TEX – A Securely Scalable Trustless Exchange

Rami Khalil
Imperial College London
Liquidity Network

Arthur Gervais
Imperial College London
Liquidity Network

Guillaume Felley
Liquidity Network

Abstract

Financial exchanges are typically built out of two trusted components: a trade matching and a trade settlement system. With the advent of decentralized ledgers, that perform transactions without a trusted intermediary, so called decentralized exchanges (DEX) emerged. Some DEXs propose to off-load trade order matching to a centralized system outside the blockchain to scale, but settle each trade trustlessly as an expensive on-chain transaction. While DEX are non-custodial, their order books remains trusted, a malicious exchange operator or miner could front-run trades — i.e. alter trade order execution for financial gain. The scalability limitations of the settlement layer (e.g. Proof of Work (PoW) blockchains) moreover hinders the practical growth of such DEX architectures.

We propose TEX, a front-running resilient, non-custodial centralized exchange. Our matching system enforces the trade order sequence provided by traders, i.e. is resilient against trade sequence alteration by the exchange operator. As such the matching system can operate in conjunction with a blockchain based settlement layer (as proposed in the following), or make custodian exchanges provably accountable for their matching process. Our layer-two settlement system executes a trade without holding the assets, and allows to reach similar scales as traditional exchanges (trading volume in USD, number of trades/second), despite a slow underlying ledger. TEX might become a point of availability-failure, but we show how the settlement system's security properties would not compromise the trader's assets, even if the centralized operator is compromised and/or colludes with all other traders. We provide an evaluation on a PoW blockchain.

I. Introduction

Financial exchanges, such as the New York Stock Exchange, are considered to be fundamentally important to the worldwide economy and trade. Those exchanges are commonly built out of two core components, *(i)* a trade matching system, which brings together supply and demand for different assets, and *(ii)* a trade settlement system which executes matching trades.

A trade matching system is typically designed as a centralized component (e.g. an order book) operated by a trusted third party (TTP), which receives trader orders and upon a match, forwards these to the trade settlement system. Similarly, a settlement system is typically a centralized custodian, i.e. traded assets are temporarily held by banks, brokers and specialist custodians which are trusted to hold and exchange the assets securely. To counterbalance this inherent trust which is put forward to these financial exchanges, regulators conduct periodic and costly audits to unveil potential misbehaviour [1].

Maleficence in market manipulation can for example take the form of so-called front-running. Front-running is defined as the process of exploiting insider information to conduct privileged trades [2]. An exchange's operator could for example step in with its own order, before a large trader order, to draw a financial gain and to ignore the legitimate sequence in which orders were submitted to the exchange. Alarmingly, front-running bears close to zero risk for an exchange operator.

Decentralized ledgers such as Bitcoin provide the ability to transfer digital value without the involvement of a trusted third party. With the advent of smart contracts, more complicated transaction types quickly emerged. One particularly interesting application is the atomic execution of two digital asset transactions - this means that both transactions execute or neither do. This functionality is the foundation upon which a settlement layer for a non-custodial exchange (an exchange does not hold the trader's assets to settle a trade) can be built.

A multitude of decentralized exchanges (DEX) have recently emerged [3–5]. The first DEXs implement both, the order book and trade settlement system on a blockchain smart contract [4], and are rightfully called decentralized. Given the low transaction throughput of existing Proof-of-Work (PoW) blockchains (~10 transactions per second) [6], the operation of early DEXs is both expensive and slow from a user experience (users are e.g. required to pay for non-fulfilled orders).

The second generation of DEXs, moves to a more centralized architecture, by operating a trusted trade matching system, external to the blockchain, while the settlement system remains on the parent-chain. As such, the settlement layer still is limited by the blockchain's transaction throughput, which might explain why currently only a total of about 15'000 trades/day [5] are performed on DEXs. In comparison, a single trader on the currently biggest centralized crypto exchange is able to perform up to 100'000 orders per day [7].

Whether crypto- or custodian exchange, traders have no choice but to trust the matching system to refrain from front-running — potentially causing damages in millions of USD. Moreover, blockchain-based order books leave traders vulnerable to transaction fee bidding and miner front-running. Critically, crypto-currency exchanges are to date unregulated in many jurisdictions, leaving exchanges with little risk for maleficence while malicious behaviour already is reported [8].

This work: TEX present a trustless exchange, which to the best of our knowledge, is the first to prevent an exchange

operator and blockchain miner from front-running trades. Our centralized non-custodial settlement layer is the first of its kind that can scale to the order throughput of custodian centralized exchanges (trades/sec), as well as to their trade volume (USD/-day). The secret sauce of our exchange is the combination of time-lock puzzles [9] and zero knowledge proofs [10] (ZKP) to create novel front-running resilient *moonwalk orders*, coupled with a 2^{nd}-layer commit-chain settlement layer, which operates through a centralized, but non-custodial intermediary. This intermediary cannot misappropriate user's funds. Trades can be settled near-instantly while reducing the load of transactions on a parent blockchain. Our main contributions are as follows:

Front Running Resilience: We introduce the novel concept of a *moonwalk order* that mitigates front-running from exchange operators, miners and adversarial traders, applicable to both custodian (e.g. traditional stock exchanges) and non-custodial (e.g. blockchain-based) exchanges.

Non-Custodial Scalability: We provide a non-custodial 2^{nd}-layer exchange settlement system that scales to trade throughput (number of trades/sec) and trade volume (USD value/day) of custodian exchanges.

Instant Trades: Trades on TEX settle instantly with a configurable amount of collateral from the exchange operator.

Relaxed Online Requirement: For improved usability, traders on TEX are not required to be online at the same time for their order to match, which to our knowledge is not feasible with existing 2^{nd}-layer constructions.

Security: We prove the security of TEX against an irrational and strong adversarial model. Our use of ZKP enforces the correct operation of the exchange, without the need for traders to dispute the integrity of the settlement layer.

The paper is organized as follows. Section II covers background and related work, Section III presents an overview of TEX. Section IV outlines the details of TEX's matching system, Section V presents the details of TEX's settlement layer. We analyze TEX's security in Section VI, evaluate TEX in Section VII and conclude the paper in Section VIII.

II. Background and Related Work

In this section review the background and related work.

A. Financial Exchanges

Financial exchanges allow one party with asset X to find a counter-party to purchase another asset Y. An exchange's architecture is built out of two main components (cf. Figure 1): *(i)* a *trade matching system* and *(ii)* a *trade settlement system*.

The trade matching system supports the trade of different asset-pairs. For each pair of assets (X,Y), the matching system entertains a process to match demand and supply. One such embodiment is an *order book* [11], where traders publish their intent to purchase an asset X at a specific amount of Y, also referred to as an *order*. When a trader specifies a fixed price for an order, the trader issues a so-called *limit order* [12]. When two orders match, the settlement layer executes them.

The settlement layer manages the traders assets securely and to execute trades atomically (i.e. either exchange the assets in both directions, or to not perform any exchange). For custodian exchanges (cf. Figure 1), the trade settlement system is performed by banks and brokers. Non-custodial settlement systems can be built on e.g. blockchains (cf. Section II-E).

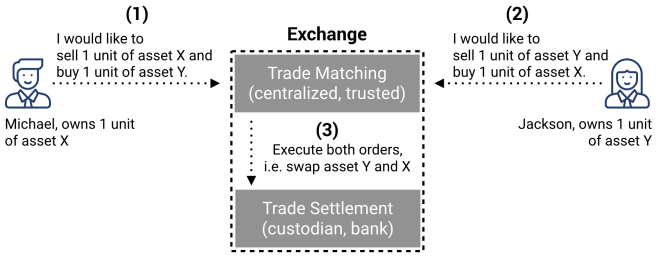

Fig. 1: Overview of custodian exchanges with centralized and trusted order books, that can perform front-running on orders.

B. Centralized (Custodian) vs. Decentralized (Non-Custodial)

Custodian exchanges, e.g. the New York Stock Exchange (NYSE), operate an order book as a trusted third party, while the settlement layer remains the duty of specialist custodians such as brokers or banks. With the advent of decentralized ledgers [13], mutually mistrusting peers can transact and trade, foregoing the need of a TTP — i.e. exchanges become non-custodial, as in they no longer hold the trader's funds.

The first generation of DEXs operate the trade matching and settlement system on the blockchain (e.g. Oasis DEX [4]). This design choice requires the trader to perform a potentially expensive and slow blockchain transaction for every trade. Note that even trades that are eventually cancelled, or not satisfied would entail transaction costs. The majority of the existing blockchains rely on Proof-of-Work [14, 15], and scale poorly (only about 10 transactions per second [6]).

The second generation of DEXs [3], chose to externalize the trade matching layer outside the blockchain (similar to custodian exchanges). Such DEX are more scalable (because orders can be submitted and canceled without blockchain fees) and remain non-custodial. The underlying blockchain, however, remains a throughput bottleneck, and might become prohibitively expensive upon blockchain congestion.

TEX operates a front-running resilient trade matching external to the blockchain, and operates the trade settlement layer on a scalable commit-chain (cf. Section II-E). We provide an overview of the different exchanges in Table I.

C. Trade Order Front-Running

Front-running is the process of exploiting insider information to conduct market manipulation, which can result in a significant monetary loss for traders [2]. Front-running occurs, if e.g. a malicious operator steps in front of larger order with its own order, to gain an economic advantage. In an exchange, an adversary could front-run a trader's order: *(i)* as the operator of an order book (cf. Figure 1), or *(ii)* as a malicious blockchain miner who (re-)orders transactions within a blockchain block. Also *(iii)* adversarial traders can front-run orders by paying higher transaction fees [18, 19].

Real-World Exchange Examples	NYSE*	Oasis DEX [4]	IDEX [3]	**Moonwalk TEX**	**Custodian TEX**	**TEX**
Order Book	TTP	Blockchain	TTP	Moonwalk	Moonwalk	TTP
Trade Settlement	Custodian	Blockchain	Blockchain	Commit-Chain (w/ or w/o zkSNARKS)	Custodian	Commit-Chain (w/ or w/o zkSNARKS)
High Frequency Trading	●	○	○	◐**	◐**	●
High Value Trades (USD/day)	●	●	●	●	●	●
High Trade Throughput (trades/sec)	●	○	○	●	●	●
Funds Controlled by Traders	○	●	●	●	○	●
Blockchain Congestion Resilient	NA	●	●	●	NA	●
Front-Running Resilient	○ (by Exchange)	○ (by Miners)	○ (by Exchange)	●	●	○ (by Exchange)

TABLE I: High-level comparison of financial architectures. TTP is a trusted third party. ● indicates this property is present, ◐ potentially present, or ○ missing. NA stands for not applicable. *New York Stock Exchange. **Improved with dedicated hardware for crypto operations and/or efficient client side proof generation crypto systems [16, 17].

In custodian exchanges, financial regulators conduct periodic audits to uncover misbehaviour [1]. In blockchain-based systems, front-running within an order book operated by a TTP is challenging to detect. If the trade matching operates on a blockchain, then those transactions can be publicly observed and analyzed for front-running [20–22].

LibSubmarine [23] proposes to counter front-running by miners of blockchain auction transactions, but does not seem applicable for trading on order-book based exchanges.

D. Cryptography for a Front-Running Resilient Order Book

We introduce the cryptography for moonwalk order.

1) Time-Lock Puzzle: The goal of a time-lock puzzle is to "send information to the future", i.e. to encrypt data, s.t. it cannot be decrypted by an adversary until a certain amount of time (bounded by computation) passes. One elegant time-lock puzzle is to use repeated squaring in an RSA group [9]. Time-lock puzzles notably satisfy the following properties:

- Puzzles must be proven as solvable with a predetermined amount of computation.
- The puzzle's difficulty to decrypt should be intrinsically sequential. Even an adversary with massive parallel resources should not be capable to decrypt a time-lock puzzle faster than a hardware-restrained adversary.

2) Zero Knowledge Proofs of Knowledge: ZKPoK [10] allow a prover to prove to a verifier that a certain statement is true, without revealing any other information. zkSNARK [24] enhance ZKPs to be non-interactive and succinct (i.e. proof lengths of a few thousand bits), and proofs can be verified with few computational costs (e.g. with a smart contract). zkSNARK, however, require a trusted setup phase.

3) Merkle Mountain Ranges: Appending data to a Merkle Tree [25] can be done by adding new leaves on one side of the tree, e.g. a Merkle Mountain Range [26]. Further extending this to a new Merkle Tree, a signature on the Merkle root may serve as a receipt of the addition of the new leaf.

E. Non-Custodial Trade Settlement Layer

A plethora of proposals aim to scale blockchains [27–35]. We focus on backward-compatible 2^{nd}-layer solutions, that alleviate the burden of the parent-chain. A rich body of work covering different off-chain proposals emerged [34–40].

1) Payment Channels: Payment channels establish private peer-to-peer channels between two parties, that are secured by blockchain escrows. A channel is instantiated and closed with a respective blockchain transaction. For parties that are not directly connected via a channel, a payment can be routed along a set of channels [33, 41, 42]. While channels might be an interesting avenue to construct a 2^{nd}-layer exchange, they face several limitations: *(i)* the exchange operator requires to perform at least one blockchain transaction for each new trader account; *(ii)* collateral in channels is locked statically with each trader; *(iii)* trader need to be simultaneously online to ratify a trade, and *(iv)* the amount of collateral would need to be equivalent to the exchange's trade volume. We chose to design TEX based on a *commit-chain* construction [40], that allows to manage users collateral flexibly, requires less collateral than the transaction volume, and doesn't require a parent-chain transaction to on-board users.

2) Commit-Chains: Commit-chains such as NOCUST [40], are an alternative design to perform off-chain transactions. A NOCUST operator, is a centralized entity that coordinates payments between users that lock their funds in a pool of collateral using a smart contract. This contract expects to periodically receive a constant-sized checkpoint commitment to the state of the commit-chain from the operator, containing each user's account in the collateral pool. The commitment to this (potentially) large state is constructed s.t. it is efficient to prove and verify in the smart contract that a user's commit-chain account is updated correctly by the operator, s.t. transfers, withdrawals and deposits can be securely enacted.

The operator becomes only a single point of failure for availability, but not custody of funds or integrity of operation. The complete disappearance of an operator, or malicious attempts by it to double spend or seize user funds in NOCUST only leads to its halt, and does not affect the ability of users to exit the smart contract with their latest confirmed balances.

NOCUST users are not required to be constantly online, but expected to monitor the blockchain regularly to observe checkpoints. User are only required to verify their respective balance proof by requesting the partial Merkle tree of the checkpoint from the operator and comparing it to the locally stored state. In case of misbehaviour, a user can issue a challenge using the NOCUST smart contract to force the server to

answer this challenge with valid information. NOCUST also supports a provably consistent mode of operation through the use of zkSNARKS. Note that commit-chains do not rely on an additional consensus mechanism as side-chains [43].

III. TEX Overview

In this section, we provide a high level overview of TEX.

A. Untrusted Centralized Intermediaries

TEX operates as a centralized, but untrusted exchange, and represents a single point of availability failure, however, does not have the capacity to front-run traders' orders, or to misappropriate users' digital assets. If a TEX instance were to act maliciously (i.e. front-running or double-trade attempts), or remain unresponsive, the smart contract forces the TEX instance to halt, and users can securely retrieve their funds anytime. Users can easily migrate to another TEX instance.

B. System Model

Our system model assumes the following entities:

Ledger: A tamper-proof, smart contract enabled ledger acting as message ordering and coarse time-stamping service.

Trader: Trader submit limit orders to the order book and have at least one private/public key pair for their ledger account which is also used to trade on TEX.

Order Book: Collects orders from traders, provides trade-receipts and matches trades to be executed.

Trade Settlement System: Executes orders authorized by traders on a commit-chain (which commits its state periodically to the ledger), without holding the funds.

C. High-Level Operations of TEX

The high-level operations are visualized in Figure 2 (order book) and Figure 3 (trade settlement system) respectively.

1) Order Book: The order book receives orders from traders, matches and forwards them to the settlement layer:

1) Jackson creates a moonwalk order hiding the order details (such as price, assets, trader address), and a ZKP to prove that the encrypted order is semantically valid and that Jackson owns sufficient assets to execute the order.
2) Jackson sends the moonwalk order to the order book, which verifies through the ZKP that the time-lock puzzle is semantically correct (verification times are small).
3) The order book creates a Merkle Mountain Range commitment [26], appends the order and acknowledges the trade with the signed commitment root (trade-receipt).
4) After receiving the receipt, Jackson either provides the solution to the puzzle, or the exchange is required to decrypt it (by investing a few seconds of computation).
5) Jackson repeats step 1-4 (with empty orders), and measures the time delay in which the order book responds with a receipt. Repeated low latency responses increase Jackson's confidence that the order book does not decrypt the moonwalk order, before providing the binding receipt.
6) The order book commits at regular time intervals the order state on-chain, which can be verified by the traders that have access to the trade-receipts.

2) Trade Settlement System: The settlement system (cf. Figure 3) interacts at regular *eon* time intervals with the smart contract [40]. A checkpoint aggregates all trades that are matched and commits those on the parent-chain. For simplicity, we omit the bootstrapping of the exchange or registration of new users, and refer the reader to NOCUST [40]. For readability, we present the orders in Figure 3 as cleartext, while each order is encrypted as outlined previously (cf. Section III-C1). The following steps settle a trade:

- Michael (1) converts 1 X parent-coin into a commit-chain asset, by depositing coins into the TEX smart contract.
- Michael (2) signs a state update for its X coin commit-chain ledger to be debited conditionally on being credited in its Y coin ledger. Michael also signs a state update for its Y coin ledger to receive this conditional credit.
- The trade settlement system (3) ratifies Michael's order.
- Jackson does symmetrically the same for its Y, X accounts (4), the exchange (5) ratifies the opposing order.
- The exchange (6) matches Michael's and Jackson's order.
- TEX (7,8) ratifies the fulfillment of the orders.
- Every *eon*, TEX (9) submits a checkpoint to the smart contract that applies the commit-chain balance updates.
- Jackson (10) and Michael both verify the validity of the checkpoint. Given our commit-chain ZKP extension (cf. Section V-D), the integrity of the checkpoint is directly enforced without interaction by the traders.

a) Conversion between Parent- and Commit-Chain: Analog to NOCUST [40], TEX allows the asset conversion between their parent- and commit-chain representation, with a deposit and withdraw operation towards the TEX smart contract. We represent every coin with its individual entry in the Merkleized Interval Tree (cf. Figure 4), comprised of a local and a global ledger representation (cf. Section V).

b) Commit-Chain Exchange: Once two order match, the settlement system enacts the financial asset exchange securely. We inventively extend NOCUST [40] to enable the non-custodial atomic exchange of digital assets, by providing the following functionality. Note that each trade involves two coins that are being exchanged and each coin is represented independently in the commit-chain ledger.

Conditional Debit Authorization: Trader Michael agrees to a new state update by cryptographically signing to debit e.g. an amount of X coin conditionally on being credited an amount of Y coin. The opposite trader Jackson does symmetrically the same.

Conditional Credit Authorization: Trader Michael then agrees to update its Y coin ledger with a conditional credit, and Jackson does symmetrically the same.

Ratification: The server, having the four state updates (two per trader), counter-signs these authorizations and sends them back to Jackson and Michael respectively.

c) Temporal Order Validity: A trade order should ideally be valid as long as it's not satisfied (or canceled). Because TEX, however, operates in well defined *eon* intervals, an order needs to be authorized for each *eon*. To ensure that only the

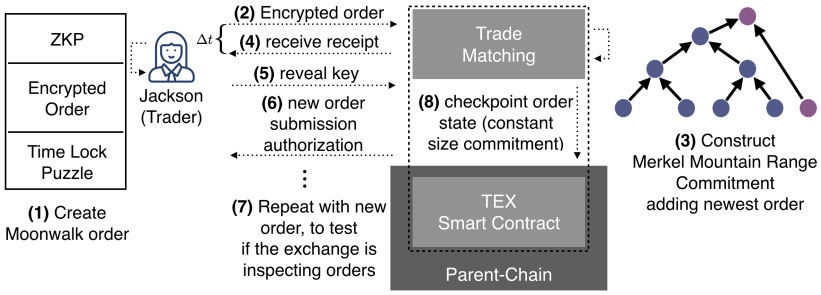

Fig. 2: High level overview of the order book. Jackson (1) creates a time-lapse encrypted order, composed of an encrypted order, a ZK proof and e.g. a timed commitment. Jackson (2) sends the order to the exchange, which (3) appends it to the Merkle Mountain Range of all orders. Jackson (4) receives the receipt that the order was appended and (5) reveals the key to decrypt the order. The exchange (6) returns a new order submission authorization. Jackson (7) repeats steps 2-6 several times to test if the exchange attempts to inspect orders. The order book (8) submits regularly the order state on the parent-chain.

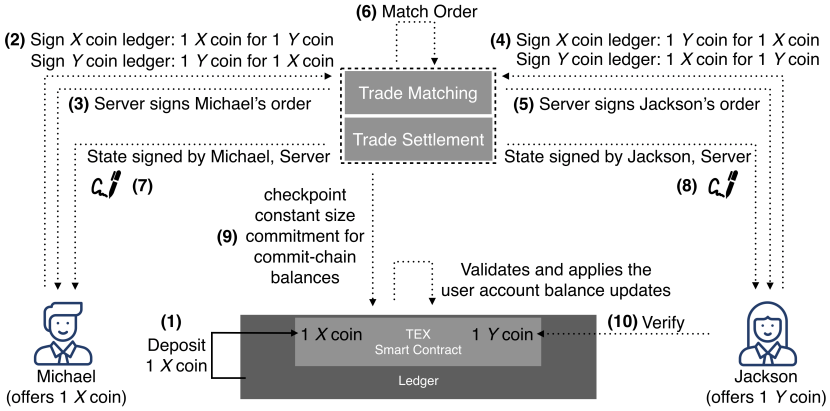

Fig. 3: High level flow of the trade settlement layer. Michael (1) deposits 1 X coin, Jackson deposits 1 Y coin into the TEX smart contract. To enact a trade, Michael (2) signs a state update for his X coin ledger, that he is willing to debit 1 X coin for 1 Y coin (symmetrically for his Y coin ledger). The server (3) signs Michael's order. Jackson (4) mirrors symmetrically the same operation, the server (5) confirms her order. Orders are matched (6), the server signs the state updates (7,8). At regular *eon* time intervals, states are settled (9) on-chain. Michael and Jackson (10) verify the correct ledger updates.

desired amount is traded, a trader is required to transfer the trade amount to a new commit-chain account, and authorize the trade over several TEX *eons*. Note that commit-chain transfers are instant and free of parent-chain transaction fees.

TEX supports active and passive enactment [40]. Under active enactment, an account is blocked once it authorizes an order until that order is either cancelled or fulfilled, while passive enactment enables concurrent orders.

d) Blockchain Settlement and Dispute: Settlement is enacted within regular time intervals, *eons*, or rounds, whereby the server submits a *constant-sized* checkpoint on the parent-chain (cf. Figure 3). Traders with commit-chain assets should audit the submitted checkpoint each *eon* by requesting the partial Merkle Tree from the TEX server. If the checkpoint does not match the trader's assets, a subsequent dispute can be initiated by the TEX smart contract. If the TEX server is not capable to appropriately respond to the challenge, the smart contract halts the exchange's operation. Given our ZKP commit-chain extension in Section V-D, the exchange is immediately halted by the verifying smart contract upon submission of an invalid state transition. Trader no longer need to verify a checkpoint's integrity.

D. Main Properties

TEX provides the following properties:

Ledger: PoW blockchain transaction throughputs restrain blockchain based exchanges. TEX only requires the periodic submission of a constant-sized checkpoint, irrespective of the number of trades executed.

Trader: A trader can convert parent-chain assets to commit-chain assets. Commit-chain assets can be traded with any other trader on the same TEX instance. When a trader submits an order to the order book, the trader can confidently guess whether the operator tried to front-run the order. A trader is required to be online to initiate an order, however, is *not* required to **remain** online for the order to securely match and execute with another trader's order. Regarding asset security, a trader remains custodian of its fund during trades and in the presence of another malicious trader, and/or a TEX server colluding with all

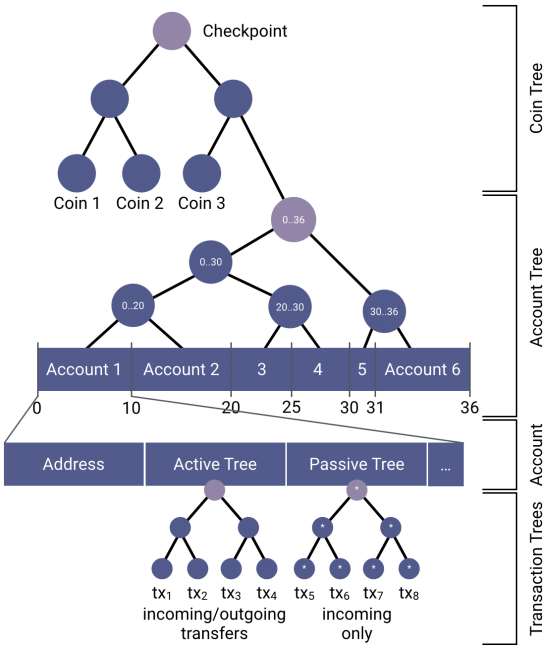

Fig. 4: TEX's commit-chain is based on a *Merkleized interval tree* to account for coins and trader account balances.

but one honest trader. Concerning privacy, traders do not learn with which counter-party their orders matched with on TEX, but only know that their trade was executed. Trade finality can be instant given collateral provided by the exchange as described in the next section.

Exchange: TEX server's duty is to match trades according to public and verifiable rules. Trades settle first on the commit-chain, then the server must submit checkpoints to the parent-chain on-time, as the smart contract would otherwise halt it. The TEX server cannot double trade (without being detected by at least one honest trader, or halted given our ZK extension), nor create more assets than have been deposited on the parent-chain.

1) Collateral for Instant Trade Finality: Commit-chain trades exchange one asset X for another Y, and are finally settled once a checkpoint passed its challenge period on the parent-chain. If a TEX instance is halted, the not-yet checkpointed trades are reverted. Assuming no value fluctuation between X and Y, no trader would entail a financial loss.

In the more realistic setting where traded assets X,Y have diverging price performances, the collateral requirements to instantly accept a trade as settled, depends on the trade volume and the change in volatility between the two assets. The exchange operator configures this volatility percentage as one of its operational parameters (cf. Section VII-A2). We prove in Section VI-C how a user can securely reclaim funds from a trade that was reverted due to a halted TEX instance.

2) Liveness Requirements: TEX requires traders to monitor the underlying ledger at least once per *eon* to verify that their funds remain consistent. Traders can choose to outsource the blockchain monitoring, if they are willing to share their trade information with a third party that can on their behalf challenge their commit-chain state. Assuming a trusted zk-SNARK setup (cf. Section V-D), the integrity of the settlement layer checkpoint is enforced automatically. Traders would still require to come online once per *eon*, or outsource their state data retrieval and liveness verification to a third party.

E. Attacker Models

In this work we assume the following adversarial models.

1) Order Book Adversarial Model:

Adversarial Exchange Front-Running: The exchange operator attempts to front-run orders from traders. To do so successfully, the operator needs to solve the time-lock puzzle, before providing a trade-receipt to the trader. The trader monitors the response-time of the operator and can thus detect suspiciously long response times.

Adversarial Trader Withholds the Moonwalk Order Key: A malicious trader withholds the order's key after reception of the trade-receipt. The exchange must then invest (a few seconds of) computational resources to solve the order. Multiple unreleased orders can be decrypted in parallel, repeated trader misbehaviour can entail a platform ban. This attack induces a temporary *order book blindness*, other traders do not see the latest order book, until the order is decrypted. This attacker is not able to submit another order, as the operator withholds the new order submission authorization.

2) Settlement System Adversarial Model:

Adversarial Trader: A malicious trader attempts to double trade commit-chain funds, the TEX server intervenes and does not ratify such trades. The trader attempts to provide an invalid order (e.g. overdrawing its balance, incorrect amounts, asset pair, etc.), the moonwalk order, however, would attest to the validity of the order.

Adversarial Trader colluding with the Exchange Operator: If all but one trader act maliciously and collude with the server, the adversary can potentially (1) double-trade or (2) create commit-chain assets indefinitely. Double-trades are detected by the remaining honest trader, which challenge the TEX's honesty. Our ZKP extension ceases the contract's operation if the operator attempts to commit to an invalid state transition. Creating commit-chain assets is rejected by the smart contract which enforces the full collateralization of assets.

Adversarial Blockchain Miner Front-Running: A miner tries to front-run trades of TEX. Trades, however, are performed on the commit-chain, where blockchain miners have no influence on transaction execution.

F. Notations

We denote an instance of TEX as $\not\subset$. The exchange server $\mathbb{O}_{\not\subset}$ is supervised by the parent-chain smart contract $\mathbb{V}_{\not\subset}$. Out of the set of $\mathbb{P}$ traders, a trader $\mathbb{P}_i$ performs a trade or swap $\mathfrak{X}_i$ with $\mathbb{P}_j$. Table III (cf. Appendix) provides a notation overview.

IV. TEX Moonwalk Order Details

In this section we present our moonwalk order model. The order book does not gain any information about an incoming order, and is expected to confirm incoming orders before learning their details. Insiders attempting to front-run orders are degraded from risk-free profiteers, to regular risk bearing traders due to lack of their ability to decide if a front-running operation is profitable before committing to enacting it in-time.

A. Delayed Key Disclosure

TEX requires that a trader $\mathbb{P}_i$ and the exchange operator $\mathbb{O}_{\not\subset}$ initialize a special purpose delayed key disclosure agreement that allows the trader $\mathbb{P}_i$ to send a trade $\mathfrak{X}_i$ to the exchange $\mathbb{O}_{\not\subset}$ without revealing that it belongs to a particular trader $\mathbb{P}_i$. Note that this trading ledger is the one containing the *not yet* satisfied trade order. This setup still allows the exchange $\mathbb{O}_{\not\subset}$ to trust that the trade $\mathfrak{X}_i$ can be safely appended to its trading ledger. Importantly, this step happens *prior* to decryption of the trade. Moreover, we condition that the trade $\mathfrak{X}_i$ passes the non-revealing validations in Section IV-B.

We rely on $\mathbb{P}_i$ creating a hash chain typical of delayed key disclosure setups. A random seed value K_0, is processed through a one-way hash function, referred to as H, l times, and the final output of the hash chain K_l is initially revealed to $\mathbb{O}_{\not\subset}$. Authenticating the jth message is done through revealing $H(K_j||\mathbb{P}_i)$ and encoding K_j within a timelock puzzle. For $\mathbb{P}_i$ to create valid moonwalk order (cf. Section IV-B), $\mathbb{P}_i$ requires $\mathbb{O}_{\not\subset}$'s signature on $(K'_{j-1}||\text{update}_i^X)$ for every asset X in $\not\subset$ (note that update_i^X represents the last state update of the commit-chain account of $\mathbb{P}_i$ for asset X [40]). Therefore, past every successful decryption of a $\mathfrak{X}_i$ sent with a reference of $H(K_j||\mathbb{P}_i)$, $\mathbb{O}_{\not\subset}$ needs to send $Sig_o(K'_j||\text{update}_i^X)$ to $\mathbb{P}_i$ for every asset X in $\not\subset$. This permits $\mathbb{O}_{\not\subset}$ to realize when multiple orders use the same reference, and throttle any $\mathbb{P}_i$ that attempts to cause an inconsistency in $\mathbb{O}_{\not\subset}$'s view of $\mathbb{B}$, or repeatedly refuse to handover decryption keys that spare $\mathbb{O}_{\not\subset}$ from solving the time-lock puzzle after attesting to receipt of the order. Running the verifications in Section IV-B allows $\mathbb{O}_{\not\subset}$ to verify that a $\mathfrak{X}_i$ is received with respect to some pre-agreed upon reference without knowing what that reference is or who it is established with prior to decrypting $\mathfrak{X}_i$.

B. Moonwalk Order Generation

The moonwalk order's ZKPs allow to prove to $\mathbb{O}_{\not\subset}$ that solving the reasonably constrained time-lock puzzle unlocks a valid trade $\mathfrak{X}_i$, without revealing the order details or the identity of its originator. This motivates the exchange $\mathbb{O}_{\not\subset}$ to sign a receipt for a $\mathfrak{X}_i$ upon request without knowing its exact contents, but knowing that it would eventually be able to decrypt its contents through solving the time-lock puzzle, and that its contents are a *logically sound order* with respect to its view of $\mathbb{B}$. Algorithm 1 first verifies that the puzzle key leads to the decryption of the expected order, and then verifies that the encrypted order is semantically correct. The ZKP of this procedure can be proven with a Bullet Proof [17] or STARK [44], likely improving its proof generation time. If a system with a trusted setup is utilized, it is safe to completely rely on $\mathbb{O}_{\not\subset}$ to perform this setup, as generating fraudulent proofs of these two procedures will harm no one but $\mathbb{O}_{\not\subset}$.

Algorithm 1: verifyMoonwalk

Verifier Input: n, C_K, C_M, H($\mathfrak{X}_i$), ref
Prover Input : update_i^X, update_i^Y, K_j, κ_h, $\mathfrak{X}_i$, σ, $\mathbb{P}_i$
assert ref $= H(K_j||\mathbb{P}_i)$
$\kappa_g \leftarrow \kappa_h^{\Pi_{i=1}^{r} q_i} \mod n$
assert $C_K = \kappa_g^2 \mod n$
assert $SYM - DECRYPT(C_M, \kappa_g) = \mathfrak{X}_i$
$K_{j-1} \leftarrow H(K_j)$
assert valid $Sig_o(K_{j-1}||\text{update}_i^X) \in \sigma$
assert valid $Sig_o(K_{j-1}||\text{update}_i^Y) \in \sigma$
assert $\mathfrak{X}_i$ is applicable to update_i^X and update_i^Y
calculate new update_i^X and update_i^Y using $\mathfrak{X}_i$
assert $Sig_i(\text{update}_i^X)$ and $Sig_i(\text{update}_i^Y) \in \mathfrak{X}_i$

It still remains to prove the correctness of the generated time-lock puzzle. For this purpose, we convert the interactive proof presented in [45] into a non-interactive one using the Fiat-Shamir heuristic [46]. Algorithm 11 (cf. Appendix) specifies the verifier for this stand-alone zero knowledge proof (not to be embedded in a SNARK or otherwise). To secure our non-interactive version using the recommendations in [45], we require the simulation of s different repetitions to attain a sufficient security level. The parameters and notations are utilized as presented in [45].

We forgo the use of the BBS [47] generator as done in [45] and instead utilize the square root κ_g of the tailing quadratic residue C_K as the symmetric encryption key of $\mathfrak{X}_i$. As g generates a subgroup of order $\phi(n)/4$, and finding κ_g from C_K is difficult without factorizing n, for sufficiently large n an adversary cannot feasibly calculate $\kappa_g \equiv \sqrt{C_K} \mod n$ [47], exhaust the search space for $_g$ or generate a valid ZKP for Algorithm 1 using $\kappa_g \neq \sqrt{C_K} \mod n$.

C. Blind Receipts

As $\mathbb{O}_{\not\subset}$ receives orders from traders in $\mathbb{P}$, it builds up the order book, whose constant-size commitment will be written to the parent-chain ledger $\mathbb{B}_G$ in the next *eon*. When a new $\mathfrak{X}_i$ comes in from a $\mathbb{P}_i$, $\mathbb{O}_{\not\subset}$ is expected to provide a blind trade-receipt confirming $\mathfrak{X}_i$'s placement in the order book *prior* to the decryption of the time-lock puzzle and any potential investigation of its contents. Note that the exchange $\mathbb{O}_{\not\subset}$ only provides the trade-receipt after verification of the provided ZKP from Section IV-B, whose successful verification only reveals the correctness of an order, not the order details.

Receipts contain a Merkle Mountain Range commitment [26] that attests to the state of the order book at the time of receiving the order, and the hash of the $\mathfrak{X}_i$ sent by $\mathbb{P}_i$. This allows $\mathbb{O}_{\not\subset}$ to commit to the sequence of incoming orders, and provide a commitment (trade-receipt) to a $\mathbb{P}_i$ who may then release the decryption key for $\mathfrak{X}_i$. The receipts can later be used to initiate a front-running challenge against $\mathbb{O}_{\not\subset}$ using $\mathbb{V}_{\not\subset}$ by simply providing the signed receipt. The answer to such a challenge is made up of path up to the Merkle root

of the final commitment of $\mathbb{O}_{\not\subset}$ to the order book, where any left siblings on the math must not be different from the ones used to generate the receipt, as shown in Algorithm 2. Because we require the order book commitment Θ to only accepted by $\mathbb{V}_{\not\subset}$ if it passes the recursively composed validation procedure (cf. Section IV-E), then a valid answer to the front-running challenge also implies a correct set of matches for that order.

Algorithm 2: closeFrontRunningChallenge

Input : $\mathfrak{X}_i^{X,Y}(e)$, $\lambda(\mathfrak{X}_i^{X,Y}(e) \in \Theta(e)^{X,Y})$
node ← H($\mathfrak{X}_i^{X,Y}(e)$);
mr_commitment ← H($\mathfrak{X}_i^{X,Y}(e)$);
for *sibling in* $\lambda(\mathfrak{X}_i \in \Theta(e)^{X,Y})$ **do**
  if *sibling is a left child* **then**
    node ← H(sibling ‖ node);
    mr_commitment ← H(mr_commitment ‖ sibling);
  else
    node ← H(node ‖ sibling);
  end
end
assert that the calculated mountain range commitment matches the one specified in the receipt used to open the challenge;
assert node = $\Theta(e)^{X,Y}$;
close $\mathfrak{F}_i^X$;

Regardless of the number of orders performed by a single trader $\mathbb{P}_i$, at most two challenges need to be issued by a $\mathbb{P}_i$ against a non-cooperative $\mathbb{O}_{\not\subset}$ to ensure that none of its orders were potentially front-run. This small upper bound of two is enforced by $\mathbb{P}_i$ refraining from submitting a new nth $\mathfrak{X}_i$ until it receives a mountain-range membership proof of inclusion of its n-2th order in the receipt of its n-1th order. This guarantees the cohesiveness of the last two receipts s.t. a response to a challenge using the n-1th receipt inductively implies that all orders prior to it remained in the same position.

D. Behavior Analysis

Traders in TEX are not perfectly immune from front-running by $\mathbb{O}_{\not\subset}$. A malicious server can always refuse to provide a blind receipt, solve the time-lock puzzle, decrypt the moon-walk order and then decide to front-run it. Even in this worst case scenario, the limit price set by $\mathbb{P}_i$ is still enforced by the settlement layer. In this section we propose a method for $\mathbb{P}_i$ to score whether $\mathbb{O}_{\not\subset}$ is behaving adversarially or not in a fault tolerant manner.

$\mathbb{P}_i$ can infer whether $\mathbb{O}_{\not\subset}$ attempted to front-run an order or not by measuring the time taken by $\mathbb{O}_{\not\subset}$ to respond with a blind receipt. A response time by $\mathbb{O}_{\not\subset}$ that is significantly lower than the estimated time to solve the time-lock puzzle implies a high likelihood that $\mathbb{O}_{\not\subset}$ is not malicious. In high throughput scenarios, the response time will increase, and consequently, the time-lock puzzle difficulty should also increase to compensate.

$$F(p, d) = \alpha * p - \beta * d \tag{1}$$

Equation 1 is a simple scoring function for $\mathbb{P}_i$ to score $\mathbb{O}_{\not\subset}$'s front-running behavior. The two parameters p and d denote how many orders were issued a prompt blind receipt by $\mathbb{O}_{\not\subset}$, and how many had suspiciously delayed response times. $\mathbb{P}_i$ can populate Equation 1 with its own satisfactory α and β values that denote its reward and penalty values for prompt and delayed responses. The decision that a response time is considered prompt or late can be taken based on an arbitrary function, and Equation 1 can also be replaced with a more complicated method. In this work, we simplify the response promptness decision to be based on not exceeding a threshold of $\frac{1}{4}$ the estimated amount of time it would take a theoretical 10GHz CPU to solve the time-lock puzzle.

Through evaluating its scoring logic, $\mathbb{P}_i$ can then predict if $\mathbb{O}_{\not\subset}$ is prematurely decrypting orders. Notably, $\mathbb{P}_i$ has the advantage that it can send zero-valued orders to $\mathbb{O}_{\not\subset}$. This allows $\mathbb{P}_i$ to gather data for its scoring function without putting itself at any risk of being front-run.

E. Order Matching Constraints

To provide resilience against front-running, $\mathbb{O}_{\not\subset}$ must be constrained on how to match orders with each other. The set of matchings must remain coherent with the sequence in which $\mathbb{O}_{\not\subset}$ received (and committed to) the orders. Also, $\mathbb{O}_{\not\subset}$ must commit to the matchings it has created with a ZKP that the contents of this commitment conform to those constraints:

1) No order C that comes after an order B, may match with an order A that precedes B, if B may still match with A.
2) If an order B matches with an order A, then A must precede B (Matches are unidirectional from new orders to previous ones).
3) If an order A may match with an order B, then this match must be carried out.

F. Provably Consistent Orderbook Matching

We now show how to prove that an order book satisfies the previously mentioned constraints. In the Appendix C-A, we present Algorithms 7 and 8 that together comprise a recursively composable ZK proof that the full contents of the order matching book conform to the constraints specified in Section IV-E. These procedures are constructed to provably transition a commitment to an order book with no matches to one where all orders are fully matched according to the defined constraints through utilizing an annotated min-max Merkle tree to verify the matching decisions taken at each transition. The annotations on internal nodes in the tree are the minimum selling price and maximum buying price of orders in the sub-tree of this node. The leaves (orders) are annotated with the remaining volume that can be matched. We prove in Section VI-B that the order book state transitions can be secured using our min-max tree method.

V. TEX Settlement Details

This section outlines the details of TEX's settlement layer. To disqualify TEX from being a custodian of exchanged funds, it would have to provide the following guarantee to a participant $\mathbb{P}_i$ wishing to exchange coins of type X for a different type Y at some exchange rate:

- $\mathbb{P}_i$'s X-coin wallet may not be debited unless $\mathbb{P}_i$'s corresponding Y-coin wallet is credited with at least the amount $\mathbb{P}_i$ requested in return.

This guarantee must be provided regardless of the behavior of an instance of TEX ($\not\subset$) as long as $\mathbb{P}_i$ remains honest

A. Commit-Chain Ledger

In this section, we provide a commit-chain ledger $\mathbb{B}$ specification to support the parallel management of different coins and secure swapping. For each supported coin X, a global $\mathbb{B}_G^X$ and local ledger $\mathbb{B}_L^X$ is managed by $\not\subset$. Both ledgers are combined to create a single parent ledger $\mathbb{B}$ encapsulating all balances within $\not\subset$.

a) Local Information Amendment: For every $\mathbb{P}_i$ in every *eon* e, the local ledger $\mathbb{B}_L^X(e)$ stores $T_i^X(e-1)$, the commit-chain transactions involving the participant i. Transactions are defined in TEX to mean either a transfer $\mathfrak{T}_{i,j}$ from a $\mathbb{P}_i$ to a $\mathbb{P}_j$ or a swap operation $\mathfrak{X}_i$ by $\mathbb{P}_i$.

b) Global Information Amendment: For every $\mathbb{P}_i$ in every *eon* e, the global ledger $\mathbb{B}_G^X(e)$ can store in addition $\mathfrak{S}_i^X(e)$, a challenge against a coin X swap for $\mathbb{P}_i$ in $\not\subset$.

Section V-C outlines how $\mathbb{B}$ enables secure swaps in $\not\subset$.

B. Periodic Commitments

This section describes our commitment structure to support the secure commitment to multiple coin types and maintain the provable integrity of a TEX *eon*. Algorithm 3 creates a commitment to attest to the states of accounts of different coins within TEX. $\mathfrak{A}$ is an exclusive balance allotment tree, a Merkle tree commitment to the individual *annotated* Merkle tree commitments, $\mathfrak{A}^X$, constructed for each supported coin X. The upper bound of the size of a proof of exclusive allotment in TEX for a $\mathbb{P}_i$ and a given coin X, $\mathfrak{A}_i^X$, is $\mathbb{O}(\log|\mathbb{P}| + \log|\mathbb{M}|)$, where $\mathbb{M}$ represents the coins supported by $\not\subset$.

Algorithm 3: combineAllocationCommitments

```
Input  : 𝔹
Output: 𝔄
roots ← {};
for 𝔹^X ∈ 𝔹 do
    𝔄^X(e) ← createAllocationCommitment(𝔹^X);
    roots ← roots ∪ {Root(𝔄^X)};
end
𝔄 ← merkleTree(roots)
for 𝔹^X ∈ 𝔹 do
    for ℙ_i ∈ ℙ do
        /* augment proofs of exclusive allotment with
           proofs of coin membership                  */
        𝔄_i^X ← 𝔄_i^X ∪ λ(𝔄^X ∈ 𝔄)
    end
end
return 𝔄
```

Accordingly, this requires the validation procedure of a $\mathfrak{A}_i^X$ to include the membership verification that the calculated root of $\mathfrak{A}^X \in \mathfrak{A}$, and that each coin X only has at most one $\mathfrak{A}^X$. This alternative reconstruction method is also to be employed for the construction and validation of the reserve collateral allocation commitment $\mathfrak{C}$.

C. Involved Participants

In the following we define the participants behaviour.

1) $\mathbb{V}_{\not\subset}$ Parent-chain Verifier: TEX allows elegantly to support all existing commit-chain operations of the previously defined $\mathbb{V}_{\not\subset}$ [40], with the addition of a coin parameter. Storage requirements of $\mathbb{V}_{\not\subset}$ are extended to store a copy of $\mathbb{B}_G^X$ for each managed coin. Our novel idea is to change $\mathbb{V}_{\not\subset}$'s commitment procedure, to accept a multiple coin commitment $\mathfrak{A}$ (cf. Section V-A) without executing any of the ascribed validations in [40] on the root allotment. The validation of the root allotment is then executed on every $\mathfrak{A}_i^X$ verification operation instead. Furthermore the $\mathfrak{A}_i^X$ verification is amended to validate that $\mathfrak{A}^X \in \mathfrak{A}$.

In the following, we present the more interesting third idea of a new swap challenge-response procedure. A $\mathbb{P}_i$ opens a new $\mathfrak{S}_i^X(e)$ against $\mathbb{O}_{\not\subset}$, who closes the challenge by proving that a swap that was requested by $\mathbb{P}_i$, was correctly enacted.

Algorithm 4: openSwapChallenge

```
Input  : 𝔄_i^X(e), update_i^X(e−1), 𝔛_i^{X,Y}(e−1),
         λ(𝔛_i^{X,Y}(e−1) ∈ T_i^X(e−1))
Output: 𝔖_i^X(e)
assert verifyProof(𝔄_i^X(e));
assert 𝔄_i^X(e) applies update_i^X(e−1);
assert λ(𝔛_i^{X,Y}(e−1) ∈ T_i^X(e−1)) is valid;
lowerLimit^X ← A_i^X(e−1) + R_i^X(e−1) + 𝔻_i^X(e−1) −
  S_i^X(e−1) − 𝕎_i^X(e−1)
undebited^X ← A_i^X(e) − lowerLimit^X
assert undebited^X ≥ 0
𝔖_i^X(e).debited^X ← 𝔛_i^{X,Y}(e−1).amount^X − undebited^X
return 𝔖_i^X(e)
```

In Algorithm 4, $\mathbb{V}_{\not\subset}$ assesses that a swap order was placed by $\mathbb{P}_i$, and calculates based on the latest committed state the amount of coin X that was debited from $\mathbb{P}_i$. Algorithm 5 defines how to calculate the amount of Y coins that is expected to be credited in favor of $\mathbb{P}_i$ in exchange for the amount that was proven to be debited in a $\mathfrak{S}_i^X(e)$, and close it if the price that was enacted matches the one specified in $\mathfrak{X}_i^{X,Y}(e-1)$.

Algorithm 5: closeSwapChallenge

```
Input  : 𝔖_i^X(e), 𝔄_i^Y(e), update_i^Y(e−1), 𝔛_i^{X,Y}(e−1),
         λ(𝔛_i^{X,Y}(e−1) ∈ T_i^Y(e−1))
assert verifyProof(𝔄_i^Y(e));
assert 𝔄_i^Y(e) applies update_i^Y(e−1);
assert λ(𝔛_i^{X,Y}(e−1) ∈ T_i^Y(e−1)) is valid;
lowerLimit^Y ← A_i^Y(e−1) + R_i^Y(e−1) + 𝔻_i^Y(e−1) −
  S_i^Y(e−1) − 𝕎_i^Y(e−1)
credited^Y ← A_i^Y(e) −lowerLimit^Y
assert credited^Y / 𝔖_i^X(e).debited^X ≈ 𝔛_i^{X,Y}(e−1).price
close 𝔖_i^X(e)
```

Algo. 4 and 5 can be combined into one non-interactive punishment method that relies on $\mathbb{P}_i$ providing both $\mathfrak{A}_i^X(e)$ and $\mathfrak{A}_i^Y(e)$ to show that the assertions in Algo. 5 are unsatisfied.

2) $\mathbb{O}_{\not\subset}$ Commit-chain Operator: TEX allows $\mathbb{O}_{\not\subset}$ to moderate transactions in different coin ledgers through maintaining

a separate commit-chain ledger $\mathbb{B}_L^X$ for each coin X that it chooses to support. $\mathbb{O}_{\not\subset}$ builds its periodic commitment as described in Section V-A. Lastly, we specify how $\mathbb{O}_{\not\subset}$ can support the enactment of swap requests from a $\mathbb{P}_i$ and prove their correct enactment in $\mathbb{V}_{\not\subset}$ by closing any opened $\mathfrak{S}_i^X$. In Algorithm 10 (cf. Appendix) we present the swap ratification procedure, that given the appropriate state update authorizations from a $\mathbb{P}_i$, $\mathbb{O}_{\not\subset}$ ratifies the requested swaps by revealing its own signature on the updates after validating the consistency of the state updates.

One interesting detail is that $\mathbb{O}_{\not\subset}$ expects two state update authorization messages for the swap, the first of which allows $\mathbb{O}_{\not\subset}$ to debit $\mathbb{P}_i$'s X coins, and the second of which does not explicitly force $\mathbb{O}_{\not\subset}$ to grant $\mathbb{P}_i$ any Y coins. Regardless of this odd setting, TEX constrains $\mathbb{O}_{\not\subset}$ to behave honestly and credit $\mathbb{P}_i$ Y coins in exchange for any X coins it debits, at the rate set by $\mathbb{P}_i$, due to the way the swap challenge $\mathfrak{S}_i^X$ can be closed in Algorithms 4 and 5 for $\mathbb{V}_{\not\subset}$.

3) $\mathbb{P}$ Users: Through TEX, a $\mathbb{P}_i$ is able to instantiate, use and maintain different accounts for each coin. Consequently, the $\mathfrak{A}_i^X(e)$ verification procedure is then employed to asses the received data from $\mathbb{O}_{\not\subset}$. Lastly, the corresponding methods of challenging a swap enactment by opening a new $\mathfrak{S}_i^X$ in $\mathbb{V}_{\not\subset}$ and requesting a swap from $\mathbb{O}_{\not\subset}$ are added.

D. Provably Consistent Settlement Checkpoints

In this section we present verification procedures for a non-interactive ZKP environment that supports combining proofs using recursive composition to allow $\mathbb{V}_{\not\subset}$ to verify the complete consistency of the trades in a checkpoint $\mathfrak{A}(e)$.

Algorithm 6: verifyTradeEnactment

Verifier Input : $\mathfrak{A}(e)$, $\Theta^{X,Y}$
Prover Input : $\mathfrak{x}_i^{X,Y}$, $\mathfrak{A}_i^X(e)$, $\mathfrak{A}_i^Y(e)$, update_i^X, update_i^Y, $\lambda(\mathfrak{x}_i^{X,Y}(e-1) \in \text{update}_i^X.T)$, $\lambda(\mathfrak{x}_i^{X,Y}(e-1) \in \text{update}_i^Y.T)$, $\lambda(\mathfrak{x}_i^{X,Y} \in \Theta^{X,Y})$
verify $\lambda(\mathfrak{x}_i^{X,Y}(e-1) \in \text{update}_i^Y.T)$;
verify $\lambda(\mathfrak{x}_i^{X,Y}(e-1) \in \text{update}_i^X.T)$;
verify update_i^Y is applied in $\mathfrak{A}_i^Y(e)$;
verify update_i^X is applied in $\mathfrak{A}_i^X(e)$;
verify $\mathfrak{A}_i^Y(e)$ leads to $\mathfrak{A}(e)$;
verify $\mathfrak{A}_i^X(e)$ leads to $\mathfrak{A}(e)$;
if $\mathfrak{x}_i^{X,Y}$ *is not finalized* **then**
  verify credited and debited amounts in $\mathfrak{A}_i^X(e)$ and $\mathfrak{A}_i^Y(e)$ match the price in $\mathfrak{x}_i^{X,Y}$;
end
verify $\lambda(\mathfrak{x}_i^{X,Y} \in \Theta^{X,Y})$;
return *hash*($\mathfrak{x}_i^{X,Y}$), $\mathfrak{x}_i^{X,Y}$.*debited_amount*

Algorithm 6 ensures that a single trade $\mathfrak{x}_i^{X,Y}$ enactment is correctly enforced in $\mathfrak{A}(e)$. To verify that a larger set of trades is consistently enacted, we apply the methods of recursively combining proofs [40]. The bottom-level of the transfer delivery combiner is augmented to accept either a delivery proof or a trade enactment proof. In the ZKP system model, $\mathbb{O}_{\not\subset}$ can require only one signature from $\mathbb{P}_i$ to enact a trade $\mathfrak{x}_i^{X,Y}$, and multiple ongoing trades can be performed in parallel by a single $\mathbb{P}_i$. This is achieved through $\mathbb{P}_i$ authorizing the debit of the full amount in update_i^X and in return expecting two passively delivered transfers. The first transfer would credit back the remaining unmatched X-coin amount in an order, and the second transfer would credit the matched Y-coin amount. The respective ZKP validation procedures then ensure the correct execution of these transfers by $\mathbb{O}_{\not\subset}$ without the involvement of $\mathbb{P}_i$. This procedure would be very similar to Algorithms 6 and we leave it as an exercise to the reader.

E. Provably Consistent Matching Settlement

When utilizing both moonwalk orders and provably consistent settlement, the changes enacted in the settlement layer need to be linked to the order matches in the order book, s.t. all the amounts credited and debited during settlement correspond to the amounts matched in the order book. In this section we describe a set of recursively composable procedures to prove this correspondence.

The combiner of Algorithm 6 is first required to aggregate the debited amounts from each child, s.t. its root combiner provably calculates the total debited volume of the placed orders from one $\mathbb{P}_i$, and then the account integrity verification procedure [40] is also expected to carry out the aggregation of this value over a token $\mathfrak{A}^X$ sub-tree. We introduce in Algorithm 9 (cf. Appendix) a procedure for verifying that the matched amounts in an an order book commitment Θ correspond to the settled amounts in a $\mathfrak{A}^X$. This procedure can then be combined recursively to validate a full $\mathfrak{A}$.

VI. Security Analysis

We analyze the security guarantees of TEX assuming that the underlying layer $\mathbb{BC}$ serves as a recourse for settling disputes on the integrity of $\not\subset$. We assume costs associated with $\mathbb{BC}$ settlement (e.g. transaction fees) as external expenses to the balance of a $\mathbb{P}_i$ in TEX. Given the commit-chain properties, these expenses are small (cf. Section V-B). TEX is designed to prevent any honest member of $\mathbb{P}$ from losing any assets despite a strong set of the following adversarial capabilities.

A. Threat Model

Further formalizing the attacker model from Section III-E, we assume two classes of users in TEX: *(1)* $\mathbb{O}_{\not\subset}$ operators and *(2)* $\mathbb{P}$ participants. We assume the existence of an irrational adversary willing to sustain financial losses to cause honest parties to lose some or all of their funds in $\not\subset$. This adversary may seize control of $\mathbb{O}_{\not\subset}$, some or all but one of $\mathbb{P}$, or a combination thereof, to attack an honest $\mathbb{P}_i$ not under its control. The adversary has full control of the identities associated with the compromised parties and may authorize any messages on their behalf or front-run any user input, but cannot violate the integrity of the honest users' identities. We consider the adversary to control all network communication between traders, and between the trader and the operator (Dolev-Yao [48]), but may not compromise an honest $\mathbb{P}_i$'s communication with $\mathbb{BC}$, respectively $\mathbb{V}_{\not\subset}$. We assume that

the adversary is incapable of causing the underlying ledger layer $\mathbb{BC}$ to malfunction or misbehave. We define malicious behavior as that which aims to cause an honest $\mathbb{P}_i$ to lose control of some or all of its funds in $\not\subset$ or cause an honest $\mathbb{O}_{\not\subset}$ to be forcibly shut down by $\mathbb{V}_{\not\subset}$. We assume that a $\mathbb{P}_i$ provides moonwalk orders to $\mathbb{O}_{\not\subset}$ through an anonymous communication channel that does not leak identities.

B. Order Book Guarantees

1) Order Secrecy: The security proof of the timed commitments used to hide the contents of $\mathfrak{X}_i$ is presented in [45]. In this section we prove that a moonwalk order does not reveal its issuer $\mathbb{P}_i$ without possession of K_j. Assuming,

1) $\mathbb{O}_{\not\subset}$ knows K_{j-1} and is given a new moonwalk order with: n, C_K, C_M, H($\mathfrak{X}_i$), H($K_j \| \mathbb{P}_i$), π_1 and π_2.
2) H is a deterministic collision resistant hash function.

If the adversary is able to compute K_j from K_{j-1}, or $\mathfrak{X}_i$ from H($\mathfrak{X}_i$), or calculate any x s.t. H($x \| \mathbb{P}_j$) = H($K_j \| \mathbb{P}_i$) then the second assumption is violated. A $\mathbb{O}_{\not\subset}$ capable of extracting the prover witness inputs used to generate π_1 or π_2 can moreover learn the order details. ■

Note that the anonymity of $\mathbb{P}_i$ does not only rely on the cryptographic secrecy of the order. Unidentifiable communication channels need to be used, such as random IP addresses and side-channel free browsing/OS environments. Moreover, $\mathbb{P}_i$ would have to randomize the number of dummy orders it plans to submit prior to submitting its real order, and randomize the time intervals between submissions to avoid generating a predictable pattern. The size of the honest participants submitting orders to $\mathbb{O}_{\not\subset}$ also determines the likelihood that $\mathbb{O}_{\not\subset}$ is able to randomly guess the owner of a $\mathfrak{X}_i$.

2) Immutable Orderbook History: In this section we prove that a malicious $\mathbb{O}_{\not\subset}$ may not successfully alter the history of the order book after providing a signed receipt to a $\mathbb{P}_i$ who posts a $\mathfrak{X}_i^{X,Y}$ and not be halted within the next *eon*. Assume that $\mathbb{O}_{\not\subset}$ causes a mutation in the order book that invalidates a receipt given to some $\mathbb{P}_i$ by altering some order data that precedes $\mathbb{P}_i$'s $\mathfrak{X}_i$. This will prevent $\mathbb{O}_{\not\subset}$ from being able to close $\mathbb{P}_i$'s $\mathfrak{F}_i^X$ because this mutation will lead to the root order book commitment Θ containing a different view of the orders that precede $\mathfrak{X}_i$, and therefore, different values of the roots of the merkle mountain range comprising the set of orders prior to $\mathfrak{X}_i$. As the running hash of all left siblings in $\lambda(\mathfrak{X}_i \in \Theta(e)^{X,Y})$ will not match the one signed in the receipt, $\mathbb{O}_{\not\subset}$ will not be able to close the challenge. ■

3) Consistent Order Matching: Algorithms 7 and 8 incrementally transition a commitment to the order book through adding a match to the order book or through appending a new order to the end. We present a proof by contradiction that no transition $\Theta_m^k \rightarrow \Theta_{m+1}^k$ or $\Theta_m^k \rightarrow \Theta_m^{k+1}$ violates any of the constraints in Section IV-E.

Let $\Theta_m^k \rightarrow \Theta_{m+1}^k$ be a correct transition accepted by Algorithm 7 that violates the first constraint when matching a $\mathfrak{X}_i^{X,Y}$, s.t. there exists some $\mathfrak{X}_k^{Y,X}$ ordered before $\mathfrak{X}_j^{Y,X}$ that can match $\mathfrak{X}_i^{X,Y}$. Let t_u be the least common ancestor of $\mathfrak{X}_k^{Y,X}$ and $\mathfrak{X}_j^{Y,X}$ in Θ_m^k with t_p and t_q as its direct children s.t. t_p is an ancestor of $\mathfrak{X}_k^{Y,X}$, and t_q of $\mathfrak{X}_j^{Y,X}$, then t_p is part of $\lambda(\mathfrak{X}_j^{X,Y} \in \Theta_m^k)$, and t_q is calculated as its sibling when validating this path. This will lead to the failure of the assertion in Algorithm 7 of t_q being the first child that can match $\mathfrak{X}_i^{X,Y}$, which violates our assumption that this is a valid state transition. Algorithm 8 requires that the intermediate transitions $\Theta_m^k \rightarrow \Theta_{m+x}^k$ are accepted by the Algorithm 7 root combiner before accepting the transition $\Theta_m^k \rightarrow \Theta_m^{k+1}$.

Violating the second constraint is trivially not possible due to the matching directionality of Algorithm 7, where the input match is automatically counted as being from the last order to a previous one, and Algorithm 7 may only append the last order to the commitment.

Lastly, let $\Theta_m^k \rightarrow \Theta_{m+x}^{k+1}$ be a correct transition accepted by Algorithm 8 that violates the third constraint, where $\mathfrak{X}_i^{X,Y}$ retains a positive remaining volume. Therefore, there exists some $\mathfrak{X}_j^{Y,X} \in \Theta_{m+x}^k$ that can match with $\mathfrak{X}_i^{X,Y}$, and one of the last two assertions in Algorithm 8 will fail, violating the assumption that this is a correct state transition. ■

4) Front-Running Resilience: In this section we informally argue that our scheme demotes $\mathbb{O}_{\not\subset}$ from being able to profit from risk-free front-running operations to a risk bearing participant, and that it provides an indicator to a potentially ill-affected $\mathbb{P}_i$ that $\mathbb{O}_{\not\subset}$ attempted to front-run its order.

The core premise behind the front-running resilience of a moon-walk order is that $\mathbb{O}_{\not\subset}$ cannot immediately infer the contents of a $\mathfrak{X}_i$ or link it to $\mathbb{P}_i$. This means that if $\mathbb{O}_{\not\subset}$ wants to insert risk-free front-running trades into the order book, it needs to decrypt the contents of incoming orders before committing to their receipt. If $\mathbb{O}_{\not\subset}$ does not attempt to prematurely learn the contents of incoming orders then any front-running it performs is a risk-bearing decision, as the strict matching algorithm may not mutate the order book in its favor later on.

During normal operation, where the participants $\mathbb{P}$ mix in zero-value orders with their regular orders, an adversarial $\mathbb{O}_{\not\subset}$ wishing to decrypt and front-run a real order would have to predict in advance which incoming order has value. By making a wrong prediction, it would dissuade the affected participant from submitting its orders in the future. As the number of users grows, making a successful prediction becomes proportionally harder. The expected reward from front-running a single order also becomes uncertain if the operator chooses to continue appending other orders while it decrypts the targeted orders.

The more orders $\mathbb{O}_{\not\subset}$ decides to target for front-running in parallel, the more traders it loses reputation with, and the more computation it has to invest to reveal the orders' contents, which may not be profitable. More formally, the probability of a randomly selected moon-walk order not being a zero-valued order is equal to $\frac{1}{k}$ when each $\mathbb{P}_i$ submits on average $k-1$ zero-valued orders around real ones.

C. Commit-Chain Settlement Guarantees

We prove now prove commit-chain security guarantees.

1) Account Integrity: For TEX, the proofs presented in NOCUST [40] for the exclusivity of balance allotment, the guarantee of balance custody, and secure off-chain registration remain applicable to TEX as they are for each coin X. In this section, we present the more significant proofs for guarantees that are affected by the different composition of TEX.

2) Double-Spend Futility: In TEX, no validation at all is performed on the submitted checkpoint $\mathfrak{A}$, since it is a commitment to multiple independent coin allotment commitments $\mathfrak{A}^X$ for each managed coin X. Therefore, the assumption in NOCUST that $\mathbb{V}_{\not\subset}$ would reject the checkpoint submission due to a miscalculated allotment_{root} no longer holds. Instead, we leverage the fact that the verification procedure in $\mathbb{V}_{\not\subset}$ is now performing this validation step for every proof, and thus an incorrect $\text{allotment}^X_{root}$ value for any $\mathfrak{A}^X \in \mathfrak{A}$ would lead to the rejection of $\mathfrak{A}^X_i$ for any $\mathbb{P}_i$, rendering $\mathfrak{A}^X$ unverifiable and thus unusable for challenge responses, which is sufficient to ensure a halt under misbehavior. ■

3) Operational Integrity: We now demonstrate how $\mathbb{O}_{\not\subset}$ proves its operational integrity while accounting for swap enactments. An honest $\mathbb{O}_{\not\subset}$ maintains functionality under a subset of malicious users in $\mathbb{P}$, and a dishonest $\mathbb{O}_{\not\subset}$ leads to $\not\subset$ being stopped. We demonstrate a proof by case analysis, where we model the provability of a $\mathbb{O}_{\not\subset}$'s integrity as a finite state machine whereby transfers and swaps are performed by $\mathbb{O}_{\not\subset}$ during e and committed during $e+1$ in $\mathfrak{A}(e+1)$. A server is defined as maintaining provable integrity during *eon* e so long as it is able to close any challenge using $\mathbb{V}_{\not\subset}$.

- $\mathbf{s_0} \rightarrow \mathbf{s_8}$: Given no interactions between $\mathbb{O}_{\not\subset}$ and $\mathbb{P}_i$ during e, an honest $\mathbb{O}_{\not\subset}$ may submit a $\mathfrak{A}$ to $\mathbb{V}_{\not\subset}$ that does not apply any update to $\mathbb{P}_i$'s balance.
- $\mathbf{s_0} \rightarrow \mathbf{s_1} \rightarrow \mathbf{s_8}$: Given only $\text{update}_i(e)$ signed by $\mathbb{P}_i$, but no $\text{update}_j(e)$ signed by $\mathbb{P}_j$, an honest $\mathbb{O}_{\not\subset}$ may discard $\mathfrak{T}_{i,j}$. No $\mathbb{X}^d_i(e+1)$ may be opened as $\mathbb{P}_i$ and $\mathbb{P}_j$ would not possess an $\text{update}_i(e)$ signed by $\mathbb{O}_{\not\subset}$ containing $\mathfrak{T}_{i,j}(e)$.
- $\mathbf{s_0} \rightarrow \mathbf{s_1} \rightarrow \mathbf{s_2} \rightarrow \mathbf{s_8}$: Given an $\text{update}_i(e)$ signed by $\mathbb{P}_i$ and an $\text{update}_j(e)$ signed by $\mathbb{P}_j$, an honest $\mathbb{O}_{\not\subset}$ may discard, or ratify and include $\mathfrak{T}_{i,j}(e)$.
- $\mathbf{s_0} \rightarrow \mathbf{s_1} \rightarrow \mathbf{s_2} \rightarrow \mathbf{s_3} \rightarrow \mathbf{s_8}$: Given an $\text{update}_i(e)$ signed by $\mathbb{P}_i$ and an $\text{update}_j(e)$ signed by $\mathbb{P}_j$, an honest $\mathbb{O}_{\not\subset}$ must synchronize $\mathfrak{T}_{i,j}(e)$'s delivery in $\mathfrak{A}(e+1)$ revealing a countersigned update to $\mathbb{P}_i$ and/or $\mathbb{P}_j$.
- $\mathbf{s_0} \rightarrow \mathbf{s_4} \rightarrow \mathbf{s_8}$: Given update^X_i and update^Y_i from $\mathbb{P}_i$, an honest $\mathbb{O}_{\not\subset}$ may discard, or ratify and include $\mathfrak{X}^{X,Y}_i(e)$.
- $\mathbf{s_0} \rightarrow \mathbf{s_4} \rightarrow \mathbf{s_5} \rightarrow \mathbf{s_8}$: Given the required update^X_i and update^Y_i from $\mathbb{P}_i$, an honest $\mathbb{O}_{\not\subset}$ must include $\mathfrak{X}^{X,Y}_i(e)$ in $\mathfrak{A}(e+1)$ after revealing its ratification to $\mathbb{P}_i$.
- $\mathbf{s_8} \rightarrow \mathbf{s_8}$: While in a state of provable integrity, $\mathbb{O}_{\not\subset}$ is able to cancel any malicious $\mathbb{W}^X_i$ initiated by $\mathbb{P}_i$ as it retains the necessary inputs to prove the overdraw to $\mathbb{V}_{\not\subset}$.

While in a state of provable integrity, $\mathbb{O}_{\not\subset}$ can justifiably cancel any malicious withdrawal by a $\mathbb{P}_i$ using $\mathbb{V}_{\not\subset}$, and can close any open challenge, as long as it correctly constructs $\mathfrak{A}$ because it retains knowledge of all the required inputs to satisfy $\mathbb{V}_{\not\subset}$'s challenge closure procedures. However, a dishonest server trying to debit a $\mathbb{P}_i$ without authorization, or without crediting $\mathbb{P}_j$ in case of a $\mathfrak{T}_{i,j}$ (or crediting sufficient Y tokens in case of a $\mathfrak{X}^{X,Y}_i$), may not find itself in a state of provable integrity in $e+1$.

- $\mathbf{s_0} \rightarrow \mathbf{s_7}$: Given no interactions between $\mathbb{O}_{\not\subset}$ and $\mathbb{P}_i$ during e, the hub cannot construct a valid $\mathfrak{A}(e+1)$ containing an $\text{update}_i(e)$ signed by $\mathbb{P}_i$. As $\mathbb{O}_{\not\subset}$ cannot forge $\mathbb{P}_i$'s signature, it cannot close a $\mathbb{X}^b_i(e+1)$.
- $\mathbf{s_0} \rightarrow \mathbf{s_1} \rightarrow \mathbf{s_7}$: Given only an $\text{update}_i(e)$ signed by $\mathbb{P}_i$, the hub cannot construct a valid $\mathfrak{A}(e+1)$ containing an $\text{update}_j(e)$ signed by $\mathbb{P}_j$. A $\mathbb{X}^d_i(e+1)$ on $\mathfrak{T}_{i,j}(e)$ by a custodian $\mathbb{P}_i$ is not closeable by $\mathbb{O}_{\not\subset}$.
- $\mathbf{s_0} \rightarrow \mathbf{s_1} \rightarrow \mathbf{s_2} \rightarrow \mathbf{s_3} \rightarrow \mathbf{s_7}$: Once the hub delivers a countersigned $\text{update}_i(e)$ and/or $\text{update}_j(e)$ to either $\mathbb{P}_i$ or $\mathbb{P}_j$ respectively, it may not back out of enforcing $\mathfrak{T}_{i,j}(e)$, as $\mathbb{O}_{\not\subset}$ not able to close a $\mathbb{X}^b_i(e+1)$, and/or $\mathbb{X}^b_j(e+1)$, if it commits an outdated state in $\mathfrak{A}(e+1)$.
- $\mathbf{s_0} \rightarrow \mathbf{s_4} \rightarrow \mathbf{s_6}$: Given update^X_i and update^Y_i from $\mathbb{P}_i$, a dishonest $\mathbb{O}_{\not\subset}$ cannot construct a correct instant of $\mathfrak{A}(e+1)$ without including update^Y_i and correctly applying the balance changes. A $\mathfrak{S}^X_i(e+1)$ by a custodian $\mathbb{P}_i$ is not closeable by $\mathbb{O}_{\not\subset}$.
- $\mathbf{s_0} \rightarrow \mathbf{s_4} \rightarrow \mathbf{s_5} \rightarrow \mathbf{s_6}$: Once the hub reveals its ratification of update^X_i to $\mathbb{P}_i$, it cannot back out of including $\mathfrak{X}^{X,Y}_i$ in $\mathfrak{A}(e+1)$, as $\mathbb{O}_{\not\subset}$ is not able to close a $\mathfrak{B}^X_i(e+1)$, and/or a $\mathfrak{S}^X_i(e+1)$ if it did not include update^Y_i in $\mathfrak{A}(e+1)$.■

4) Volatility Protection: If a $\not\subset$ fails, a $\mathbb{P}_i$ can reclaim the potential change in coin value that could have occurred during the past two *eons*, up to a certain predefined percentage — which we refer to as volatility protection. This calculation utilizes the fact that halting $\not\subset$ also implies reversing the debit of the swapped coins, not just the credit, when the swaps are performed within the same chain. We proceed to prove how a $\mathbb{P}_i$ enacting swaps in a $\not\subset$ instance is guaranteed to be able to finalize receipt of the gained value from its trades up to a known total amount, regardless of the adversary's behavior while controlling $\mathbb{O}_{\not\subset}$ and/or all other members of $\mathbb{P}$.

$$Rec(e) = Value(C_i(e) + \mathbb{T}_i(e-1) + \mathbb{T}_i(e)) + \sum_{\mathfrak{X} \in T_i(e) \cup T_i(e-1)} Value(\mathfrak{X}) \tag{2}$$

$$Rec(e+1) = Value(C_i(e+1) + \mathbb{T}_i(e)) + \sum_{\mathfrak{X} \in T_i(e)} Value(\mathfrak{X}) \tag{3}$$

$$\sum_{\mathfrak{X} \in T_i(e) \cup T_i(e-1)} \Delta Value(\mathfrak{X}) \leq Rec(e) \tag{4}$$

$$\sum_{\mathfrak{X} \in T_i(e)} \Delta Value(\mathfrak{X}) \leq Rec(e+1) \tag{5}$$

$Value$ is defined as the amount of a coin multiplied by its price in terms of an arbitrary unit chosen by $\mathbb{P}_i$. The change in valuation, denoted as $\Delta Value$, is assumed to be continuously calculated by $\mathbb{P}_i$ using ongoing market prices as comparison points to the originally set prices.

a) Proof: The proof that a $\mathbb{P}_i$ observing the constraints in Equation 4 and 5 has protection against the volatility of its coin portfolio held within $\mathbb{O}_{\not\subset}$ is demonstrated as follows.

- If the $\not\subset$ instance fails during *eon* e, then $\mathbb{P}_i$ can recover an amount equal to the R.H.S of Equation 4.
- If the $\not\subset$ instance fails during *eon* $e+1$, then $\mathbb{P}_i$ can recover an amount equal to the R.H.S of Equation 5.
- Otherwise, $\mathfrak{T}_{j,i}(e)$ has been in included in $\mathfrak{A}(e+1)$, and its amount can be withdrawn in $e+2$.

In the first two cases, $\mathbb{P}_i$ recovers the originally held value prior to swapping[1] plus the value gained through holding the swap position in the form of the value of the collateral exclusively allocated by $\mathbb{O}_{\not\subset}$ to $\mathbb{P}_i$. $\mathbb{T}_i(e-1)$ and $\mathbb{T}_i(e)$ are accounted for by $\mathbb{V}_{\not\subset}$, while $C_i(e)$ and $C_i(e+1)$ are committed to by $\mathbb{O}_{\not\subset}$ and learned by $\mathbb{P}_i$ before *eon* e commences, or assumed to be zero. The exclusivity of the amounts $C_i(e)$ and $C_i(e+1)$ are guaranteed through validation of $\mathfrak{C}_i(e-2)$ and $\mathfrak{C}_i(e-1)$ respectively. Moreover, the recoverable amount from a $\mathfrak{C}_i(e-2)$ is always available in *eon* e. This is because $\mathbb{O}_{\not\subset}$ cannot withdraw staked collateral s.t. the total staked amount $\mathbb{C}(e)$ in $\mathfrak{C}(e-2)$ is unavailable for recovery.■

VII. Evaluation

In the following section, we evaluate the TEX settlement.

A. TEX Commit-Chain Evaluation

Deploying the TEX settlement smart contract (3024 LOC Solidity) on Ethereum[2] costs 17.8M gas (13.38 USD[3]). TEX follows a 36 hour *eon* interval. $\mathbb{O}_{\not\subset}$, implemented in Python (1448 LOC), operates on a dual-core CPU with 4GB RAM and SSD HDD. We deployed a parity node as blockchain source. Users can interact and challenge $\mathbb{O}_{\not\subset}$ through a JavaScript library (12096 LOC). Table II outlines the parent-chain costs.

Operation	Paid by	Gas	USD	Complexity
Checkpoint (every *eon*)	Exchange	96'073	0.072	$\mathcal{O}(1)$
Deposit	Trader	64'720	0.048	$\mathcal{O}(1)$
Withdraw	Trader	169'238	0.126	$\mathcal{O}(\log n)$
Initiate State Challenge	Trader	281'686	0.211	$\mathcal{O}(\log n)$
Answer State Challenge	Exchange	80'769	0.061	$\mathcal{O}(\log n)$
Initiate Swap Challenge	Trader	318'195	0.239	$\mathcal{O}(\log n + \log v)$
Answer Swap Challenge	Exchange	76'143	0.057	$\mathcal{O}(\log n + \log v)$

TABLE II: Costs for a TEX operator and it's traders. n represent the number of users and v the number of swaps.

1) Storage Costs and Performance:

Parent-Chain Storage As long as a trader does not perform any withdraw, deposit or challenge with the TEX smart contract, the only parent-chain footprint is the constant-sized checkpoint hash that compresses all of the user's balances — allowing TEX to scale.

User Storage Users store at least their transfers of the last two *eons* (312 bytes per transfer) and query each *eon* their individual Merkle proof from $\mathbb{O}_{\not\subset}$ (1984 bytes at 1B users and four coins in the coin tree).

Operator Storage The operator stores all trades of the last two rounds, all account states and their Merkle proofs.

Trade Throughput Our non-optimized implementation achieves about 10 trades per second between two accounts on a single CPU core, without network latency. This performance scales out with the number of accounts.

[1]When spent from A_i or C_i

[2]cf. address 0x6B9f10931E88349A572F2f0883E49528902B4b5D

[3]Assuming a gas price of 5 Gwei and Ether price of 150 USD.

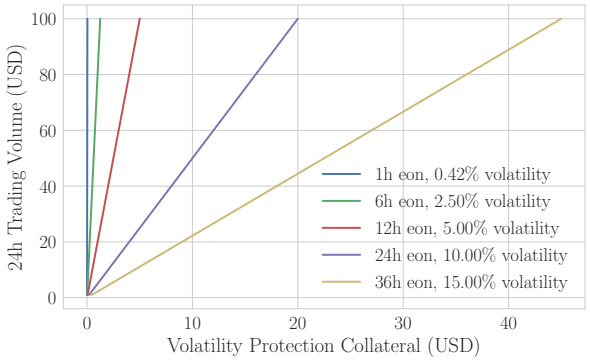

Fig. 5: Volatility protection collateral by TEX for instant trade finality, assuming a max. price volatility of 10% within 24 hours between two coins X,Y.

2) Instant Exchange Collateral Costs: The collateral requirements of TEX for instant trade finality equals to the trade volume of the last two *eons*, times the price difference of the traded assets within the last two eons. Assuming an *eon* interval of 12 hours, a daily trade volume of 300M USD between a pair of cryptocurrencies, and a currency fluctuation of 10% between this pair, the collateral amounts to $300 \times 0.1 = 30$M USD (cf. Figure 5).

VIII. Conclusion

Alarmingly, malicious crypto-currency exchange operators, miners and adversarial traders can currently perform highly rewarding market manipulations, without significant risks of being caught due to missing legal frameworks to protect traders. Moreover, the honest behaviour of custodian exchanges currently is only enforced through manual regulatory audits, instead of being held accountable through cryptographic and non-repudiable supervision.

TEX allows the operation of either a custodian exchange which raises the bar for operational accountability, or a non-custodial, blockchain-based exchange that can scale to trade loads similar to custodian exchanges. Maleficence by the TEX operator, i.e. front-running of orders, can be transparently uncovered via secure cryptographic means.

We hope our work spurs further efforts on provable transparency and accountability of financial exchanges which remain at the core of our worldwide economy.

IX. Acknowledgements

This work is partially funded by the Imperial College London President's PhD Scholarship.

References

[1] Foreign exchange manipulation: Finma issues six industry bans, 2019. https://www.finma.ch/en/news/2015/12/20151217-mm-devisenhandel/.

[2] Nasdaq glossary, 2019. https://www.nasdaq.com/investing/glossary/f/front-running.

[3] Idex - decentralized ethereum asset exchange, 2019. https://idex.market/.

[4] Oasis dex - wiki, 2019. https://github.com/OasisDEX/oasis/wiki.

[5] Top dex pie chart, 2018. https://etherscan.io/stat/dextracker.

[6] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 3–16. ACM, 2016.

[7] Binance api documentation, 2019. https://github.com/binance-exchange/binance-official-api-docs/blob/master/rest-api.md.

[8] Blockchain transparency report, 2019. https://www.blockchaintransparency.org.

[9] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. 1996.

[10] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *Journal of cryptology*, 1(2):77–94, 1988.

[11] Investopedia - order book, 2019. https://www.investopedia.com/terms/o/order-book.asp.

[12] Sec glossary, 2019. https://www.sec.gov/fast-answers/answerslimithtm.html.

[13] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[14] Cynthia Dwork and Moni Naor. *Pricing via Processing or Combatting Junk Mail*, pages 139–147. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993.

[15] Adam Back. Hashcash - a denial of service countermeasure. Technical report, 2002.

[16] What are zk-snarks?, 2019. https://z.cash/technology/zksnarks.

[17] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334. IEEE, 2018.

[18] Parity transaction fee order preference, 2019. https://wiki.parity.io/Configuring-Parity-Ethereum.html.

[19] geth transaction fee order preference, 2019. https://github.com/ethereum/go-ethereum/blob/290e851f57f5d27a1d5f0f7ad784c836e017c337/core/types/transaction.go#L372.

[20] Frontrun.me - visualizing ethereum gas auctions, 2019. http://frontrun.me/.

[21] Implementing ethereum trading front-runs on the bancor exchange in python, 2019. https://hackernoon.com/front-running-bancor-in-150-lines-of-python-with-ethereum-api-d5e2bfd0d798.

[22] Blockchain frontrunning - swende.se, 2019. http://swende.se/blog/Frontrunning.html.

[23] Libsubmarine - to sink frontrunners, send in the submarines, 2019. http://hackingdistributed.com/2017/08/28/submarine-sends/.

[24] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 326–349. ACM, 2012.

[25] Ralph C. Merkle. *A Digital Signature Based on a Conventional Encryption Function*, pages 369–378. Springer Berlin Heidelberg, Berlin, Heidelberg, 1988.

[26] Merkle mountain ranges, 2019. https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md.

[27] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 279–296. USENIX Association, 2016.

[28] Loi Luu, Viswesh Narayanan, Kunal Baweja, Chaodong Zheng, Seth Gilbert, and Prateek Saxena. Scp: A computationally-scalable byzantine consensus protocol for blockchains. *IACR Cryptology ePrint Archive*, 2015:1168, 2015.

[29] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model, 2016.

[30] Ittay Eyal, Adem Efe Gencer, Emin Gun Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 45–59. USENIX Association, 2016.

[31] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30. ACM, 2016.

[32] Adem Efe Gencer, Robbert van Renesse, and Emin Gün Sirer. Service-oriented sharding with aspen. *arXiv preprint arXiv:1611.06816*, 2016.

[33] Pavel Prihodko, Slava Zhigulin, Mykola Sahno, Aleksei Ostrovskiy, and Olaoluwa Osuntokun. Flare: An approach to routing in lightning network. 2016.

[34] Andrew Miller, Iddo Bentov, Ranjit Kumaresan, and Patrick McCorry. Sprites: Payment channels that go faster than lightning. *arXiv preprint arXiv:1702.05812*, 2017.

[35] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2015.

[36] Rami Khalil and Arthur Gervais. Revive: Rebalancing

$\mathbb{BC}$	An operational blockchain.
$\not\subset$	An instance of TEX.
$\mathbb{O}_{\not\subset}$	The $\not\subset$ operator server.
$\mathbb{V}_{\not\subset}$	The $\not\subset$ parent-chain verifier smart-contract.
$\mathbb{P}$	The set of participants in $\not\subset$.
$\mathbb{P}_i$	A single participant $\mathbb{P}_i \in \mathbb{P}$.
e	The current *eon* (round).
$\mathbb{B}$	The commit-chain account-based ledger.
$\mathbb{B}_G$	The global parent-chain component of $\mathbb{B}$.
$\mathbb{B}_L$	The local commit-chain component of $\mathbb{B}$.
A_i	Initially allotted balance of $\mathbb{P}_i$.
R_i	The total received on the commit-chain by $\mathbb{P}_i$.
S_i	The total sent on the commit-chain by $\mathbb{P}_i$.
T_i	The commit-chain transactions involving $\mathbb{P}_i$.
C_i	The reserve collateral allocated for $\mathbb{P}_i$.
$\mathbb{D}_i$	The total deposited by $\mathbb{P}_i$.
$\mathbb{W}_i$	The total requested for withdrawal by $\mathbb{P}_i$.
$\mathfrak{T}_{i,j}$	A transfer from $\mathbb{P}_i$ to $\mathbb{P}_j$
$\mathfrak{X}_i$	A swap by $\mathbb{P}_i$.
$\mathfrak{B}_i$	A challenge of $\mathbb{P}_i$'s balance integrity.
$\mathfrak{E}_i$	A challenge of the delivery of a transfer.
$\mathfrak{S}_i$	A challenge of a swap by $\mathbb{P}_i$.
$\mathfrak{A}$	An exclusive balance allotment tree.
$\mathfrak{C}$	An exclusive collateral allocation tree.
$\mathfrak{A}_i$	A proof of exclusive balance allotment for $\mathbb{P}_i$.
$\mathfrak{C}_i$	A proof of exclusive collateral allocation for $\mathbb{P}_i$.
K_0	Random seed value for the delayed key disclosure setup.

TABLE III: A summary of notation used in TEX. **Any symbol can be parameterized to refer to its value for a specific coin and/or a specific *eon*.** For example, $A_i^X(e)$ would denote the initially allotted *X-coin* balance of $\mathbb{P}_i$ *at* eon e.

off-blockchain payment networks. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[37] Bitcoinj. https://bitcoinj.github.io/working-with-micropayments.

[38] Conrad Burchert, Christian Decker, and Roger Wattenhofer. Scalable funding of bitcoin micropayment channel networks. *Royal Society open science*, 5(8):180089, 2018.

[39] Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. *White paper*, 2017.

[40] Rami Khalil and Arthur Gervais. Nocust–a non-custodial 2 nd-layer financial intermediary. Technical report, Cryptology ePrint Archive, Report 2018/642. https://eprint. iacr. org/2018/642, 2018.

[41] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. Settling payments fast and private: Efficient decentralized routing for path-based transactions. *arXiv preprint arXiv:1709.05748*, 2017.

[42] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. Concurrency and privacy with payment-channel networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 455–471. ACM, 2017.

[43] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling blockchain innovations with pegged sidechains. *URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains*, 2014.

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

[45] Dan Boneh and Moni Naor. Timed commitments. In *Annual International Cryptology Conference*, pages 236–254. Springer, 2000.

[46] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology—CRYPTO'86*, pages 186–194. Springer, 1986.

[47] Lenore Blum, Manuel Blum, and Mike Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on computing*, 15(2):364–383, 1986.

[48] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.

[49] Washtrading definition, 2019. https://www.investopedia.com/terms/w/washtrading.asp.

APPENDIX A
NOTATION

Table III provides an overview of the notation in this paper.

APPENDIX B
WASH TRADING

A trader might perform wash trading by purchasing and selling a financial asset, e.g. a crypto-currency coin, for the sole purpose to feed the market with misleading information [49]. While wash trading is illegal for security assets in e.g. the US, it currently is unregulated for many crypto-currency coins. There are increasingly efforts to provide more transparency on such malpractices [8], while missing identities for blockchain addresses clearly hinder the practical detection of wash trading on anonymous decentralized exchanges.

A. Implications of Moonwalk Order Books on Wash Trading

Wash trading is challenging to detect on order books without enforcable order sequence and exchanges with missing trader identification procedures (e.g. know-your-customer verifications). That is because the exchange operator can collude with a wash trader and match the sell and buy order of the same entity, s.t. the wash trader is unlikely to suffer from financial losses, if e.g. his order were matched with other accounts. An exchange has an incentive to support wash trading to increase its apparent trading volume.

Given TEX's moonwalk order submission model, a wash trader is required to repeatedly provide buy and sell orders with the least possible time delay to increase the likelihood of matching with its own orders. Such pattern are possibly identifyable and could potentially deter wash traders.

Algorithm 8: verifyOrder

Verifier Input : Θ^k_m,, $\mathfrak{A}(e)$, $\mathfrak{X}^{X,Y}_i$
Prover Input : $\{\theta^k_{m+x}(0)...\theta^k_{m+x}(n)\}$, π, Θ^k_{m+x}, $\mathfrak{A}^X_i(e)$, $\text{update}^X_i(e-1)$
Verifier Output: Θ^{k+1}_{m+x}
validate ;
validate $\text{update}^X_i(e-1)$ is applied in $\mathfrak{A}^X_i(e)$;
validate $\mathfrak{A}^X_i(e)$ leads to $\mathfrak{A}(e)$;
assert $\Theta^k_{m+x} = \bigcup_{i=0}^{n} \theta^k_{m+x}(i)$
rem_vol $\leftarrow verifyMatchCombiner_\pi(\Theta^k_m, \Theta^k_{m+x}, \mathfrak{X}^{X,Y}_i)$
if *rem_vol > 0* **then**
  if $\mathfrak{X}^{X,Y}_i$ *is sell* **then**
    assert Θ^k_{m+x}.max_buy $< \mathfrak{X}^{X,Y}_i$.price;
  else
    assert Θ^k_{m+x}.min_sell $> \mathfrak{X}^{X,Y}_i$.price;
  end
end
return $\Theta^{k+1}_{m+x} = \bigcup_{i=0}^{n} \{\theta^k_{m+x}(i) \cup \mathfrak{X}^{X,Y}_i\}$

Algorithm 9: verifyMatchingSettlement

Verifier Input : $\mathfrak{A}^X(e-1)$, $\mathfrak{A}^X(e)$
Prover Input : π_S, π_M, $\mathfrak{A}(e-1)$, $\mathfrak{A}(e)$
Verifier Output: H(Θ)
debited $\leftarrow verifyAccountIntegrity_{\pi_S}(\mathfrak{A}^X(e-1), \mathfrak{A}^X(e))$
$\Theta \leftarrow verifyOrderCombiner_{\pi_M}(\varnothing, \mathfrak{A}(e))$
assert debited = Θ.matched_volume[X];
return *H($\mathfrak{A}^X(e-1)$), H($\mathfrak{A}^X(e)$), H(Θ)*

Algorithm 10: ratifySwap

Input : update^X_i, update^Y_i, $\mathfrak{X}^{X,Y}_i$
Output: Sig(update^X_i, $\mathbb{O}_{\not\in}$), Sig(update^Y_i, $\mathbb{O}_{\not\in}$)
assert $\mathfrak{X}^{X,Y}_i \in \text{update}^X_i$;
assert $\mathfrak{X}^{X,Y}_i \in \text{update}^Y_i$;
assert update^X_i debits the full $amount^X$;
assert update^Y_i credits no additional Y coins;
return *Sig(update^X_i, $\mathbb{O}_{\not\in}$), Sig(update^Y_i, $\mathbb{O}_{\not\in}$)*

Algorithm 11: verifyPuzzle

Verifier Input: n, h, k, C_K, α, b, y
$g \leftarrow h^{\Pi^r_{i=1} q_i} \mod n$
$c \leftarrow H(\alpha \| b) \mod R$
for *j in 1..s* **do**
  for *i in 1..k* **do**
    assert $g^{y_{i,j}} * b^{-c_{i,j}}_{i-1,j} = g^{\alpha_{i,j}} \mod n$
    assert $b^{y_{i,j}}_{i-1,j} * b^{-c_{i,j}}_{i,j} = b^{\alpha_{i,j}}_{i-1,j} \mod n$
  end
end
assert $C_K = b_k$ (mod n)

Algorithm 7: verifyMatch

Verifier Input : Θ^k_m, Θ^k_{m+1}, $\mathfrak{X}^{X,Y}_i$
Prover Input : $\lambda(\mathfrak{X}^{Y,X}_j \in \Theta^k_m)$, $\mathfrak{X}^{Y,X}_j$
Verifier Output: $\mathfrak{X}^{X,Y}_i$.rem_out_vol
n $\leftarrow \mathfrak{X}^{Y,X}_j$;
for $s \in \lambda(\mathfrak{X}^{Y,X}_j \in \Theta^k_m$ **do**
  if *if* $\mathfrak{X}^{X,Y}_i$ *is a sell order* **then**
    assert n is first child with max buy price $\geq \mathfrak{X}^{X,Y}_i$.price;
  else
    assert n is first child with min sell price $\leq \mathfrak{X}^{X,Y}_i$.price;
  end
  n.buy = max(n.buy, s.buy);
  n.sell = min(n.sell, s.sell);
  if *if s is left child* **then**
    n.hash $\leftarrow$ H(n.sell $\|$ s.hash $\|$ n.hash $\|$ n.buy);
  else
    n.hash $\leftarrow$ H(n.sell $\|$ n.hash $\|$ s.hash $\|$ n.buy);
  end
end
assert n.hash = Θ^k_m;
matched_volume $\leftarrow$ min($\mathfrak{X}^{X,Y}_i$.rem_out_vol, $\mathfrak{X}^{Y,X}_j$.rem_in_vol);
$\mathfrak{X}^{Y,X}_j$.rem_in_vol -= matched_volume;
$\mathfrak{X}^{Y,X}_j$.rem_out_vol -= matched_volume * $\mathfrak{X}^{X,Y}_i$.price;
n $\leftarrow \mathfrak{X}^{Y,X}_j$;
if $\mathfrak{X}^{Y,X}_j$*.rem_vol == 0 or* $\mathfrak{X}^{Y,X}_j$ *is sell* **then**
  n.buy $\leftarrow$ 0;
end
if $\mathfrak{X}^{Y,X}_j$*.rem_vol == 0 or* $\mathfrak{X}^{Y,X}_j$ *is buy* **then**
  n.sell $\leftarrow$ INF;
end
for $s \in \lambda(\mathfrak{X}^{Y,X}_j \in \Theta^k_m$ **do**
  n.buy = max(n.buy, s.buy);
  n.sell = min(n.sell, s.sell);
  if *if s is left child* **then**
    n.hash $\leftarrow$ H(n.sell $\|$ s.hash $\|$ n.hash $\|$ n.buy);
  else
    n.hash $\leftarrow$ H(n.sell $\|$ n.hash $\|$ s.hash $\|$ n.buy);
  end
end
assert n.hash = Θ^k_{m+1};
return $\mathfrak{X}^{X,Y}_i$*.rem_out_vol - matched_volume*

APPENDIX C
EXTENDED SPECIFICATIONS

A. Recursiveley Composable ZK Proofs

We outline the algorithms 7 and 8 for the recursively composable ZKP.

A swap challenge enforces that a trade is matched with at least the asked limit price (cf. Section V-C1). Figure 7 shows the cost evolution of swap challenges given the number of swaps within one *eon* (measured with 10 users). Note that these costs are reset each eon.

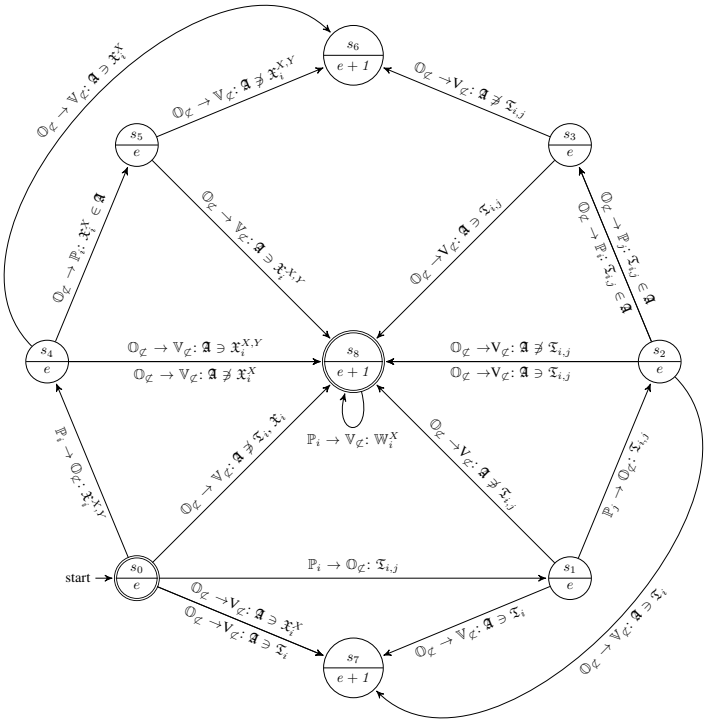

Fig. 6: Finite state automaton capturing the provable integrity of $\mathbb{O}_{\not\subset}$. A dishonest $\mathbb{O}_{\not\subset}$ always finds itself trapped in a reject state, while an honest $\mathbb{O}_{\not\subset}$ can prove its integrity in *eon* $e+1$ after committing to its e operations regardless of the behavior of members of $\mathbb{P}$..

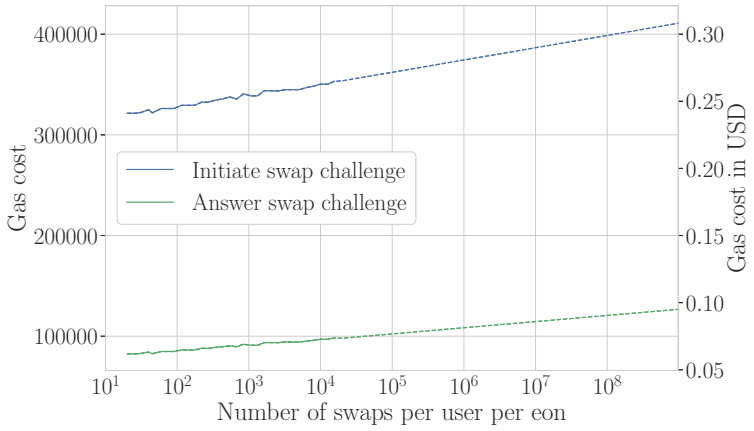

Fig. 7: Cost evolution for swap challenges depending on the number of swaps executed by a single user. The continuous line is measured empirically, the dotted line estimated.