyOrders.p \geq priceToCheck].size))/CP$
108:        $sellVolumeNew \leftarrow sellVolume - sum(revealedSellOrders[CP \geq revealedSellOrders.p > priceToCheck].size)$
109:        Assert((minimum$(buyVolumeNew, sellVolumeNew) < volumeSettled) \lor$
            (minimum$(buyVolumeNew, sellVolumeNew) = volumeSettled$
            $\land\ imbalance \leq |buyVolumeNew - sellVolumeNew|$))
110:        $SettleOrders(CP, buyVolume, sellVolume)$
111:    $protocolContract.transfer(2resBounty, ₿, player)$)   ▷ Return deposit, and reward player for submitting valid clearing price
```

Algorithm 5 Resolution: Settle Orders

```
112: function SettleOrders($CP$, $buyVolume$, $sellVolume$)
113:     $buyVolume \leftarrow buyVolume / CP$    ▷ Convert sell volume to equivalent in $A_{tkn}$
114:     if $buyVolume > sellVolume$ then ▷ pro-rate buy orders at the min price above (or equal to) the clearing price
115:         $p_{pro\text{-}rate} \leftarrow minimum(revealedBuyOrders[revealedBuyOrders.p \geq CP].p)$
116:         $size_{pro\text{-}rate} \leftarrow sum(revealedBuyOrders[revealedBuyOrders.p = p_{pro\text{-}rate}].size)/CP$
117:         for $order \in revealedBuyOrders[revealedBuyOrders.p = p_{pro\text{-}rate}]$ do
118:             $protocolContract.transfer(order.size \cdot (1 - \frac{buyVolume - sellVolume}{size_{pro\text{-}rate}}), A_{tkn}, order.player)$ ▷ return tokens not going to be exchanged
119:             $order.size \leftarrow order.size \cdot \frac{buyVolume - sellVolume}{size_{pro\text{-}rate}}$
120:     else if $sellVolume > buyVolume$ then ▷ pro-rate sell orders at the max price below (or equal to) the clearing price
121:         $p_{pro\text{-}rate} \leftarrow maximum(revealedSellOrders[revealedSellOrders.p \leq CP].p)$
122:         $size_{pro\text{-}rate} \leftarrow sum(revealedSellOrders[revealedSellOrders.p = p_{pro\text{-}rate}].size)$
123:         for $order \in revealedSellOrders[revealedSellOrders.p = p_{pro\text{-}rate}]$ do
124:             $protocolContract.transfer(order.size \cdot (1 - \frac{sellVolume - buyVolume}{size_{pro\text{-}rate}}), B_{tkn}, order.player)$ ▷ return tokens not going to be exchanged
125:             $order.size \leftarrow order.size \cdot \frac{sellVolume - buyVolume}{size_{pro\text{-}rate}}$

126:     for $order \in revealedBuyOrders, revealedSellOrders$ do    ▷ iterate through orders
127:         if $order \in revealedBuyOrders \wedge (order.p \geq CP \vee order.p = mkt)$ then    ▷ execute buy order if bid greater than clearing price
128:             $tokenTradeSize \leftarrow order.size/CP$
129:             $protocolContract.transfer(tokenTradeSize, B_{tkn}, order.player)$
130:         else if $order \in revealedSellOrders \wedge (order.p \leq CP \vee order.p = mkt)$ then    ▷ execute sell order if bid greater than clearing price
131:             $tokenTradeSize \leftarrow (order.size)/CP$
132:             $protocolContract.transfer(tokenTradeSize, A_{tkn}, order.player)$
133:         else    ▷ Order not executed
134:             $protocolContract.transfer(order.size, order.tkn, order.player)$

135:     $phase \leftarrow Commit$
136:     $currAucNotional \leftarrow 0$
137:     $revealedBuyOrders, revealedSellOrders \leftarrow []$
138:     $w_{tight} \leftarrow any$
139:     $lastPhaseChange \leftarrow Blockchain.height()$
```

G.6 Existence of irrational players and coalitions

When analysing the optimal strategies of rational players in WSFBAs, our results are based on all players being rational and that n_ψ instances of Register() are called. If we consider the presence of irrational players in the system, we can apply the following adjustments:

– **Irrational MM**: In Lemma 1, it is shown that the optimal strategy for a MM is to show markets centred around the MIFP. Any other (irrational) strategy must therefore result in reduced expectancy for the MM, and higher expectancy for clients. Therefore, given the presence of irrational MMs, submitting market orders maximises client expectancy (with greater expected utility than in the presence of rational MMs, although with increased variance).
– **Irrational client**: Given the optimal strategy for rational clients is to submit market orders, a irrational client may then submit limit orders. This

merely reduces the irrational client's chance of trading vs. other clients. This would not change the strategy of non-colluding rational MMs, but may have some affect on a monopolistic MM's interpretation of f_{mcf}.

Furthermore, if less than n_ψ instances of Register() are called, clients resort to submitting limit orders. This can be seen in the proof of Theorem 1. In the proof, if clients are not sure that a MM will show a market with reference price equal to the MIFP, the case when less than n_ψ instances of Register() are called, the optimal strategy for clients is to submit limit orders, which only stands to reduce the clients' probability of trading. As the number of non-cooperative players in FairTraDEX decreases towards two, the guarantees of FairTraDEX approach those of an AMM. However, as client price and order size remain hidden until the counterparty chooses her strategy, and before the clearing price is fixed (end of the Commit phase), FairTraDEX maintains advantages over AMMs against client-based EEV attacks, such as price/order-size specific front-running and selective participation.

H Practical Considerations for FairTraDEX

Practical approaches to ensure equivalent to n_ψ Register() calls are taken in existing anonymity protocols. For example, Tornado Cash [14] rewards players proportionally to the (Tornado Cash equivalent of the) number of Register() calls, as well as the length of time between calling Register() and CommitClient(). ,

Escrow choices

Choosing escrow amounts for clients and MMs should reflect the emergent use cases of the protocol. It is possible to create different FairTraDEX instances for the same pair of tokens based on trade size, both for liquidity purposes (MMs will require wider markets for larger-sized orders, but the corresponding increased escrow requirements might prevent smaller clients from participating) and auction use-cases (day-trading vs. end-of-day balancing). Furthermore, as the escrow denomination (₿) in our description is different to at least one of the tokens, there needs to be some way to translate the escrow amount into order sizes. This will depend on the environment, but on Ethereum for example, price oracles (existing AMMs, Chainlink[15], etc.) can be used. It is also possible to use previous clearing prices from within the FairTraDEX ecosystem, although a self-referential oracle must be implemented carefully as there may be game-theoretic implications. If a satisfactory price oracle exists, deposits can be made in the respective tokens of the swap, further reducing the capital requirements for players and encouraging adoption.

[14] `https://torn.community/t/anonymity-mining-technical-overview/15` Accessed: 20/07/2022

[15] https://chain.link/

Incentive compatibility given transaction fees

In Section 5, we mention that our Nash equilibria are dependent on the utility gained by clients and MMs being greater than the cost for participation. The choice of smart-contract enabled blockchain on which to deploy will dictate the barrier of entry for clients and MMs alike. Like existing attempts to implement blockchain-based FBAs [20,14], we have an amortised number of transactions per player of two. A naive comparison to AMMs, where this is reduced to 1 for clients, and 0 for MMs, certainly has less direct costs than FairTraDEX. However, when factors like impermanent loss, slippage[16], front-running, and EEV attacks in general, the value being extracted from DEXs incurs a significant indirect cost for clients. Immediately, we can increase the expected cost of using AMMs by the slippage required by AMMs (set to 0.5% as of writing in Uniswap V3, but for larger orders this must increase by the nature of AMMs). We can increase this further by the probability orders are not executed (where prices move more than the slippage, potentially due to front-running) but are added to the blockchain. As such, the indirect costs are substantial, are increasing in order-size increases and proportionally to improvements in client trading ability as strategies can be replicated/front-run. A thorough comparison of the monetary costs of FairTraDEX vs. AMMs over various order-sizes, and trading scenarios makes for interesting future work as FairTraDEX begins to be deployed and tested in the wild.

It can be seen from the proof of Theorem 1 that the client strategies identified are strong incentive compatible in expectation as the MM always shows markets of width less than or equal to f_{mcf}. However, the MM strategy of showing width 1 markets is not strong incentive compatible. In addition to the fees described in the protocol, an additional fee can be applied within the protocol itself to incentivise the participation of MMs. This can be a function of MM participation/market widths. As with all additional fees/rewards, the game-theoretic implications of such an incentivisation scheme must be considered.

In our encoding of FairTraDEX, we do not explicitly introduce a cost for clients and MMs in Commit/Reveal phases to reward the submission of the clearing price. In reality, the result in Lemma 2 holds without the introduction of an explicit reward, as there all participating clients and MMs will have positive expectancy to receive tokens through correct order resolution. The use of an explicit reward is for illustrative purposes, and to avoid complications regarding transaction fees for running the clearing price checks. The costs of running the Resolution contracts must be ensured to be less than the utility gained by at least one player in the blockchain protocol for calling the contract.

Differences to previous versions of FairTraDEX

In a previous version of this paper, the notional amount Q_{not} was was also used to upper-bound the notional of client orders allowable in an auction. This was

[16] `https://docs.uniswap.org/protocol/concepts/V3-overview/swaps`

intended to simplify the strategy analysis in the proof of Theorem 1, although without adding any additional guarantees to the protocol. In the current version of the paper, we manage to provide a simpler definition of a WSFBA without significantly adding to the complexity of Theorem 1 by noting any MM order through the MIFP must be negative in expectancy for the MM.