An Exchange Protocol for the Decentralized Web

ABSTRACT

Blockchains and specifically smart contracts are a promising tool to enable a secure and fair trading ecosystem. The first exchanges built on these technologies emulate the traditional continuous double auction design. Existing solutions that do not rely on a centralized operator face significant challenges with front-running.

This paper proposes a new trading concept that uses discrete double auctions and time lapse order encryption. The protocol eliminates front-running, provides information symmetry for all participants and is fully decentralized in the sense that there is no central party operating the system.

KEYWORDS

Decentralized Exchange, Front-running, Distributed Key Generation, Auction Mechanism, Ring Trades

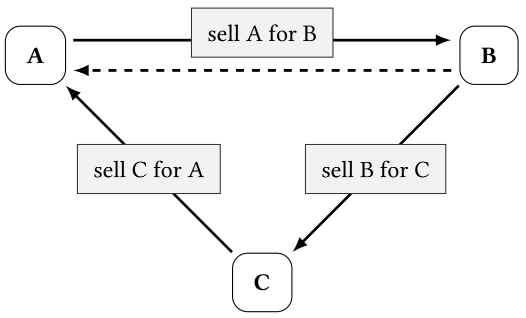

Figure 1: Ring trade between 3 tokens. Without an order selling token B for A, the order selling A for B can still be matched with the two orders selling token B for C and C for A (provided complying limit prices).

1 INTRODUCTION

Exchanges are a fundamental component of any financial system. In order to prevent fraud and keep these platforms fair, they are heavily regulated [23]. Such regulations impose a high barrier to entry, thus limiting innovation and competition within the space [20].

Blockchains that perform transactions and computation without trusted intermediaries are a promising technology to build a new, decentralized financial ecosystem with fewer regulatory requirements. Decentralized exchanges (DEXs) built on blockchains consist of two major components: a matching engine and a settlement layer. Computational blockchains allow advanced asset management via *smart contracts* (pre-programmed logic enforced by the blockchain), making them particularly useful as a settlement layer. *Non-custodial* exchanges no longer require transfer of asset ownership. Instead, traders issue an allowance to the DEX smart contract to exchange one asset for another at a specified price. Unlike traditional cryptocurrency exchanges, some of which have suffered multimillion dollar hacks [24], non-custodial exchanges significantly decrease the default risk of the deposited assets, since the exchange never has custody over the funds.

Nearly all DEXs today implement non-custodial settlement layers (cf. Section 2). However, most of them lack a decentralized matching engine. This makes them prone to front-running by the operator and leads to information asymmetry between the matching engine operator and the trader. For customers this means that the operator can both reorder thus front-run transactions and censor certain orders.

Another characteristic of blockchains is that they significantly lower the cost of generating new *tokens* (tradable assets) [13]. Unlike traditional markets, where most liquidity is available in a dominant asset (e.g. USD), blockchain markets suffer from a highly fragmented liquidity space among a wide range of low liquidity assets [29]. Examples of such are tokens created from initial coin offerings, prediction market outcome tokens, tokenized game assets or financial derivatives. *Stable coins*, which peg their value to a real-world currency, exemplify this fragmentation. Ethereum, as the leading computational blockchain, today trades eight different USD stable coins with more than $1 million in market capitalization each according to the *stable coin index*[1].

Having a fragmented token space with pairwise auctions leads to thin order books and high slippage. In such markets, arbitrage traders provide liquidity across different markets and, in return, make profit from the price difference of overlapping order books. In traditional markets this profit comes with the risk of inability to fulfill both trades atomically.

Blockchains by design collate multiple transactions into discrete blocks. This discreteness allows to execute a series of arbitrage trades atomically, leading to risk-free arbitrage at the expense of regular traders in many DEX's designs. Recent work shows that the lower bound for such risk-free arbitrage on Ethereum alone amounts to $1.6 million per year [8].

This work proposes a new trading protocol that addresses both front-running as well as fragmented liquidity, while not relying on the integrity of a centralized operator. This is achieved by combining *sealed-bid auctions* with *uniform clearing price multi-asset batch auctions*.

Traders submit orders between any token pair in a sealed-bid auction over a specified amount of time. The resulting order book is viewed as a directed multi-edge graph whose nodes represent tokens and edges represent orders. The matching engine then allows participants to propose solutions consisting of a price for every token along with execution information for each order in this graph. Thereby, all orders on the same token pair either remain untouched or are executed at the same exchange rate ("uniform clearing prices"). Trades can be executed along cycles of tokens (cf. Figure 1). This mechanism is especially powerful in an ecosystem where there are few dominant trading pairs. The engine verifies and compares all proposed solutions by cumulative *trading surplus*. It

[1]https://stablecoinindex.com/volume

rewards and executes the best submitted solution while punishing those which are invalid.

The proposed exchange protocol enables the following feature set:

Full Decentralization. Both order matching and asset management do not require a centralized party to steer the operations. Trades are executed in the best available way with respect to a globally desirable metric (trading surplus). The matching engine outsources price determination to its decentralized peers and accepts submissions from anyone in the network. Hence, liveness is achieved provided one peer is willing to facilitate the price calculation.

Front-Running Resilience. Discrete double auctions (referred to equivalently as batch auctions) are less vulnerable to front-running attacks compared to continuous double auctions, as the exact sequence of the orders does not matter for execution. Additionally, sealed bidding ensures that no meaningful front-running attacks are possible and information symmetry between all participants is guaranteed.

Pooled Liquidity. The protocol pools fragmented liquidity over multiple assets into the same auction and allows atomic transitive order matching via ring trades. Hence it is well suited for a multi-asset market with no dominating currency. Most importantly, due to uniform clearing prices, the increase in liquidity does not come at the cost of arbitrage for regular traders. Instead, the difference between limit and market price flows back to the traders.

Scalability. The mechanism is designed specifically for ledgers that support arbitrary computation but are constrained on the amount of computation that can be performed in each block. The protocol is built as a second layer on top of existing blockchains, allowing it to process a multiple of the transaction throughput that current blockchains offer. Complex computations such as finding an optimal order matching is done completely off-chain, while only the computationally less expensive verification of proposed solutions takes place on-chain.

Censorship Resistance. Trading is guaranteed and can not be stopped or censored by any operator. Bids are collected on-chain and are visible to the entire network. While individual solution proposers can choose to ignore specific orders, their solutions are likely to be inferior to the ones submitted by honest, non-censoring participants.

Considering that this protocol has no information asymmetry and does not require centralized operators, it has the potential to become an open and trust-minimized platform for international trades.

The remainder of this paper is organized as follows: Section 2 covers related work in the field. Section 3 explains the proposed auction mechanism in detail. Section 4 specifies the complete trading protocol of the exchange and Section 5 demonstrates how it can be implemented on the Ethereum blockchain. Section 6 discusses trade-offs of the design and concludes the paper.

2 RELATED WORK

Related work for this protocol falls into three categories: existing DEXs, auction mechanism design, and research on scaling computational blockchains.

DEXs. As outlined in Section 1, exchanges usually have two components: a matching engine and a settlement layer. Early versions of DEXs, such as Etherdelta, implement an on-chain matching [17]. They allow makers to place a *bid-order* onto the blockchain, which can be settled by order takers. Takers are often competing for the same bid-order setting up a racing game between them, which has severe front-running implications [8].

Leading exchanges today, such as IDEX and 0x, implement an approach, where a central party is responsible for collecting and matching signed orders off-chain. Only matched orders are submitted for execution on-chain, which significantly improves throughput and lowers the cost per order. However, it shifts the racing game towards the central operator, who is able to determine the sequence of order settlements and hence can front-run traders.

TEX is a decentralized exchange protocol that also matches orders off-chain, but prevents sequence alteration with a "time-lock puzzle" [19]. Participants hide their orders using a verifiable delay function [4] and require operators to commit to their order at a certain position. The timeout for such a commitment is shorter than the lower bound of the delay function and thus happens before order information is revealed. Unfortunately, the method relies on expensive computation which grows linearly with the number of orders.

The Loopring protocol addresses the issue of siloed liquidity by allowing so-called ring-miners to find and atomically execute trades along cycles of up to ten bids across different token pairs [34]. Each order specifies the percentage of its limit price margin (surplus) which is paid to the ring-miner that includes it in a trade. Loopring prevents front-running of mined rings by augmenting orders with two levels of authorization – one for settlement, and one for ring-mining. Only the ring miner has access to the latter and can thus sign rings and immutably claim the rewards. In this protocol, however, liquidity is still split between multiple relay nodes. Moreover, Loopring does not address front-running of orders or cancellations and only matches single trades rather than batches with uniform clearing prices.

Auction mechanisms. Sealed-bid auctions are commonly used to prevent front-running from other auction participants [28], e.g., in the Ethereum Name Service [12]. However, these auctions either imply a "free-option" for the bidder to not reveal their bid or rely on trust-assumptions towards the auctioneer, who gains sensitive information from the participants bids. Schemes that do not require trust in the auctioneer by allowing bidders to jointly compute the highest bid without revealing information have been proposed [5]. For example, Juang et al use distributed key generation (DKG) to design a privacy preserving sealed-bid auction [18].

Another trading mechanism less prone to front-running by design than continuous auctions is the *discrete double auction.* In particular, Budish et al [7] have described the benefits of frequent batch auctions with respect to reducing arbitrage opportunities and increasing liquidity, whereas the aspect of uniform clearing prices has been discussed by ENGELBRECHT-WIGGANS and KAHN [11].

Discrete double auctions, where the prices are calculated via a complex optimization problem, have often been dismissed as most of them are NP-hard. Yet, in a multi-asset setup, these auctions result in better prices compared to auctions that solve each asset pair separately [9]. The fact that these optimization problems are hard to solve has been worked around by decentralizing the solution process and opening it up to anyone. BREWER [6] and WELLMAN ET AL [35] propose protocols in which an operator publishes their best solution to the optimization problem and anyone can improve the solution by submitting a better one. A very similar auction mechanism to the one employed for our protocol, together with a price-finding algorithm via mixed-integer linear programming, is presented in [33].

Scaling. Blockchains underlie a trilemma stating that no ledger can achieve more than two of the three properties: correctness, decentralization, scalability [1]. Therefore, secure blockchains are inherently slow and hard to scale. Ethereum, as an example, currently executes around 15 transactions per second [37]. The most popular DEX on Ethereum, IDEX, executed 198,000 trades during March 2019, roughly one every 13 seconds. Ethereums transaction processing speed is expected to scale by a factor of 1000 with the introduction of its Serenity hard-fork [14].

Second layer scaling solutions, such as State Channels [10], furthermore increase the network's throughput for asset transfer. However, their computations are restricted to a small set of predefined participants.

Plasma [27] is a second layer solution that does not restrict the participants and anchors the root of its current state on the main chain. However, most implementations suffer from data availability issues and complex exit games in the event that the operator decides to withhold information.

Another second layer scaling solution called Rollup solves the data availability problem by broadcasting all data as part of the transaction and only runs a succinct proof verification of the smart contract logic on-chain [36]. The proof system requires reformulating the program code into an arithmetic circuit [16]. While cost of verification of such proofs is constant with regards to the size of the underlying computation, off-chain proof generation is time-consuming and computationally expensive.

The proof system used in Rollup is instantiated via a complex multi-party computation that requires at least one honest party. STARK [2] is a comparable proof system that does not require a trusted setup, is post-quantum resistant and currently under development for use in a decentralized exchange called StarkDex [21]. However, STARK proofs are large in size and thus expensive to verify on-chain.

TrueBit allows verifying large amounts of logic on-chain by splitting the computation into a set of small intermediate steps and running an interactive binary search between the solution provider and challenger to find the single step in which their results disagree [31]. The smart contract is then used to resolve only the disputed step. However, the interactivity of the protocol requires validators to stay online for a long time and the number of transactions required grows with the underlying size of the computation.

3 ORDER MATCHING MECHANISM

One of the key innovations of this exchange protocol is the underlying order matching mechanism referred to here as *multi-token batch auction with uniform clearing prices.* On a high level, the mechanism intends to provide fair and consistent prices for all market participants while maximizing the overall liquidity that is available. It relies on a specific batch trading approach where orders for a set of multiple token pairs are first collected during the course of discrete time intervals and matched simultaneously afterwards ("multi-token batch auctions"). Thereby, all orders on the same token are executed at the same exchange rate ("uniform clearing prices"). This enables so-called *ring trades*, where orders can be executed not only between token pairs, but along arbitrary cycles of tokens as illustrated in Figure 1. The following section describes the order matching mechanism in detail, closely following the notation in [33].

3.1 Problem statement

Data. Let $\mathcal{T} = \{\tau_1 \dots \tau_n\}$ denote the set of the n tokens that are being considered. Additionally, we introduce an artificial token τ_0 that shall be referred to as *reference token*, which will be of important use as a common numeraire in the definition of optimization criteria later. Pairwise exchange rates between two tokens τ_j and τ_k will be denoted by $p_{j|k}$, meaning the price of one unit of τ_j measured in units of τ_k. As an example, at an exchange rate of $p_{j|k} = 10$, one would need to pay an amount of 10 units of τ_k in order to purchase one unit of τ_j. Moreover, let there be a set $O = \{\omega_1 \dots \omega_N\}$ of N orders in the batch to be processed. Each order ω_i is specified as a tuple $\omega_i := (j, k, \overline{x}, \overline{y}, \pi)_i$ of parameters that are defined as follows:

$j \in \{1 \dots n\}$	...	τ_j is the token to be bought,
$k \in \{1 \dots n\}$	...	τ_k is the token to be sold,
$\overline{x} \in \mathbb{R}_{\geq 0} \cup \{+\infty\}$	...	maximum amount of τ_j to be bought,
$\overline{y} \in \mathbb{R}_{\geq 0} \cup \{+\infty\}$	...	maximum amount of τ_k to be sold,
$\pi \in \mathbb{R}_{>0} \cup \{+\infty\}$	...	limit exchange rate for order execution, i.e., $p_{j|k} \leq \pi$.

Using this universal notation, we can express a variety of different order types. As an example, a standard *limit sell order* can be stated as $(j, k, +\infty, \overline{y}, \pi)$ with $\overline{y} < +\infty$ and $\pi < +\infty$, which reads

"Sell (at most) $\overline{y}$ units of token τ_k for τ_j if $p_{k|j}$ is at least $1/\pi$".

The market participant sets a fixed maximum sell amount $\overline{y}$ and a limit exchange rate π, but does not have a limit on the amount of tokens τ_j to be obtained. The higher the exchange rate $p_{k|j}$, the more tokens τ_j the trader receives. Similarly, we can also specify *limit buy orders* as well as *double-sided limit orders* and *market orders.* For simplicity, however, only limit sell orders are considered in this paper.

Objectives. Our goal is to determine all exchange rates $p_{j|k}$ between token pairs (τ_j, τ_k) together with a feasible order matching such that a certain metric is maximized. Therein, for every limit sell order $\omega = (j, k, +\infty, \overline{y}, \pi)$, the order matching information shall be given as two values $y \in [0, \overline{y}]$ and $x \geq 0$, stating the amounts of the tokens τ_k and τ_j that are sold and bought, respectively. An order can be executed (fully or fractionally) only if its limit price is satisfied by the computed exchange rates, or must be left untouched otherwise. A very intuitive optimization criterion would be the maximization of the *total trading volume* across all orders, i.e.,

$$\max \sum_{i=1}^{N} v_i.$$

Therein, for a single limit sell order, the trading volume $v \in \mathbb{R}_{\geq 0}$ is defined as

$$v := y \cdot p_{k|0},$$

i.e., the amount of tokens τ_k sold multiplied by the price of τ_k with respect to the reference token τ_0. It is important to note that the reference token (even though itself not directly participating in the auction) serves as a common numeraire for the trading volumes of all orders. With this, trading amounts across different tokens can be aggregated in a single metric. See Observation 1 for another insight on the usefulness of the reference token. Alternatively, another possible optimization metric could be the maximization of the *total trading surplus* across all orders,

$$\max \sum_{i=1}^{N} w_i.$$

Here, the trading surplus for a single order is defined as

$$w := (x - y/\pi) \cdot p_{j|0}.$$

In words, for a limit sell order $\omega = (j, k, +\infty, \overline{y}, \pi)$ it considers the amount x of token τ_j received as compared to the minimum amount y/π that the trader would have accepted for selling y tokens τ_k. This difference, again, is weighted by the token price $p_{j|0}$ denoted in units of the reference token for a common scale of the surplus values. Note that the trading surplus will always be nonnegative. It can take positive values if the limit price of the order is satisfied, otherwise the traded amounts x and y must equal zero. The key advantage of the trading surplus over the trading volume as an optimization metric is that it yields economically efficient results. This means, in particular, that the execution of orders on every token pair is implicitly sorted by limit price, whereas trading volume maximization is only concerned whether an order may be executed or not. Generally, this yields a fairer solution in the sense that the tokens are distributed to the traders who value them the most, i.e., to those offering the best limit prices. For the example of a single token pair, Figure 2 illustrates the intuition of the optimization criteria.

Constraints. The desired solution must satisfy several requirements that can be stated on a high level as follows:

(a) *max sell amount* – for every order $\omega = (j, k, \overline{x}, \overline{y}, \pi) \in O$:
The transacted amount y of the sell-token τ_k must not be higher than the maximum sell amount $\overline{y}$ set in the order, i.e., $y \leq \overline{y}$.

(b) *limit price* – for every order $\omega = (j, k, \overline{x}, \overline{y}, \pi) \in O$:
The order can only be executed (fully or fractionally) if the exchange rate satisfies the limit price, i.e., $p_{j|k} \leq \pi$. This guarantees that no trader loses from joining the auction and is commonly referred to as *individual rationality*.

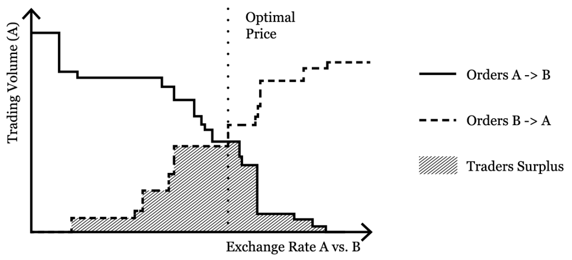

Figure 2: Trading volume vs. trading surplus.
The total trading surplus yields an exchange rate for which the area shaded in gray is maximized, whereas the total trading volume yields an exchange rate where the minimum of the solid and dotted line is maximal. In many cases, the two solutions coincide.

(c) *token conservation* – for every token $\tau \in \mathcal{T}$:
The amount of tokens τ that were bought must equal the amount of tokens τ that were sold across all orders. This constraint is also known as *strong balanced budget*, ensuring that our proposed protocol (as the auctioneer) may not lose or gain any token amounts in the course of running the auctions.

(d) *price coherence* – for all token pairs (τ_j, τ_k):
The exchange rate from token τ_j to τ_k must be reciprocal to the exchange rate from τ_k to τ_j, i.e., $p_{j|k} \cdot p_{k|j} = 1$. This ensures that traders can not increase their token balance by trading on both directions of any token pair.

(e) *arbitrage-freeness* – for all token triples (τ_j, τ_k, τ_l):
As a generalization to the price coherence constraint, the traders also must not benefit from trading along cycles of tokens. This is guaranteed by requiring $p_{j|k} \cdot p_{k|l} = p_{j|l}$. It can be easily verified that this also yields arbitrage-freeness on token cycles of arbitrary length.

(f) *exchange rate normalization* – Given a solution with valid exchange rates $p_{j|k}$ for all token pairs (τ_j, τ_k), without any further condition one could always generate a better solution by multiplying all rates with a constant $c > 1$. To prevent this, impose that $\sum_{j=1}^{n} \gamma_j \cdot p_{j|0} = \gamma_0$ with constants $\gamma_0 \ldots \gamma_n \geq 0$.

(g) *account balances* – For each account that is participating in the auction, the balance of every token after order execution must be nonnegative. In other words, traders are not allowed to take on debts in any token beyond settlement completion.

Observation 1. *Essentially, it suffices to determine all exchange rates $p_{j|0}$, i.e., the token prices with respect to the reference token τ_0. Then, applying the price coherence (d) and arbitrage-freeness (e) conditions, the exchange rate for every token pair (τ_j, τ_k) is immediately given via $p_{j|k} = p_{j|0} \cdot p_{0|k} = p_{j|0} \cdot p_{k|0}^{-1}$.*

Lemma 3.1. *The optimization as described above is well-defined.*

Proof. The proof has two parts – existence and boundedness:

(i) Existence of solution. There always exists a set of feasible exchange rates. The condition (f) can be trivially satisfied by setting $p_{j|0} := \gamma_0/(n \cdot \gamma_j)$. Together with the the conditions (d) and (e), this yields $p_{j|k} = \gamma_k/\gamma_j$ for the all exchange rates. Using these exchange rates, a valid solution can always be obtained through setting the sell and buy amount of every order to zero, i.e., $y := 0$ and $x := 0$, thus trivially satisfying (a)–(c). This leads to an objective value equal to zero for both volume and surplus maximization.

(ii) Boundedness. There can only be a strictly positive objective value if at least two orders exist that can be matched according to their limit prices. Since all limit prices are finite, the respective exchange rates between tokens of matching orders (either on a token pair or along a cycle) need to be finite as well. Hence, both the trading volume and the trading surplus of any order may only take a finite value, so their respective sum across all orders may also only be finite. □

It is possible to find a straight-forward model for our order matching mechanism as described in this section as a continuous non-linear and non-convex optimization problem (NLP). Unfortunately, this class of optimization problems tends to be difficult to solve, with the tractable problem size being rather limited. However, as mentioned before, an equivalent mixed-integer linear programming (MIP) reformulation has been proposed in [33]. Even though the problem is NP-hard, this allows for developing sophisticated pre-processing techniques and employing high-performance commercial MIP solvers such as Gurobi or CPLEX in order to compute solutions for the order batches. Different solution approaches might, for instance, involve a heuristic search based on the prices computed in the previous batch, which could yield good solutions fast in situations where price fluctuations are relatively low.

3.2 Classification of the auction mechanism

Theoretical research [26] classifies auction models by the following criteria and proves that no double auction model can simultaneously satisfy all four of these characteristics:

- *Individual rationality*: A mechanism is individually rational if all players always get a non-negative utility. That is, no one should lose from joining the auction.
- *Balanced budget*:
  - *Strong*: all transfers must occur between buyers and sellers; the auctioneer should not lose or gain money.
  - *Weak*: the auctioneer should not lose money, but may gain money.
- *Truthfulness*: reporting of the true desired value should be a dominant strategy for all players.
- *Economic efficiency*: the total social welfare (sum of all player's value) should be the "best" possible. That is, after trading, the assets should be in possession of those who value them most.

The proposed auction mechanism by design fulfills the individual rationality (via constraints (a) and (b)) while (strong) balanced budget follows from (c).

Economic efficiency holds in the case of two assets as follows: Assume, for a contradiction, that economic efficiency does not hold. This means that there exist two (limit-sell) orders ω_i and ω_j

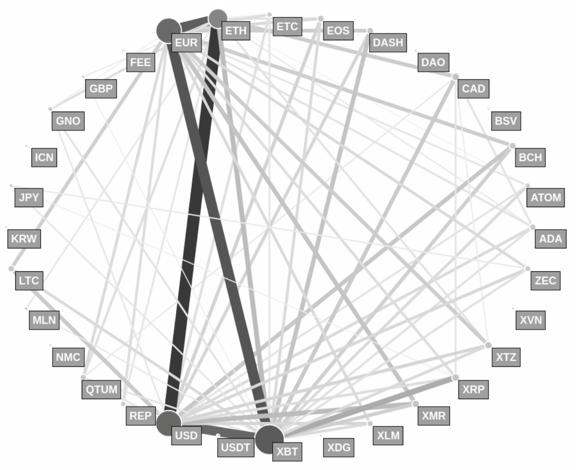

Figure 3: Tokens and orders.
Distribution of orders between a set of assets traded on a centralized exchange. Edge thickness corresponds to the number of orders placed on an asset pair. Even though a subset of the assets is dominant, a significant amount of trade happens between many token pairs.
(data: `www.kraken.com`)

with the same sell tokens at two different prices $\pi_i < \pi_j$ such that ω_j is at least partially filled ($y_j > 0$) while ω_i is not completely filled ($y_i < \overline{y}_i$). This means that participant for ω_j values the buy token less than participant for ω_i but still receives some, while the latter does not get all of their desired quantity. Now, any shift of $0 < \epsilon \leq y_j$ from y_j to y_i increases trading surplus (since $\pi_i < \pi_j$) while still satisfying all constraints.

3.3 Computational results

After demonstrating that our price finding and order matching mechanism is a well-defined optimization problem, it remains to provide empirical evidence that the resulting exchange rates are comparable with actual rates from an existing exchange, and that simultaneously considering the orders for all token pairs yields a benefit in terms of total trading surplus.

For the computational study, we have extracted 50 test instances from the data provided by Kraken, one of the biggest centralized cryptocurrency exchanges. At time of writing, Kraken offers trading on 75 asset pairs with 25 different assets, some of which are dominant with respect to the number of orders placed as compared to others. This reflects our assumption that liquidity is not evenly distributed across all tradable assets. In particular, we have taken snapshots of the order books of all asset pairs that are traded on Kraken at several discrete points in time. It is important to notice that these snapshots only contain limit orders that can not currently be matched by the continuous double-auction mechanism considering orders on every asset pair separately. An example instance as a graph of tokens and orders is shown in Figure 3.

For each of the test instances, we have run a MIP formulation of the surplus maximization problem using Gurobi 8.1.0 as a solver using 16 parallel threads and a time limit of 1000 seconds. The results of these runs are given in Figure 4. The bar chart shows that

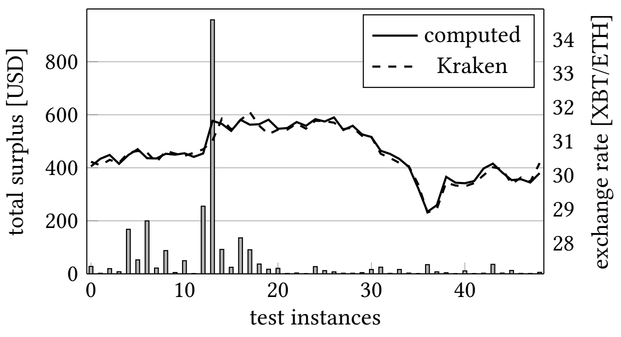

Figure 4: Computed surplus and exchange rate comparison. The bar chart shows the total surplus computed for each of 50 test instances. The line graphs exemplarily depict the exchange rates for XBT/ETH as computed by our optimization model vs. the real-world exchange rate from the data. (data: `www.kraken.com`)

almost all instances yield at least a small surplus between 0 and 50. Individual asset prices are denoted in USD on Kraken, the total surplus value can be interpreted as the overall benefit (measured in USD) that traders can obtain with respect to their stated limit prices. Since no orders can be matched on individual asset pairs, all surplus results from one or several ring trades. Moreover, the figure shows two exchange rate graphs for the most heavily traded asset pair on Kraken (XBT vs. ETH): one containing the exchange rates as taken from the data and the other containing the exchange rates as computed by solving the surplus maximization problem. As both graphs are nearly identical, this indicates that our suggested price-finding mechanism yields realistic market prices on real-world data.

3.4 Limitations

Currently, we expect to be able to treat batches of 1000 orders on a set of 30 tokens with a batch time of 3 minutes. Even when not solved to optimality, the solution obtained after this time period is valid for submission and could generally be considered close to optimal in our tests. Since we are working with a MIP formulation for the optimization problem, the running time generally grows exponentially with the number of binary variables. In our model, one binary variable is needed for every order, hence an increase in the amount of orders leads to significantly longer solving times. However, there are several heuristic approaches to reducing the number of orders that need to be considered. For example, orders with the same or at least a similar limit price may be aggregated. Moreover, the problem may be split into two stages, where the prices are first determined based on a subset of the orders, then subsequently applied to all orders. Increasing the number of tokens that are being considered also increases complexity and thus the expected run-times, even though this only requires the addition of linear constraints. Intuitively, this can be attributed to the larger amount of dependencies in the token-order-graph (see Figure 3). In general, the closer this graph is to being complete, i.e., the more balanced the trade is across all edges, the more difficult the optimization problem becomes. Ideas to circumvent this limitation include partitioning the tokens into subsets that are optimized separately. Finally, the number of accounts is the least restrictive parameter. The account balance requirement (constraint (g) in Section 3.1) yields a number of linear inequalities on the one hand, but narrows the search space for the optimal solution on the other.

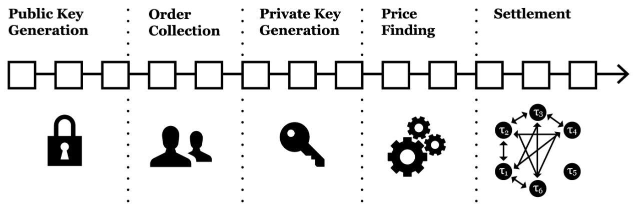

Figure 5: Process picture of the trade mechanism

4 D$\mathcal{F}$USION PROTOCOL

In this section we give a detailed overview over the actors and phases of the protocol. The protocol is summarized in Figure 5.

4.1 Actors

The decentralized trading protocol is determined by four primary actors:

Traders. Traders submit limit orders for each auction and pay a fee per submitted order. They benefit by having their orders executed at the specified or a better price.

DKG-actors. DKG-actors participate in the distributed encryption key generation procedure to shield the content of orders. Each actor is bonded, such that any malicious behaviour can be penalized. They receive a small portion of the collected fees in return for their service.

Price finders. Price finders competitively attempt to compute the optimal value to the optimization problem described in Section 3. They submit feasible solutions to the blockchain. The price finder with the best solution receives the majority of the collected fees.

Anchor contract. The anchor contract is a smart contract in the blockchain and a passive actor that enforces the protocol logic. Other actors interact with it, e.g., to submit orders or solutions. The underlying consensus mechanism of the blockchain resolves disputes about state transitions, deposits and withdrawals from the exchange. The contract stores traded assets, account state and is used to ensure data availability for all clients.

4.2 Trade Mechanism Phases

The decentralized trading protocol is determined by five phases:

Distributed Encryption Key Generation. DKG allows a set of participants – the DKG-actors – to generate a public key, for which the private key is unknown. The key generation process is described in detail in the paper by Tang [30]. Essentially, the DKG-actors generate two entangled messages, one public and one private. Then, they share the public messages to generate the public key. The public key is used to encrypt orders via an *integrated encryption scheme* [15]. Only when a threshold of actors publish their secret pairs, can the private key be generated. This should only happen after all orders have been collected.

If any DKG-actors reveal their secret before the closure of the batch, their bond is taken away and given to those who disclose the dishonest behaviour.

Order collection. During this phase, traders submit their signed, encrypted order(s) to the anchor contract. Once an auction batch has closed, either after a predefined amount of time or once the maximum order capacity has been reached, all further orders are directed into the next batch. Note that a fee is charged on a per-order basis as described in Section 4.3.

Order Decryption. After an auction batch has closed, DKG-actors publish their private secrets among themselves. Once the number of secrets revealed has reached a certain threshold, the private key may be generated and the orders can be decrypted by the price finders. In the event that the threshold is not reached and thus the private key can not be generated, all the DKG-participants are penalized and the auction must be restarted.

Generally speaking, the process has two potential risks:

(1) An auction can be reverted if a sufficiently large subset of DKG-actors do not disclose their secrets and the order decryption threshold is not met.
(2) Similarly, a subset of DKG-actors could collude in order to decrypt orders before an auction batch has closed.

That is to say, a small threshold implies higher potential for collusion, whereas a large threshold could lead to locked auctions. A quantitative analysis of the trade-off between these risks can be performed by a variation of the threshold for the private key generation. Determining the most appropriate threshold is subject to future work.

Decentralized Price Finding. Once order collection for a batch is closed and all information has been decrypted, price finders can begin solving the trading surplus optimization problem (cf. Section 3.1). Theoretically, any client could generate reasonably good solutions by searching for prices in proximity to current market prices or closing prices of the previous auction. However, players with specialized hardware and algorithms are likely to find the best solutions maximizing the total trading surplus. A solution submission consists of the following information:

(1) calculated surplus
(2) price vector containing a price for each asset
(3) the traded buy and sell volume for each order

Solutions are collected by the anchor contract for a predefined period of time. After this period, no further solutions are accepted, unless there was no submission at all. The contract validates submissions according to the constraints contained in Section 3.1 and rejects invalid ones.

Depending on the computational resources of the underlying blockchain, verification may be too expensive to be done in the native smart contract language. Section 5 illustrates a feasible implementation of this computation on Ethereum.

Auction Settlement. The solution submitted with the highest trading surplus is selected by the anchor contract. The account state is updated according to the traded volumes and the submitter is rewarded with a fraction of the collected trading fees.

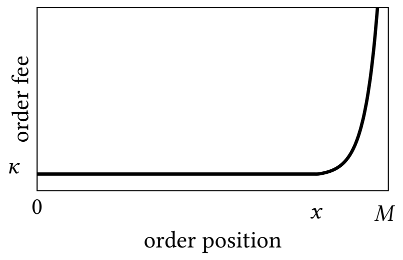

Figure 6: Symbolic representation of fee model for a fixed order limit. Order fee is constant κ and increases exponentially after threshold x.

4.3 Fee model

The fee model is responsible and necessary to reduce the potential attack vectors imposed by the limited order and account capacities in the protocol and incentivize e.g. price finders to participate. Capacity limitations arise not only from the complexity of the optimization problem, but also from the implementation of zkSNARKs described in Section 5.

Although account slot hogging is a potential threat to the usability of the system, accounts do not directly affect the resulting prices of any given auction. This implies that account capacity less susceptible to financial attack and so the fee model is aimed primarily at order capacity.

With this in mind, a functional reservation fee for order slots is imposed as follows: For each order, a fee is charged based on its index within the batch. A vast majority of the orders pays only a marginal constant fee.

Only the last x percent of orders pay an exponentially increasing fee as shown in Figure 6.

The increasing tail of this model prevents a potential denial of service attack where the entire batch is filled with unsatisfiable orders.

5 FEASIBILITY STUDY ON ETHEREUM

In this section the limits and reasonable parameters of the d$\mathcal{F}$usion protocol are explored. In the following case study an implementation of the trading mechanism on Ethereum is proposed. The study concludes that with the state of Ethereum today, the following key-metrics can be achieved:

- *batch time:* Each auction collects orders for 3 minutes.
- *orders:* Each auction contains up to 1000 orders.
- *cost:* Orders have an amortized cost less than 1000 gas.
- *tokens:* Prices are optimized for trades between 32 tokens.

Multiple instances of the exchange with different subsets of tokens can be used to support more than 32 tokens.

5.1 State compression and transition logic

Ethereum has limited capacity for computation and the amount of data that can be handled in each block. Transaction data and instructions for the Ethereum Virtual Machine (EVM) are priced in a unit called gas. At the moment, each Ethereum block is limited to 8 million gas. In order to verify solutions and execute state transitions within the block gas limit, the following compression techniques are applied:

State compression. The state of the application is compressed using Merkle trees [25]. Token balances are represented as leafs so that the root hash of the tree commits to the global account state of the system in a compressed yet collision-resistant form. The cost for storing the root on-chain is constant with regards to the number of tokens and accounts, whereas storing all balances incurs a linear cost.

Merkle trees allow to prove inclusion of leafs with space and time complexity of $O(\log n)$, by recursively hashing the leaf with its sibling along the path to the root and comparing the result with the state root stored on-chain. Such a proof can also be used to update the state root by first proving leaf inclusion in the existing root, updating the leaf and then recomputing the root along the same path.

The state of the application is stored and transitioned by the *anchor contract.*

Order compression. All data, such as orders, deposit requests and withdrawal requests are collected on the blockchain. Users send their requests to the anchor contract which hashes the data and stores its *rolling hash* rather than the data itself.

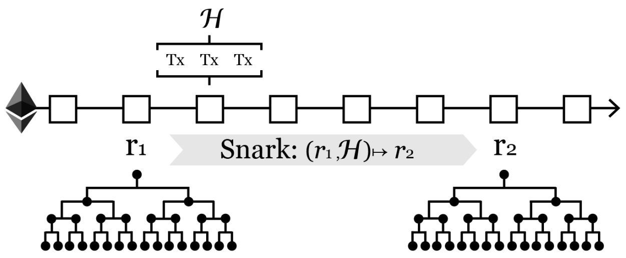

Figure 7: Schematic description of a state and its transition: r_1 and r_2 are consecutive states stored on the Ethereum blockchain. Data is collected in the anchor contract and new state transitions are verified by zkSNARKs.

This hashing technique is defined for deposit, withdraw and order batches as

$$\begin{aligned} h_n &= \mathcal{H}(h_{n-1}, \phi) \\ h_0 &= 0 \end{aligned} \tag{1}$$

where h_i denotes the *rolling hash*, ϕ the object data and $\mathcal{H}$ is a collision-resistant hash function.

This hash serves as a censor-resistant commitment to the submitted orders without the cost of storing them on-chain. While order details are not stored on-chain they are still submitted with each transaction to update the rolling hash and thus part of the transaction history. This guarantees data availability to all actors.

State transition. A state transition is a mapping from one state root to the next. It can represent deposits into the exchange, withdrawals from the exchange and trade settlements.

Transitions between different states of the system are not calculated on the blockchain, instead price finders (or, for deposits and withdrawals, any actor) propose the state-root that results from applying their solution to the current state. Valid state transitions are accepted by all peers. Invalid state transitions can be challenged by any actor using *fraud proofs.* These proofs use the underlying consensus mechanism of the blockchain to resolve the dispute. The verifying logic can be implemented in different ways. This feasibility study uses *succinct non-interactive zero-knowledge proofs* (zkSNARKs) for attesting that a proposed state root is not the correct outcome of a state transition. Since the zero knowledge aspect of such proof system is not relevant for this application, the term SNARK will be used in place of zkSNARK.

The proof is an approximately 300 byte long argument for a predefined function f, a known output y and a private variable x so that $f(x) = y$.

Applied here, y are the commitments stored in the anchor contract (current state root, order root, solution commitment, updated state root) and x are their uncompressed representation as private input. Since x is private it does not have to be known to verify the proof that f was executed correctly. Thus the smart contract does not require any uncompressed data for verification. The function f is an arithmetic circuit. It asserts the data in x is indeed the data compressed in the public commitments y and that the solution provided as part of x fulfills the constraints previously described in Section 3.1. It finally checks that the updated balances evaluate to the updated state root. Verifying such a proof on-chain costs between 1 and 2 million gas.

Successful challenges are rewarded the bond that the invalid state proposer submitted and the transition is reverted by the anchor contract.

State transitions can be challenged for a certain time period (*finality period*), before they are assumed to be correct and finalized. Finalization is needed to process withdrawals (cf. Section 5.3).

5.2 Protocol adjustments

The compression techniques described above lead to an additional validation phase, a two-phased price finding and new validator actor, compared to the original description (cf. Section 4.2).

Validators. Since state updates are not directly verified on the blockchain, there is a risk of invalid states being reported. However, since the logic for state transitions is predefined and all data is made available on-chain, clients can inexpensively redo the computation and locally validate state transitions. Invalid state transitions could e.g. steal funds, execute orders whose limit price is not satisfied, or yield a surplus lower than the one originally claimed. *Validators* have the task of observing state transitions and challenge those which are invalid. When a validator's challenge is successful they receive a reward which is taken from the proposers bond and the invalid state is reverted. Invalid challenges are ignored and incur transaction cost for the validator. Traders can, for example, run a validator node for their own security.

Price finding and auction settlement validation. Price finders submit their solutions to the optimization problem on-chain. This is done in a two step process: First, all price finders only submit their trading surplus to register this solution in the anchor contract. Participating in this phase is inexpensive since it contains only the surplus and not the full solution data.

Once the submission phase ends, the price finder with the highest surplus publishes their solution entirely including all of the prices, trade executions along with the resulting state root obtained by updating balances according to the trade executions. If the price finder fails to provide a solution within a predefined time, their bid is discarded and the process is repeated with the next best bid. Since trade executions are public, validators can quickly verify the correctness of the solution and challenge incorrectness. When a challenge is successful, the process gets restarted with the next best solution. If there is no successful challenge for a certain period, the state transition becomes final and the price finder is rewarded.

In the event that all solutions are invalid, the submission period is reopened.

5.3 Deposits and withdrawals

In order to deposit into the exchange, traders initiate a request to the anchor contract. This request will transfer funds from the trader to the anchor contract and compresses all relevant deposit information (account, token, amount) into a rolling hash (cf. Equation (1)). The anchor contract enforces that between auctions the pending deposit hash is applied to the state root in the same way as solutions are applied (cf. Section 5.1).

Note that the state root cannot change during the price finding period, as price finders are executing potentially long-running computations based on fixed account balances.

The process also applies for withdrawals, although it is more complex. Applied withdrawal state transitions yield a second Merkle root, called *valid withdrawal hash*. It is the root of the tree containing all eligible withdrawal amounts of the current batch and the corresponding accounts as leaves. After the *finality period* elapsed, traders, whose withdrawal was processed, are able to claim the funds from the anchor contract by providing a Merkle proof for their withdrawal balance stored in the withdrawal hash. The tree is limited to 1024 withdrawals per transition. Thus such a proof consists of ten items. During the finality period, validators can still challenge the state transition of withdrawals.

5.4 Scalability limitations

The scalability limitations of the protocol are rooted in the limitations of SNARKs and the Ethereum chain itself.

For the elliptic curve that is pre-compiled into the EVM (alt_bn128 [3]), SNARK proofs can be generated efficiently for up to 2^{28} logical gates [16]. Figure 8 shows how many logical gates our implementation requires as a function of orders per batch. The data was generated with a reference implementation of the settlement transition proof with unencrypted orders using the pepper framework [32] and can be found on github[2].

From the extrapolation shown in the figure, it is concluded that the SNARK can prove the settlement of up to 3000 orders for this concrete test implementation. The number of logical gates depends also on the number of accounts in the system and the number of tokens considered. The referenced implementation was configured

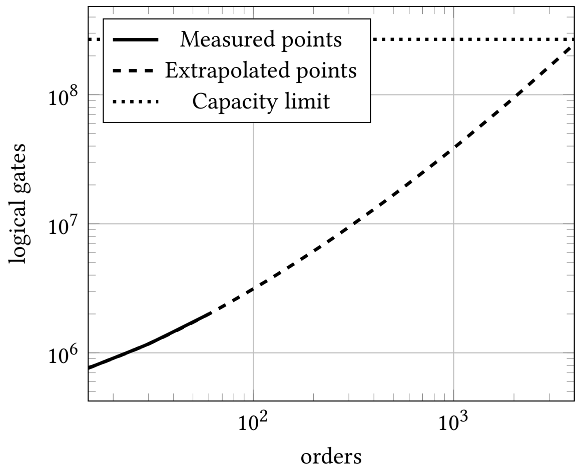

Figure 8: Dependency between logic gates in the SNARK and the number of orders.

with 5 accounts and 10 tokens. With 1000 accounts and 32 tokens, it is expected that 1000 orders can be processed in a single SNARK. Hashing account balances in order to reproduce the state root requires most of the logic gates.

The implementation is not optimized: e.g., SNARK-friendly hash functions and linear, rather than tree-based, hashing layout would yield significant improvements. It is also possible to split the logic over several circuits, each of which proving one aspect of the solution.

The throughput limitation of the Ethereum chain also imposes an upper bound on the number of orders. Moreover, an overload of the Ethereum chain will drive up the the costs for the protocol operations.

5.5 Fee model

The implementation outlined above has two main scarce resources: number of accounts (1000) and number of orders available in each batch (1000). The exchange therefore implements a two-fold fee model, one for reserving an account and another one for placing an order in the exchange. We implement the fee model outlined in Section 4.3 for both accounts and orders. The fees collected for orders are paid to the price finders as a reward for their solution. The account fees are paid to the anchor contract and DKG participants.

For simplicity our analysis is focused on the constant portion of the function as the exponentially increasing part is merely protection against clogging.

Accounts are reserved for a certain amount of time and rent is paid for holding the account. The constant part of the function is \$0.10 per hour, leading to an expected platform revenue of \$72 per trader per month. This revenue is used to finance non-auction related state transitions (deposit & withdraw) and the DKG process.

Orders cost a fixed fee of \$0.10 per order. The solver described in Section 3.3 runs on a cloud server costing \$10 per hour. In a system with batch time of three minutes there are 20 auctions per hour. Thus, the cost of running a solver and submitting a solution is approximately 50 cents per auction. Assuming the probability

[2]https://github.com/aftSubmissionAccount/SnarkCaseStudy

Table 1: Gas cost for interacting with the exchange. $ cost assumes a gas price of 2 Gwei and Ether at $170.

Method	Gas cost	$ cost
Place order	950	$\ll 0.01$
Submit surplus	60 K	0.02
Submit complete solution	550 K	0.19
Fraud Proof	1.5 M	0.51
Deposit	50 K	0.02
Withdraw	90 K	0.03

Table 2: Encoding of order data in bits.

buy/sell token	buy/sell amount	account	**Total**
5 (× 2)	32 (× 2)	24	**98**

of submitting a winning solution is evenly distributed, it becomes profitable for a price finder to participate if there are at least 5 orders per batch. Moreover, we expect that for every additional 5 orders per batch there will be one more price finder for which it is possible to participate. The expected *decentralization* factor Δ of the exchange is therefore

$$\Delta = avg\left(|\text{orders per batch}|\right)/5$$

The cost of submitting the complete solution on-chain is omitted in this analysis as it grows linearly with the amount of orders and thus amounts to a negligible amount per order.

In case of low batch sizes that don't justify price finder participation, simple algorithms that run in the trader's browser can be used to clear auctions.

5.6 Gas cost analysis

A summary of all gas costs is found in Table 1. The calculations assume a cost of 2 Gwei per unit gas and an Ether price of $170.

Order. The cost of submitting an order is a very important value for any on-chain exchange, as this cost directly affects individual traders. It is especially important for market makers, as they need to adjust orders regularly due to the fluctuation of prices. Each order consists of the attributes contained in Table 2.

Recall, these attributes as defined in Section 3.1. Amounts are represented as 32 bit decimal floating point numbers (base 10), 5 bit exponent and 27 bit significand. This means buy and sell amounts can be represented with an accuracy of 8 decimal places. Accounts are indexed according to their position in the state Merkle tree of height 24. With the gas costs of Ethereum as of today, transmitting this amount of data in a transaction (68 gas per byte) costs 884[3] gas, plus 72 gas for hashing it. The base cost of a transaction (20,000 gas) and the cost of updating the rolling hash costs (5,000 gas) is shared between all orders that are submitted in this transaction. This means that the fixed cost of a single order is 0.03 cent.

[3] $884 = 68 \cdot (98/8) \simeq 68 \cdot 13$ (rounding to the nearest byte)

Solution submission. As described in Section 5.2 solutions are submitted in a two step process. At first, only the surplus for a solution is reported to the anchor contract. This requires storing two variables on-chain - roughly 60,000 gas (20 cents).

Submitting the complete solution in contrast is more expensive. As described in Section 4.2 the prices and the trade executions of all trades must be published. Prices and executions are represented in the same format as buy/sell amounts (32 bit). A complete solution of 32 tokens and 1000 orders thus contains roughly 8 kB[4] and roughly 550,000 gas (19 cents).

Fraud proof. Verifying a SNARK on Ethereum requires around 1 million gas plus 100,000 gas for each public input that is being verified. Challenging the state transition of a solution requires five public inputs:

(1) old state root
(2) new state root
(3) rolling order hash
(4) hash of solution
(5) surplus

This costs around 1.5 million gas (51 cents).

Deposit. The cost of a deposit is similar to the submitting an order plus the cost of transferring the token that is being deposited to the anchor contract. An ERC20 transfer invocation costs around 20,000 gas. Moreover, deposits cannot be batched and therefore the base cost for an order as well as the cost for updating the rolling hash cannot be shared among multiple deposits. A deposit therefore costs around 50,000 gas (1.7 cents)

Withdrawal. The cost of withdrawal consists of the request and claiming the tokens. The request submits the withdraw information and updates the pending withdraw hash, which results in roughly 30,000 gas (1 cent). When claiming a withdraw after the state transition is finalized the ten items along the Merkle path are required to prove inclusion in the *withdrawal hash* (cf. Section 5.3). These 320 bytes as well as the cost for transferring tokens from the anchor contract to the trader results in roughly 90,000 gas (3 cents).

5.7 Fund security analysis

The protocol described is non-custodial if the following assumptions holds: Each proposed invalid state transition will be challenged and reverted by some validator within the finality period.

Traders are incentivized to be validators, as it improves the security of their funds and reporting invalid states will be rewarded by the protocol. If in any given finality period at least one trader is online and is willing to report invalid states, then above assumption will hold.

The protocol could require traders to come online and run a validation once a finalization period. This would ensure that each trader has the chance to challenge any invalid state transition. However, under this requirement the finalization period would have to be increased in order to keep the protocol viable for traders, who are unable to frequently come online.

[4] $(1000 \cdot (2 \cdot 32) + 32 \cdot 32)/8$

In the long-term, it is also expected that the calculation of SNARK-proofs and their execution on the blockchain will become considerably cheaper, as the scalability capabilities of blockchains improve and GPUs are employed to calculate proofs [22]. This would allow to evolve the protocol to require a validity proof for each state transition instead of relying on validators submitting fraud proofs.

6 CONCLUSION

This work proposes a new decentralized trading protocol which provides information symmetry between all participants, has no single point of failure and utilizes a new batch trading mechanism.

Being non-custodial, d$\mathcal{F}$usion ensures the security of funds by design. Information symmetry between all participants does not allow front-running and guarantees fair price finding. Traditional regulations, such as customer funds protection and front-running prevention, are directly imposed by the protocol and cannot be evaded.

Trading and liquidity behaviour differs between batch auctions and continuous order book exchanges. Compared to today's dominating DEXs with centralized operators, in the proposed protocol liquidity is not fragmented between token pairs and no central operator has the opportunity to front-run orders. Both aspects are expected to reduce slippage compared to existing DEXs.

Arbitrage trading between batch auctions and a continuous order books involves more risk than arbitrage trading between two continuous order book exchanges, since the matched volume in a batch auction is only known after the solution has been calculated. Therefore, assets can not be hedged immediately in other exchanges.

Due to these considerations, a natural use case for the exchange protocol is trading new digital assets between many stable coins tracking the same currency (e.g. USD). For these markets aggregating liquidity is likely more important than facilitating arbitrage with traditional continuous markets.

The feasibility study shows that the protocol can be implemented on Ethereum today. The capacity of the exchange is currently limited by technicalities, such as the elliptic curve used in the SNARKs and high cost of broadcasting data on-chain. Advancements in computational blockchains are expected to facilitate significantly better scaling.

REFERENCES

[1] Joseph Abadi and Markus Brunnermeier. 2018. *Blockchain economics.* Technical Report. National Bureau of Economic Research.

[2] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2018. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptology ePrint Archive* 2018 (2018), 46.

[3] Jean-Luc Beuchat, Jorge E. González-Díaz, Shigeo Mitsunari, Eiji Okamoto, Francisco Rodríguez-Henríquez, and Tadanori Teruya. 2010. High-Speed Software Implementation of the Optimal Ate Pairing over Barreto–Naehrig Curves. In *Pairing-Based Cryptography - Pairing 2010*, Marc Joye, Atsuko Miyaji, and Akira Otsuka (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 21–39.

[4] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. 2018. Verifiable Delay Functions. In *Advances in Cryptology – CRYPTO 2018*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer International Publishing, Cham, 757–788.

[5] Felix Brandt and Tuomas Sandholm. 2005. Efficient Privacy-Preserving Protocols for Multi-unit Auctions. In *Financial Cryptography and Data Security*, Andrew S. Patrick and Moti Yung (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 298–312.

[6] Paul J Brewer. 1999. Decentralized computation procurement and computational robustness in a smart market. *Economic theory* 13, 1 (1999), 41–92.

[7] Eric Budish, Peter Cramton, and John Shim. 2015. The High-Frequency Trading Arms Race: Frequent Batch Auctions as a Market Design Response. *The Quarterly Journal of Economics* 130, 4 (2015), 1547–1621. https://doi.org/10.1093/qje/qjv027

[8] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. 2019. Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges. arXiv:abs/1904.05234

[9] Sven de Vries and Rakesh Vohra. 2003. Combinatorial Auctions: A Survey. *INFORMS Journal on Computing* 15 (08 2003), 284–309. https://doi.org/10.1287/ijoc.15.3.284.16077

[10] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. 2018. General State Channel Networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. ACM, New York, NY, USA, 949–966. https://doi.org/10.1145/3243734.3243856

[11] Richard Engelbrecht-Wiggans and Charles M. Kahn. 1998. Multi-Unit Auctions with Uniform Prices. *Economic Theory* 12, 2 (1998), 227–258. https://doi.org/10.1007/s001990050220

[12] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. 2019. SoK: Transparent Dishonesty: front-running attacks on Blockchain. *CoRR* abs/1902.05164 (2019), 1–19. arXiv:1902.05164 http://arxiv.org/abs/1902.05164

[13] G. Fenu, L. Marchesi, M. Marchesi, and R. Tonelli. 2018. The ICO phenomenon and its relationships with ethereum smart contract environment. In *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. IEEE Computer Society, Campobasso, Italy, 26–32. https://doi.org/10.1109/IWBOSE.2018.8327568

[14] Ethereum Foundation. 2018. ETH 2.0 specification. https://github.com/ethereum/eth2.0-specs.

[15] Vctor Gayoso Martnez, Luis Hernandez Encinas, and Carmen Snchez ffivila. 2010. A Survey of the Elliptic Curve Integrated Encryption Scheme. *Journal of Computer Science and Engineering* 2 (01 2010), 7–13.

[16] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. In *Advances in Cryptology – EUROCRYPT 2016*, Marc Fischlin and Jean-Sébastien Coron (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 305–326. https://eprint.iacr.org/2016/260.pdf

[17] Yung-Chen Hsieh, Chih-Wen Hsueh, and Ja-Ling Wu. 2018. The Exchange Center: A Case Study of Hybrid Decentralized and Centralized Applications in Blockchain. In *2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN)*. IEEE, IEEE Computer Society, Shenzhen, China, 232–233.

[18] Wen-Shenq Juang, Horng-Twu Liaw, Po-Chou Lin, and Chi-Kai Lin. 2005. The design of a secure and fair sealed-bid auction service. *Mathematical and computer modelling* 41, 8-9 (2005), 973–985.

[19] Rami Khalil, Arthur Gervais, and Guillaume Felley. 2019. TEX - A Securely Scalable Trustless Exchange. Cryptology ePrint Archive, Report 2019/265. https://eprint.iacr.org/2019/265.

[20] Leora Klapper, Luc Laeven, and Raghuram Rajan. 2006. Entry regulation as a barrier to entrepreneurship. *Journal of financial economics* 82, 3 (2006), 591–629.

[21] Avihu Levy. 2019. Scalability First - with STARK. In *Zero Knowledge Summit zk0x03*. Adjy Leak, Berlin, Germany, 12:44–38:38. https://youtu.be/cYgpzX-gjhs?t=764.

[22] Eric Mahe and Jean-Marie Chauvet. 2014. Fast GPGPU-Based Elliptic Curve Scalar Multiplication. *IACR Cryptology ePrint Archive* 2014 (2014), 198.

[23] Jerry W Markham. 1988. Front-running-insider trading under the commodity exchange act. *Cath. UL Rev.* 38 (1988), 69.

[24] Ralph C. Merkle. 1988. A Digital Signature Based on a Conventional Encryption Function. In *Advances in Cryptology — CRYPTO '87*, Carl Pomerance (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 369–378.

[25] Ralph C. Merkle. 1988. A Digital Signature Based on a Conventional Encryption Function. In *Advances in Cryptology — CRYPTO '87*, Carl Pomerance (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 369–378.

[26] Roger B Myerson and Mark A Satterthwaite. 1983. Efficient mechanisms for bilateral trading. *Journal of Economic Theory* 29, 2 (1983), 265 – 281. https://doi.org/10.1016/0022-0531(83)90048-0

[27] Joseph Poon and Vitalik Buterin. 2017. Plasma: Scalable autonomous smart contracts. *White paper* 2017 (2017), 1–47.

[28] Roy Radner and Andrew Schotter. 1989. The sealed-bid mechanism: An experimental study. *Journal of Economic Theory* 48, 1 (1989), 179–220.

[29] Shahar Somin, Goren Gordon, and Yaniv Altshuler. 2018. Network Analysis of ERC20 Tokens Trading on Ethereum Blockchain. In *Unifying Themes in Complex Systems IX*, Alfredo J. Morales, Carlos Gershenson, Dan Braha, Ali A. Minai, and Yaneer Bar-Yam (Eds.). Springer International Publishing, Cham, 439–450.

[30] Caimu Tang. 2005. ECDKG: A Distributed Key Generation Protocol Based on Elliptic Curve Discrete Logarithm.

[31] Jason Teutsch and TrueBit Estsblishment. 2017. On decentralized oracles for data availability.

[32] Michael Walfish and Andrew J Blumberg. 2015. Verifying computations without reexecuting them. *Commun. ACM* 58, 2 (2015), 74–84.

[33] Tom Walther. 2019. An optimization model for multi-asset batch auctions with uniform clearing prices. In *Operations Research Proceedings 2018*, Bernard Fortz

and Martine Labbé (Eds.). Springer International, Berlin, Germany, (to be published).

[34] Daniel Wang, Jay Zhou, Alex Wang, and Matthew Finestone. 2018. Loopring: A decentralized token exchange protocol. URLhttps://github.com/Loopring/whitepaper/blob/master/en_whitepaper.pdf.

[35] Michael P. Wellman, William E. Walsh, Peter R. Wurman, and Jeffrey K. MacKie-Mason. 2001. Auction Protocols for Decentralized Scheduling. *Games and Economic Behavior* 35, 1 (2001), 271 – 303. https://doi.org/10.1006/game.2000.0822

[36] Barry Whitehat. 2018. roll-up: scaling with snarks. https://github.com/barryWhiteHat/roll_up.

[37] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151 (2014), 1–32.