FuturesMEX: Secure, Distributed Futures Market Exchange

Fabio Massacci*, Chan Nam Ngo*, Jing Nie†, Daniele Venturi‡ and Julian Williams§
*University of Trento, IT †University of International Business and Economics Beijing, CN
‡University of Rome "La Sapienza", IT,§University of Durham, UK

To appear in the Proceedings of the IEEE Symposium on Security & Privacy, May 2018, San Francisco.

Abstract

In a Futures-Exchange, such as the Chicago Mercantile Exchange, traders buy and sell contractual promises (futures) to acquire or deliver, at some future pre-specified date, assets ranging from wheat to crude oil and from bacon to cash in a desired currency. The interactions between economic and security properties and the exchange's essentially non-monotonic security behavior; a valid trader's valid action can invalidate other traders' previously valid positions, are a challenge for security research. We show the security properties that guarantee an Exchange's economic viability (availability of trading information, liquidity, confidentiality of positions, absence of price discrimination, risk-management) and an attack when traders' anonymity is broken. We describe all key operations for a secure, fully distributed Futures-Exchange, hereafter referred to as simply the 'Exchange'. Our distributed, asynchronous protocol simulates the centralized functionality under the assumptions of anonymity of the physical layer and availability of a distributed ledger. We consider security with abort (in absence of honest majority) and extend it to penalties. Our proof of concept implementation and its optimization (based on zk-SNARKs and SPDZ) demonstrate that the computation of actual trading days (along Thomson-Reuters Tick History DB) is feasible for low-frequency markets; however, more research is needed for high-frequency ones.

I. Introduction

A futures contract is a standardized agreements between two parties to buy or sell an underlying asset, at a price agreed upon today with the settlement occurring at some future date [56]. They are "promises" to buy or sell, and these "promises" can themselves be traded. Such trading is conducted in a double auction market operated by a centralized clearing house called Futures Exchange [1], such as the Chicago Mercantile Exchange (CME). [1] Traders can 'quote' a future by specifying a price and notional volume of assets at which they will buy or sell (a limit order), or initiate a trade by placing a market order for a "promise" of a quantity (purchase or sale) at the best price from the standing quotes.

General financial intermediation as embodied by a Futures Exchange is still centralized and more expensive than traditional payment networks which have been successfully challenged by Bitcoin [48]. ZeroCash [7] amongst others.

As opposed to simple decentralized price discovery [20], making a full trading exchange distributed requires the designed to solve several security challenges, besides the typical security issues of distributed payments which can be solved by leveraging (and indeed we do so) on ZeroCash [7]: zero-knowledge succinct non-interactive arguments of knowledge, i.e. zk-SNARKs [10].

The first challenge is the *interplay between security and economic viability* [43]. Whereas integrity is obviously needed for payments (see the Ethereum DAO mishaps [21]), confidentiality seemed less critical for exchanges [20]; one can trace all transactions to a Bitcoin's ID by using public information in the blockchain, yet this hardly stopped Bitcoin from thriving [7, p. 459]. Here, disclosing a trader's account allows attacks based on price discrimination and would inevitably lead to a market collapse (we illustrate this effect in §III).

FuturesMEX technologies are the object of the following patent applications: US62/625,428 and PG448130GB.

[1] On the CME, futures contracts range from bushels of corn to Euro/US$ exchange rates. Recently, CBOE and CME launched Bitcoin futures markets. These are 'cash-futures', that is as they are settled in cash. Eurodollars futures are the largest world market by notional volume: in quadrillions of dollars/year. See https://en.wikipedia.org/wiki/List_of_futures_exchanges.

Another challenge w.r.t. other crypto applications such as auctions (e.g., [54]) is the simultaneous need to i) make publicly available all offers by all parties, ii) withhold the information on who actually made the quote and iii) trace the *honest* consequences of an anonymous public action to the responsible actor. The prototypical example is posting a public, anonymous buy order while personally accruing the revenues from the sale (without even the seller knowing the actual buyer and vice-versa). The Exchange must also guarantee that iv) actors do not offer beyond their means, which is an issue related to double spending [4], double voting [12], or groundless reputation rating [62]. E-voting provides traceability for one's own vote, not to the ensemble of agents. Applications of e-cash and privacy-preserving reputation systems guarantee anonymity for honest actions and traceability only in case of malfeasance, not for honest behavior.

Further, *e-cash or voting protocols are essentially monotonic* in terms of legitimacy of digital assets of honest parties when other honest parties join: valid security evidence (e.g. commitments, etc.) accrue over protocol steps performed by honest parties. Once's Alice proved she has money (or she casted a correct vote), the protocol can move on and check Bob's assets. Alice's claims are never invalidated by Bob (if she stays honest and is not compromised). Monotonicity is clearly visible in the security proofs for cash fungibility in ZeroCash [7], or vote's eligibility in E2E [33]. This allows for efficient optimization (e.g. [62]) as a multi-party computation (MPC) with n interacting parties may be replaced by n independent non-interactive proofs (i.e. zk-SNARK).

In contrast, *financial intermediation is not monotonic*: Alice's asset (e.g. a trader's inventory) might be proven (cryptographically) valid by Alice and later made (economically) invalid by the honest Bob who, by just offering to sell assets and thus changing the market price, can make Alice bankrupt without any action on her side. Hence, v) security evidence by honest parties must be potentially discarded, and vi) the protocol must account for Alice's "honest losses" and fix them because there is no centralized exchange covering them.

Our contribution. We provide the first, secure, distributed Futures Market Exchange which replicates the functionalities from the CME Globex specifications manual, including each of the main quote types (limit and market orders), needed to build more complex quotes and all standard margin accounting and marking to market features. Our *goals* are the following:

1) To put forward a cryptographic ideal functionality for a distributed futures market, that captures all of the key security requirements. This is an ideal realization of a distributed futures market, where the market is run by a trusted third party which knows the secret inputs of all participating traders, and lets the market evolve on their behalf. Such a functionality, by construction, embodies features (i)-(vi) described above.

2) To design a cryptographic protocol securely realizing our ideal functionality. Our protocol combines multiparty computation (MPC) and non-interactive zero-knowledge proofs on committed inputs, only relying on the basic assumptions of secure broadcast channels between traders and an anonymous network. These assumptions already appeared in several prior works, most notably [7]. We replace the local constraints verification of the MPC with non-interactive proofs for efficient generation of publicly verifiable transactions and scalability w.r.t. the number of traders. Full MPC is only performed for sub-tasks capturing the non-monotonicity and anonymity requirements of the market. We prove the security of our protocol with security-with-abort—where we allow an adversary to abort the computation after receiving its own intermediate outputs [32]—and extend it so that an aborting adversary is penalized by forfeiting its hard won stake in the market, the ultimate discouragement in our setting.

3) To show that our approach is feasible. We do so, by providing a proof of concept implementation using zk-SNARKs for the zero-knowledge proofs, and the SPDZ protocol [24] for securely realizing the required MPC sub-tasks. We further optimize our protocol in order to yield a 70% efficiency gain. Our results show that our solution is *feasible for low frequency markets* at CME (e.g. trading in Lean Hog commodities): a trading day can be executed in a day by an Amazon's EC2 large VM. Further optimizations are needed for high frequency trading in the largest markets (Eurodollar, Foreign Exchange and Crude Oil futures), for instance by parallelizing proof generation as most of them are independent, improvements in the zk-SNARK implementation; different commitment functions or batch proofs for good standing (e.g. proving the validity of a trader's inventory for a range of prices); or buying a 30M$/year hardware such as the CME

data centre.[2]

Non-Goals. The focus of our paper is to protect against *operational* attacks on integrity, anonymity and confidentiality, *economics and social* attacks are, and always will be, possible similarly to centralized systems (e.g. insider trading or cartels manipulating the underlying assets) and they are typically dealt with, ex-post law enforcement [42].

Technical challenges. The main difficulty that we need to face is the fact that futures markets are fully stateful systems where at each round the functionality changes its internal state, due to a valid move performed by an agent which updates the public information and her own private information. As mentioned, the global constraint is such that an agent's legitimate move can unpredictably make another agent's state invalid due to the change in the public information. The market as a whole must transit to a new state where the legitimate move is accepted and the invalid state is fixed. This intrinsic feature (which we feel is best described by *non-monotonicity*) limits the ability of protocol designers to improve on communication complexity by replacing interactive MPC steps with independent non-interactive proofs.

While the satisfaction of individual constraints could be solved by a standard "commit-and-prove" approach among the concerned individuals, this would not work for the global constraints. The alternative would be to implement the whole functionality via general-purpose MPC. However, this solution is unacceptable given the large variance in trading activity: some traders only make few large orders, others make several trades every few milliseconds, or even fractions of milliseconds [41]. This leads to an efficiency requirement (which we dub *proportional burden*), informally stating that each computation should be mainly a burden for the trader benefiting from it, which cannot be met by a naïve MPC implementation. This intuition is confirmed by our experiments, which show that our approach is superior under some general conditions that are realized in practice.

Paper organization. In the rest of the paper we introduce the key aspects of futures markets (§II) and illustrate a price discrimination attack due to loss of confidentiality (§III). A formal model of the centralized futures market (§IV) is followed by the description of the ideal functionality its secure distributed version (§V). Then we describe the (nonmonotonic) security state of the functionality (§VII), our crypto building blocks (§VI), and our protocol (§VIII). We provide a proof sketch on its security (§IX) and discuss how to go beyond security-with-abort (§X). Our proof of concept and its performance results are presented in §XI and §XII. Finally, we survey related work (§XIII), and conclude the paper (§XIV).

II. An Introduction to Futures Markets

To illustrate how markets work, we explain the key trading mechanisms and discuss some aspects of the market microstructure of futures contracts [30], [31]. Fundamental participants in a futures market include traders, exchanges and regulatory bodies as summarized in Table I.

Traders post buy (bid) or sell (ask) orders for a specific futures contract in the market. The *trading position* characterizes a trader as a buyer or a seller: sellers take *short* positions by selling an amount of futures contracts; buyers take *long* positions by buying futures. Obviously buyers prefer to purchase contracts at lower prices and sellers prefer to sell contracts at higher prices. Traders can also cancel orders immediately after having posted them to adapt to fast changing markets (a heavily used feature).

The *Exchange* acts as centralized intermediary between buyers and sellers and guarantees price discovery, matching and clearing. It manages risks and guarantees the fairness of the market (See Table I for a short summary of key requirements from an economic perspective).

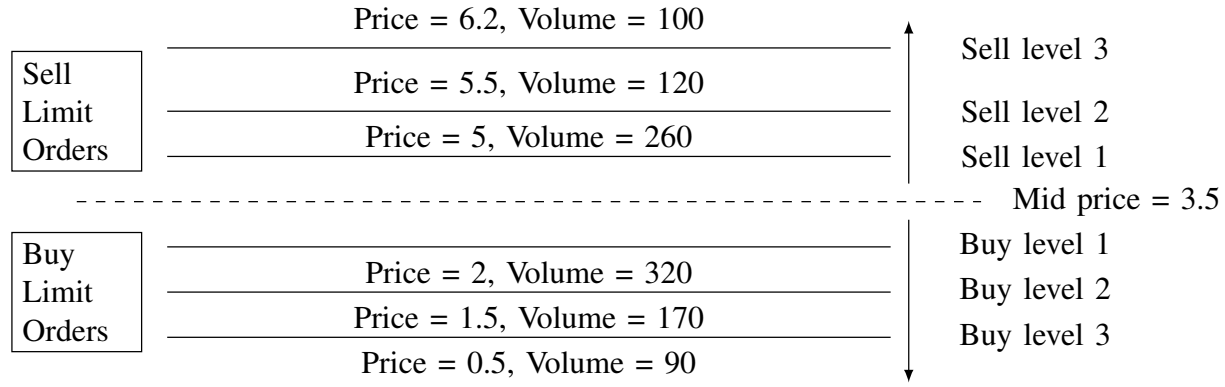
The first important functionality is to made available to all traders an aggregated list of all waiting buy and sell (anonymized) orders: the *central limited order book*. It includes the volume of contracts being bid or offered at each price point. It is illustrated in Figure 1.

Buy and sell orders at the same prices are matched by the Exchange until the required volume of contracts is reached. Matched orders will go through a clearing and settlement process to complete a transaction [52]. The exchange usually operates its own clearing house which is responsible for having a daily settlement

[2]See "Lease back datacenter" in the last CME SEC 10-K report (page 58).

TABLE I: Key Compositions and Characteristics of Futures Market

Traders Characteristics:	
Possible Positions	Buy-side traders holding long positions. Sell-side traders holding short positions.
Possible Actions	Submit (Market/Limit Orders) and Cancel (Limit) Orders.
Exchanges Main Functions:	
Price discovery and order matching	Disseminating the real-time market data to market participants; Providing a *central limited order book* (cf. Fig 1): an electronic list of all waiting buy and sell quotes organized by price levels and entry time. Matching engines use algorithms to match buy and sell quotes with a price and time priority principle.
Risk management and clearing of orders	Clearing house is responsible for having a *daily/final settlement* by the process of "mark-to-market", so that no pending promise (to buy or sell) and no debt remains unfulfilled. Traders need to deposit an *initial margin* and maintain a minimum funding in the margin account above the *maintenance margin*; otherwise, they will receive a *margin call* for additional funding. Traders failing below the minimum are forced to liquidate their open positions and netted out.
Market fairness and absence of price discriminations	For fairness, *traders are anonymous* as exchanges hold all info about them and never reveal it to others. A trader only see the details of her own orders, and not even the ID of the counter party of an order matching her own order executed through the exchange, as this would allow for *price discrimination*.
Major Players	
Chicago Mercantile Exch.	The largest derivatives market with 3.53 billion of contracts traded in 2015 [26].
Eurex Exchange (Eurex)	The largest European derivatives market with 2.27 billion of contracts traded in 2015 [26].
Regulatory Bodies	Futures markets are regulated by independent government agencies to protect market participants and prevent fraud and manipulation activities, such as the CFTC [57] and the SEC [58].



An order book with limit orders. The dashed line is the average mid-price which is calculated by the CME as the (unweighted) average of all price levels. Traders' holdings are evaluated against the mid-price.

Fig. 1: Order Book

TABLE II: Samples of Market Activity

The table shows the maximum number of active traders (#T), number of posted orders (#PO), and matching orders (#MO) for some futures contracts (Eurodollar being world's largest). Cancelled orders' number is close to that of posted orders. Data is obtained from the CME tapes via the Thomson Reuters Tick History database.

Contract	Lean Hog LHZ7			Eurodollar GEH0		
Trading Day	#T	#PO	#MO	#T	#PO	#MO
Low	15	1067	46	14	23469	85
Normal	17	3580	146	199	267089	7907
High	33	6709	536	520	376075	8402

for each futures contract by the process of "mark-to-market", which is valuing the assets covered in future contracts at the end of each trading day. Then profit and loss are settled between long positions and short positions.

Table II illustrates the variability of the markets by comparing some days for the Eurodollar, the largest market in the world, together with Lean Hog, a less frequently traded futures.

Informal Properties. From a security perspective an exchange is clearly an instance of a *multi-party reactive security functionality* [17]: every agent must satisfy individual constraints (monotonic) and the

TABLE III: Forcing Alice out of the market

Alice accumulates 90 selling contracts currently at the price of 10 and have a cash margin of 1400. As the price fluctuates by δ_P her inventory liquidation price is $X_{\mathsf{Alice}} = -90 \times (10 + \delta_P)$, and her net position is $N_{\mathsf{Alice}} = 1400 + X_{\mathsf{Alice}} = 500 - 90 \times \delta_P$. When $\delta_P = 0$, she holds a small margin (at \$500). When $\delta_P = 6$, her net position drops to -\$40 and she has to be netted out from the market.

	Price = \$10			Price = \$16
Trader	**Cash**	**Contracts**	**Position**	**Position**
Alice	1400	-90	500	*-40*
Bob	1200	30	1500	1680
Carol	1200	30	1500	1680
Eve	1200	30	1500	1680

system as a whole must satisfy global constraint (possibly non-monotonic). The economic requirements in Table I can be directly transformed into the security requirements below.

Availability of Order Book with Confidentiality of Trader Inventory. Acting as counter party for each trader, the exchange must hold all trading information including prices, volumes, margins, and traders ownership of orders, etc. It has to protect a trader's own inventory without leaking it to other traders.

Market Integrity and Loss Avoidance. The exchange implements trading (execute matching orders), and guarantee final settlements (traders' margin meet posted orders) after each event to ensure the integrity of the marketplace. More constraints such as limiting a trader's largest position are added in practice (we omit them due to lack of space).

Trader's Anonymity. The exchange must prevent the linkage of orders by the same trader. This is done by managing an anonymous central limit order book where only bid and ask prices are publicly available. In this way, traders will not be able to identify and forecast others' trading strategies.

Trader's Precedence Traceability. The exchange must allow the linking of limit orders to the individual traders so that matching orders can be accrued to the traders who made them in the exact order in which they where posted.

In traditional applications of MPC, such as auctions and e-voting, there is no difference between the parties: everybody submits one bid or casts one vote. This is not true for general financial intermediation: retail and institutional investors are 71% of traders in the TSX market, but only make 18% of the orders [41]. Traders responsible for the bulk of the over 300K orders per day were "algorithmic traders" who, in 99% of the cases, only submitted limit orders (i.e., never to be matched in an actual trade). Such a difference must be accounted for by any protocol, an efficiency constraint that we state below.

Proportional Burden: Each computation should be mainly a burden for the trader benefiting from it (e.g. posting an order or proving one's solvency). Other traders should join the protocol only to avoid risks (of failed solvency).

III. Loss of Anonymity and Price Discrimination

If confidentiality and anonymity fail, some traders can act strategically by posting orders that they do not intend to honor so that other traders will be maliciously forced out of the market(see the Risk Management entry from Table I in §II). This attack has been first reported by [43].

Assume Alice, Bob, Carol, and Eve are in a market. Alice accumulates a large short position of 90 contracts selling at \$10 each, each other trader buys 30 contracts from Alice at this price. In English, her inventory holds 90 promises to sell. To estimate a trader's exposure, the Exchange assumes that all contracts are bought and sold instantaneously at the current mid price of \$10 (See Figure 1). So, to fulfill her promise to sell 90 contracts Alice would have to buy them first from the current mid price and reduce her cash availability to $1400 - 90 \cdot 10 = 500$. We have the situation shown in Table III (left).

If Alice could wait, she could post a buy order of \$9.50. If somebody eventually matched her order later in the day she would obtain a modest profit (50c per contract). If Carol and Eve know that Alice is a small investor and needs cash, they can generate an instant profit by changing the liquidity profile of

TABLE IV: Market Indicators for the current round t of the Futures Market

Indicator	Notation	Definition	Description
Best sell price index	l_{sell}	$\min\{\ell' \mid (t', \ell', i', v' < 0) \in \mathcal{O}\}$	Index of the lowest price of all sell orders in the order book
Best buy price index	l_{buy}	$\max\{\ell' \mid (t', \ell', i', v' > 0) \in \mathcal{O}$	Index of the highest price of all buy orders in the order book
Mid price	$\bar{p}$	$(p_{l_{\mathsf{sell}}} + p_{l_{\mathsf{buy}}})/2$	The average value of the best buy price $p_{l_{\mathsf{buy}}}$ and best sell price $p_{l_{\mathsf{sell}}}$
Available volume at price p_h	V_h	$\sum_{(t' \leq t, h, i', v') \in \mathcal{O}} \lvert v' \rvert$	The sum of volumes over all orders at price p_h
Available sell volume up to p_h	V_h^{sell}	$\sum_{\ell=l_{\mathsf{sell}}}^{h} V_\ell$	Aggregation of all volumes available from the best sell price l_{sell} to the final maximum acceptable price p_ℓ ($\ell \geq l_{\mathsf{sell}}$)
Available buy volume down to p_h	V_h^{buy}	$\sum_{\ell=l_{\mathsf{buy}}}^{h} V_\ell$	Aggregation of all volumes available from the best buy price l_{buy} to the final least acceptable price h ($\ell \leq l_{\mathsf{buy}}$)

the market. They can post buy orders at slightly higher prices, this changes the mid prices and pushes the liquidation price of Alice's position higher. Alice could try to sell to those buy orders, but this pushes the contracts more deeply negative in a rising market exacerbating her problem of being close to the margin call. Eventually, the liquidation price is high enough, e.g. \$16, that Alice's net position is below the margin call threshold and Alice is cashed out, with a realized payout to the other traders, i.e. her \$500 is given to the other traders.

The other traders can then cancel their orders and the price could then decrease back to \$10 or even lower (when Alice's trades would have been profitable), but Alice cannot benefit from this price as she has already been cashed out. The other traders have not actually traded anything and still forced out Alice by adjusting their buy quotes strategically. Eve and Carol have price discriminated Alice: their pricing strategy could only work because they *knew exactly* how much was in Alice's pocket and therefore how much was needed to nudge her out. The opposite problem can be generated from a long position and the market then being artificially deflated.

IV. Formal Futures Market Definition

Formally a futures market consists of N traders, each trader identified via an index $i \in [N]$, and a sequence of L available prices[3] (for the limit orders) in ascending order (i.e., $p_1 < p_\ell < p_L$ for $\ell \in [L]$). The market evolves in rounds, where T is the maximum (constant) number of rounds[4]. The data stored (and updated) for the current round $t \in [T]$ is a tuple $(\mathcal{O}, \mathcal{I})$.

- The set $\mathcal{O}$ is the *limit order book*, and consists of a sequence of tuples $o' = (t', \ell', i', v')$, where o' represents a limit order posted at round $t' \leq t$ by a trader $P_{i'}$ for a desired volume $v' \neq 0$ of price $p_{\ell'}$. A limit order is a "sell" order if $v' < 0$, and a "buy" order otherwise.
- $\mathcal{I}_i = (m_i, v_i)$ is the *inventory* of a trader $i \in [N]$ where:
  - The value v_i is the number of contracts held by the trader (for long positions $v_i > 0$, for short ones $v_i < 0$);
  - The value m_i is the cash available to the trader.

Initially, every trader starts with no contract in the inventory as well as a non-negative deposit (i.e., $\forall i \in [N] : v_i = 0, m_i \geq 0$), and the market is an empty order book (i.e., $\mathcal{O} = \emptyset$).

To express the constraints that a trader can meet her obligations and make orders within her means we introduce some auxiliary functions. The *instant net position* η_i is the cash she can get (or must pay) upon liquidating all her contracts:

$$\eta_i = m_i + \mathsf{cash}(v_i) \tag{1}$$

[3] In the CME Globex, trading operations starts with an indicative opening price (IOP). Other prices are an integer number of upward or downward ticks from the IOP. A price is always non-zero and each underlying asset of a futures contract usually has a reasonable upper bound for the price. Hence we can map possible prices into a finite list of L available prices and refer to a price only with its index ℓ.

[4] At CME an open cry starts at 7:20 and ends at 13:59:00, the evolution of time is accounted for by with the number of rounds.

TABLE V: Value $\mathsf{cash}(v)$ to liquidate an inventory of volume v

Cases	Definition	Description
$v > 0$ (long) and $V_1^{\mathsf{buy}} \geq v$	$\sum_{h=l_{\mathsf{buy}}}^{l+1} p_h \cdot V_h + p_\ell \cdot (v - V_{l+1}^{\mathsf{buy}})$	Cash a trader can get upon selling all volume v at the current buy quotes in the order book, where l is the least index s.t. $V_l^{\mathsf{buy}} \geq v$.
$v > 0$ (long) and $V_1^{\mathsf{buy}} < v$	$\sum_{h=l_{\mathsf{buy}}}^{1} p_h \cdot V_h + p_1 \cdot (v - V_1^{\mathsf{buy}})$	The order book *does not have enough* supply on the buy side.
$v < 0$ (short) and $V_L^{\mathsf{sell}} \geq \lvert v \rvert$	$-\sum_{h=l_{\mathsf{sell}}}^{l-1} p_h \cdot V_h + p_l \cdot (\lvert v \rvert - V_{l-1}^{\mathsf{sell}})$	Cost a trader must pay to buy all volume v from the current sell quotes in the order book, where l is the least index s.t. $V_l^{\mathsf{sell}} \geq \lvert v \rvert$
$v < 0$ (short) and $V_L^{\mathsf{sell}} < \lvert v \rvert$	$-\sum_{h=l_{\mathsf{sell}}}^{L} p_h \cdot V_h + p_L \cdot (\lvert v \rvert - V_L^{\mathsf{sell}})$	The order book *does not have enough* supply on the sell side.

where $\mathsf{cash}(v_i)$ represents the *liquid* value of the inventory, i.e., the amount of cash a trader P_i can get (or must pay) upon selling (or buying) all volume holding v_i at the current buy (or sell) quotes in the order book.

The function $\widehat{\cdot}$ represents the estimated value of a trader's inventory variables if the market accepted her new order. Auxiliary definitions used in the calculation of market conditions are listed in Table IV (mid price, best sell price, etc.) while $\mathsf{cash}(v_i)$ is defined in Table V. For the estimated value of the inventory when a trader P_i posts an order (t, ℓ, i, v) at price p_ℓ for a volume v in round t, we have:

$$\widehat{m_i} = m_i - p_\ell \cdot v, \quad \widehat{v_i} = v_i + v, \quad \widehat{\eta_i} = \widehat{m_i} + \mathsf{cash}(\widehat{v_i}) \tag{2}$$

We can now formalize the properties, which must hold at every round, corresponding to the security/economic requirements informally introduced in §II.

Definition 1 (Market Integrity)**.** The amount of cash available by all traders is constant ($\sum_{i=1}^{N} m'_i = \sum_{i=1}^{N} m_i$) where m'_i is the margin at time $t' \leq t$), the total volume holding is zero ($\sum_{i=1}^{N} v_i = 0$), and the best buy price is less than the best sell price ($1 \leq l_{\mathsf{buy}} < l_{\mathsf{sell}} \leq L$).

Definition 2 (Traders Solvency)**.** All traders have a positive instant net position ($\eta_i \geq 0$) and can afford the new limit order at posting time ($\widehat{\eta_i} \geq 0$).

Definition 3 (Availability of Orders with Anonymity of Trader)**.** For any order (t, ℓ, i, v) posted at time t, the order information (t, ℓ, v) must be made public before time $t + 1$, whilst information about i is only known to P_i.

Definition 4 (Confidentiality of Trader Inventory)**.** Only P_i knows the values of $\mathcal{I}_i = (m_i, v_i)$ as well as η_i with the exception of time T after mark-to-market when $v_i = 0$.

The two previous requirements imply that $\widehat{m_i}$ and $\widehat{v_i}$, as well as $\widehat{\eta_i}$ must also be confidential (otherwise one could recover the inventory by reversing the computation from orders).

Definition 5 (Trader's Precedence Traceability)**.** Let $\mathcal{O}$ be the current order book, (t, ℓ, i, v) be an order, and t' be the smallest round $t' < t$ such that $(t', \ell, i', -v') \in \mathcal{O}$ then the order book $\mathcal{O}^*$ at time $t+1$ *respects traders precedence* given order (t, ℓ, i, v) and order book $\mathcal{O}$ iff

1) if no such t' exists for $\mathcal{O}$, then $\mathcal{O}^* = \mathcal{O} \cup \{(t, \ell, i, v)\}$,
2) if $|v| < |v'|$, then $\mathcal{O}^* = \mathcal{O} \cup \{(t', \ell, i', v - v')\} \setminus \{(t', \ell, i', -v')\}$
3) else $\mathcal{O}^*$ respects traders precedence given order $(t, \ell, i, v - v')$ and order book $\mathcal{O} \setminus \{(t', \ell, i', -v')\}$

V. The Ideal Reactive Functionality

For expository purposes, both in the functionality's and in the protocol's description we allow an adversary to abort the computation after receiving its own intermediate outputs. This flavor of security is known as security with aborts [32]. In Section X we change the protocol to avoid scot-free aborts.

The futures market evolution is captured by an ideal reactive functionality $\mathcal{F}_{\mathsf{CFM}}$ where all the traders send their private initial inventory to a trusted third party (during the so-called **Initialize** phase), which lets the market evolve on their behalf. A typical evolution of the market includes processing orders (**Post/Cancel**

Futures Exchange Ideal Functionality $\mathcal{F}_{\mathsf{CFM}}$ runs in phases with a set of traders $(P_1, \ldots, P_N)$ and a list of prices $(p_1, \ldots, p_L)$.

Initialization: Upon $(\mathsf{init}, P_i, m_i)$ from all traders, accept the input iff $m_i \geq 0$. Hence, store $(m_i, v_i := 0)$ as the inventory of P_i. Finally, initialize $t := 0$ and $\mathcal{O} := \emptyset$.

Post/Cancel Order: If $t < T$, upon receiving $(\mathsf{post\_order}, P_i, \ell, v)$ (resp. $(\mathsf{cancel\_order}, P_i, t')$) from P_i, let $t := t + 1$.

1) Check $\ell \geq l_{\mathsf{buy}}$ for $v < 0$ ($\ell \leq l_{\mathsf{sell}}$ for $v > 0$). In case of **Cancel Order**, retrieve (t', ℓ', j, v') from $\mathcal{O}$ and check $j = i$..
2) Let $\mathcal{I}_i^*$ be an identical copy of $\mathcal{I}_i$, check $\widehat{\eta}_i \geq 0$ w.r.t. to $\mathcal{I}_i^*$ and the order book $\mathcal{O}^* := \mathcal{O} \cup (t, \ell, i, v)$ (resp. $\mathcal{O}^* := \mathcal{O} \setminus (t', \ell', j, v')$):
3) If any check fails, send $(\mathsf{invalid\_post}, t, \ell, v)$ (resp. $(\mathsf{invalid\_cancel}, t')$) to every trader; else send $(\mathsf{post\_order}, t, \ell, v)$ (resp. $(\mathsf{cancel\_order}, t')$) to every trader and proceed to **Margin Settlement** with input $\mathcal{I}_i^*$ and $\mathcal{O}^*$ (c.f. Fig. 3). ; if "succeed", let $\mathcal{I}_i = \mathcal{I}_i^*$, $\mathcal{O} := \mathcal{O}^*$, otherwise proceed to **Mark to Market**.
4) In case of **Post Order**, fulfill the order starting from the earliest opposite order of the same price already in the order book, until the new order is filled or there is no past order to match it with. (c.f. Fig. 3).

Mark To Market: at $t = T$, offset all positions, i.e. $\forall P_i : m_i := m_i + v_i \cdot \bar{p}$, and $v_i := 0$.

Fig. 2: The operations of the ideal functionality $\mathcal{F}_{\mathsf{CFM}}$ for posting, cancelling and marking to market

Order phases), netting out traders with insufficient funds to maintain their position, we refer to these traders hereon as "broke" traders (**Margin Settlement** phase), and finally offset all positions (**Mark to Market** phase). A formal description is in Fig. 2.

Intuitively, the matching process performed during the **Post Order** phase (c.f. Fig. 2). takes the new order (t, ℓ, i, v) and tries to match it with all previous limit orders of opposite side in the order book that have the same price. In other words, if the limit order is a buy order it will be matched with a sell order, and vice versa. The priority to match is given to the limit order with a smaller round index. When a match is found, the trade is reconciled, and the available cash, as well as the volume holding of the traders, is updated accordingly (i.e., on buy side: increase volume, decrease cash; on sell side: decrease volume, increase cash). The matching process stops either when the new order is fulfilled, or there is no past order that can fill the new one. In the latter case, the remaining volume is left in the order book as a new limit order.

An important feature of $\mathcal{F}_{\mathsf{CFM}}$, is to guarantee payable losses by each trader (i.e., $\eta_i \geq 0$). Hence, when the last round is reached, all traders must then offset their position, so that the data at round T will consist of all zero volumes, non-negative balances, and an empty order book.

Since the net position might change (due to the updates of the order book) it is necessary to check the new instant net position η_i^* of each trader P_i after the update. In case of any negative net position, the last update cannot be committed until all *broke* traders P_i (i.e., $\eta_i < 0$) are netted out, which is done in the so-called **Margin Settlement** phase. (c.f. Fig. 3). This requires each new broke trader to cancel all pending orders (becomes *canceled*), and buy/sell all contracts in the inventory that the trader is short/long, at whatever price available at the moment (becomes *netted*). At the end of the **Margin Settlement** phase the order fulfillment is resumed, and the update will be committed.

For simplicity, after a trader P_i is *netted out*, the trader cannot participate in the market in the subsequent rounds. In the worst-case scenario where: (i) the market cannot supply the margin settlement of broke traders (because, e.g., they hold too many contracts comparing to the current available volume in the order book), or (ii) even the margin settlement cannot bring a broke trader's position back to non-negative, the ideal functionality proceeds directly to **Mark to Market**.

Non-monotonicity. A challenging feature of the futures market's ideal functionality is its intrinsic non-monotonic behaviour, in a sense made precise below.

Remark 1. *The properties of private values belonging to a honest trader P_i executing the ideal functionality of Fig. 2–3 are* non-monotonic *in the actions of other honest traders: Let P_i be a* good trader *(private value $\eta_i > 0$) at round t with order book $\mathcal{O}$, and further assume that at round $t + 1$ the order book gets updated*

Margin Settlement is run with a candidate order book $\mathcal{O}^*$ and a candidate inventory $\mathcal{I}^*$ starting with a set of new broke traders $\mathcal{B} := \emptyset$.

1) Repeat the following steps until $\eta_i^* \geq 0$ for all good traders P_i:
   a) Compute the new instant net position η_i^* of all good traders P_i; if $\eta_i^* < 0$ let $\mathcal{B} := \mathcal{B} \cup \{P_i\}$.
   b) For each $P_i \in \mathcal{B}$, remove *all* limit orders $o_i := (t', l', i, v')$ from both $\mathcal{O}^*$ and $\mathcal{O}$, send $(\mathsf{remove}, (t', l', v'))$ to each trader.
2) if $\mathcal{B} = \emptyset$, return "succeed'; else let $\mathcal{O}^* := \mathcal{O}$ and $\mathcal{I}^* := \mathcal{I}$ and repeat the following steps for each $P_i \in \mathcal{B}$, until $\mathcal{B} := \emptyset$:
   a) Net out P_i by repeatedly running Order Fullfilment with fixed input $(t, l_{\mathsf{sell}}, i, v_i)$ for short position (or $(t, l_{\mathsf{buy}}, i, v_i)$ for long position), until $v_i = 0$. If the market cannot supply the margin settlement of P_i, i.e, there is no order to match, return "fail".
   b) Let $\mathcal{B} := \mathcal{B} \setminus \{P_i\}$.
3) Return "succeed".

Order Fulfillment for $o_i = (t, l, i, v)$ starts with $t' = 1$, repeat the following for each entry $o_j = (t', l, j, v') \in \mathcal{O}$ such that $v \cdot v' < 0$:

- Send $(\mathsf{match}, t', l, v')$ to each trader;
- Compute the matched volume $\delta := \mathsf{min}(|v|, |v'|)$, then remove δ from o_i and o_j, i.e. in case $v > 0$, $o_i^* := (t, l, i, v - \delta)$ and $o_j^* := (t', l, j, v' + \delta)$ (otherwise swap i and j).
- Let $\mathcal{O}^*$ be an identical copy of $\mathcal{O}$, where the orders o_i and o_j are replaced, respectively, with o_i^* and o_j^*.
- In case $v > 0$, update the inventories as follows (in case $v < 0$, swap i and j in the equations below):

$$m_i^* := m_i - p_l\delta \qquad v_i^* := v_i + \delta \qquad\qquad m_j^* := m_j + p_l\delta \qquad v_j^* := v_j - \delta;$$

- Let $\mathcal{I}^*$ be an identical copy of $\mathcal{I}$ where the inventories of P_i and P_j are replaced, respectively, with (m_i^*, v_i^*) and (m_j^*, v_j^*).
- Run **Margin Settlement** with input $\mathcal{O}^*$ and $\mathcal{I}^*$.
- If **Margin Settlement** returns "fail", proceed to **Mark to Market** (Fig.2) otherwise, let $\mathcal{O} := \mathcal{O}^*$, $\mathcal{I} := \mathcal{I}^*$, and:

$$\text{if } v' = 0, \text{ let } \mathcal{O} := \mathcal{O} \setminus o_j; \qquad \text{if } v = 0 \text{ , let } \mathcal{O} = \mathcal{O} \setminus o_i$$

- Define $t' := t' + 1$, and repeat the above until $t' = t$ or $v = 0$.

Fig. 3: The operations of the ideal functionality $\mathcal{F}_{\mathsf{CFM}}$ for margin settlement and order fulfillment

to $\mathcal{O}^*$ due to an offer posted by another good trader $P_j \neq P_i$. The new order book $\mathcal{O}^*$ affects the value $\mathsf{cash}(v_i)$ *(Table V), which might result in a* negative *instant net position η_i (Eq.* (1)*), thus making P_i a* bad trader *at round $t + 1$, even if it was inactive during that round.*

Security properties. We briefly illustrate why $\mathcal{F}_{\mathsf{CFM}}$ fulfils the security requirements of the futures market in §II. The *Availability of Orders with Anonymity of Trader* property is guaranteed by broadcasting only $(\mathsf{post\_order}, \ell, v)$ upon receiving a $(\mathsf{post\_order}, P_i, \ell, v)$ from P_i. The same reasoning applies for canceling orders. *Confidentiality of Trader Inventory* is guaranteed as $\mathcal{F}_{\mathsf{CFM}}$ keeps the trader's inventory secret, all broadcasts post_order, cancel_order, invalid_post, invalid_cancel, match and remove contains no inventory information (m_i, v_i, η_i $\widehat{m_i}$ or $\widehat{v_i}$). As all the computations of $\mathcal{F}_{\mathsf{CFM}}$ respect the conditions in Def. 1 and Def. 2, *Market Integrity* and *Traders Solvency* properties are preserved. The *Trader's Precedence Traceability* property is also maintained due to: (i) only the owner of an order can match/cancel that order and (ii) only a *good* trader can post/cancel in normal phase while only *broke* traders can cancel and *canceled* traders can post during margin settlement phase. The *Proportional Burden* is obviously satisfied because we have a centralized functionality. We return to its satisfaction on the actual distributed protocol.

VI. Assumptions and Crypto Building Blocks

We elected to use as many standard crypto blocks as possible for both protocol construction and reliability of security proofs.

a) Anonymous Communication Network and Secure Broadcast Channel: Recall that the futures market ideal functionality guarantees full anonymity of the traders. To this end, we assume an underlying anonymous network that hides the traders' identifying information (e.g., their IP address). This assumption was already used in several prior works, most notably [7]. We also assume secure broadcast channels between the traders. Such channels could be implemented by utilizing a consensus protocol, e.g. PBFT [18].

Initial Bootstrap: As we employ several zero knowledge functionalities in our protocol, instantiated with zk-SNARK [10], an initial setup is required for global information such as proving keys and verifying keys, which can be achieved securely in practice with MPC as in [9].

b) Commitment Schemes.: We rely on a non-interactive commitment scheme Com, with domain $\{0,1\}^*$. We typically write $[\![v]\!] := \mathsf{Com}(v; r_v)$ for a commitment to value v using randomness $r_v \in \{0,1\}^*$. To open a given commitment $[\![v]\!]$, it suffices to reveal (v, r_v), so that a verifier can check that $[\![v]\!] = \mathsf{Com}(v; r_v)$. For the proof of security we need that $[\![v]\!]$ statistically hides the committed value v, and after publishing $[\![v]\!]$ it is computationally infeasible to open the commitment in two different ways. We follow [28] for the formal definitions.

We use the following standard NP relations: (i) R^{vc}, for validity of commitments; (ii) R^{oc}, for ownership of an opening; (iii) R^{zero^-} (resp. R^{zero^+}, R^-) for commitments to non-positive (resp. non-negative, negative) values; (iv) R^{ec}, for equality of two openings; (v) R_{nec}, for commitments to values different from a pre-defined constant.

c) Hybrid Ideal Functionalities.: To implement $\mathcal{F}_{\mathsf{CFM}}$ we use hybrid ideal functionalities, with the usual simulation-based proofs relying on the composition theorem [16].

All our functionalities receive some values/randomnesses and the corresponding commitments, and must first check whether the commitment actually corresponds to the claimed value, returning $\perp$ otherwise (as in R^{vc}). The remaining features outlined below are specific to our application. They are similar to range proofs [14], [15].

- The *Secure All Positive Check* functionality $\mathcal{F}_{\mathsf{pcheck}}$ receives from every trader the net position η_i and guarantees solvency (i.e., $\bigwedge_i \eta_i \geq 0$).
- The *Secure Sum Comparison* functionality $\mathcal{F}_{\mathsf{compare}}$ receives from every party a pair of old and new binary flags $\{f_i, f_i^*\}$. It checks whether the total number of flags has not changed (i.e., $\sum_i f_i = \sum_i f_i^*$).
- Finally, the zero-knowledge functionality $\mathcal{F}_{\mathsf{zk}}^R$ is parameterized by an NP relation R and receive inputs from a trader P_i in the role of a prover, while all other traders $\{P_j\}_{j \neq i}$ play the role of verifiers. As usual the prover sends the statement x_i and the corresponding witness w_i to the functionality, while each verifier sends its own statement x_j to be checked. Each verifier gets the outcome of $R(x_j, w_i)$ if $x_i = x_j$, otherwise it gets $\perp$. For simplicity we omit the zk subscript. MPC will be identified by subscripts and zk by superscripts.

The NP relations we use are summarized in Table VII. To describe them, we use some auxiliary values that are not needed in the ideal functionality $\mathcal{F}_{\mathsf{CFM}}$ (albeit they might well be present in an actual centralized exchange implementation). These additional values are defined in §VII and Table VI. Some of our relations directly test the requirements of Def. 1,2 and 5, whereas other relations are used to validate intermediate results in our protocol construction, and share similarities with the NP statement POUR in [7].

VII. Solution Overview

The first challenging part of the protocol construction is to identify a suitable form for the state of the reactive security functionality implementing $\mathcal{F}_{\mathsf{CFM}}$ that would account for its non-monotonic behavior in the legitimacy of traders and assets. A simple (but wrong) solution would be to use just the private inventory values of the individual traders. Each trader could prove in ZK that it respect the constraints stated in Def. 1, 2 and 5. Unfortunately, the arrival of new valid orders could make the constraints of some other trader invalid, i.e. the protocol should no longer consider her ZK proof valid.

TABLE VI: Futures Market Notation

Nota.	Description
ρ	Root of a Merkle tree
$path$	Authentication path of a token τ_i in a Merkle tree with root ρ
$\mathcal{O}_{\mathsf{buy}}$	Current range choices for long position trader to use in net position calculation, defined as $\{((p_1, V_{\mathsf{max}}), (p_1, V_1^{\mathsf{buy}})), ((p_1, V_1^{\mathsf{buy}}), (p_2, V_2^{\mathsf{buy}})) \ldots, ((p_{l_{\mathsf{buy}}}, V_{l_{\mathsf{buy}}}^{\mathsf{buy}}), (0,0))\}$
$\mathcal{O}_{\mathsf{sell}}$	Current range choices for short position trader to use in net position calculation, defined as $\{((0,0), (p_{l_{\mathsf{sell}}}, V_{l_{\mathsf{sell}}}^{\mathsf{sell}})), \ldots, ((p_{L-1}, V_{L-1}^{\mathsf{sell}}), (p_L, V_L^{\mathsf{sell}})), ((p_L, V_L^{\mathsf{sell}}), (p_L, V_{\mathsf{max}}))\}$
p_{lb}	Lower bound price used for net position calculation
V_{lb}	Lower bound cumulative volume used for net position calculation
p_{ub}	Upper bound price used for net position calculation
V_{ub}	Lower bound cumulative volume used for net position calculation
δ_c	Incremental value for the pending order counter

TABLE VII: Futures Market Relations

Relation	Additional Conditions
R^{token}	Token τ_i is correctly constructed from the inventory values, i.e. $\tau_i = \mathsf{Com}(m_i \|\| v_i \|\| \widehat{m_i} \|\| \widehat{v_i} \|\| c_i \|\| f_{\mathsf{bad},i} \|\| f_{\mathsf{del},i} \|\| f_{\mathsf{out},i}; r_i)$
R^{inv}	The new inventory values $m_i, v_i, \widehat{m_i}, \widehat{v_i}, c_i, f_{\mathsf{bad},i}, f_{\mathsf{del},i}, f_{\mathsf{out},i}$ are correctly constructed from an old inventory (with token τ_i'), i.e. $\mathsf{Auth}(\rho, path_i, [\![\tau_i']\!]) = 1$; $\tau_i' = \mathsf{Com}(m_i \|\| v_i \|\| \widehat{m_i} \|\| \widehat{v_i} \|\| c_i \|\| f_{\mathsf{bad},i} \|\| f_{\mathsf{del},i} \|\| f_{\mathsf{out},i}; r_i')$
R^{uinv}	The new inventory values $\widehat{m_i}^*, \widehat{v_i}^*, c_i^*$ are correctly updated from an old inventory (w.r.t. δ_c, l, v), i.e. $\widehat{m_i}^* = \widehat{m_i} - \delta_c \cdot p_l \cdot v$; $\widehat{v_i}^* = \widehat{v_i} + \delta_c \cdot v$; $c_i^* = c_i + \delta_c$
R^{rng}	The upper and lower bounds of cumulative volumes and prices $p_{\mathsf{lb}}, V_{\mathsf{lb}}, p_{\mathsf{ub}}, V_{\mathsf{ub}}$ are correctly selected from the $\mathcal{O}_{\mathsf{buy}}$ or $\mathcal{O}_{\mathsf{sell}}$, i.e. $V_{\mathsf{lb}} \leq \lvert v \rvert \leq V_{\mathsf{ub}}$ and one of the following holds: $v > 0 \wedge ((p_{\mathsf{lb}}, V_{\mathsf{lb}}), (p_{\mathsf{ub}}, V_{\mathsf{ub}})) \in \mathcal{O}_{\mathsf{buy}}$ or $v < 0 \wedge ((p_{\mathsf{lb}}, V_{\mathsf{lb}}), (p_{\mathsf{ub}}, V_{\mathsf{ub}})) \in \mathcal{O}_{\mathsf{sell}}$ or $v = 0 \wedge (p_{\mathsf{lb}} V_{\mathsf{lb}}) = (p_{\mathsf{ub}}, V_{\mathsf{ub}}) = (0,0)$
R^{net}	(Estimation) of an instant net position η_i (resp. $\widehat{\eta_i}$) are correctly computed, i.e. $\eta_i = m_i + p_{\mathsf{lb}} \cdot V_{\mathsf{lb}} + p_{\mathsf{ub}} \cdot (\lvert v_i \rvert - V_{\mathsf{lb}})$
R^{match}	The order fulfillment is correctly done, i.e. $m_i^* = m_i - p_l \cdot v$; $v_i^* = v_i + v$; $c_i^* = c_i + \delta_c$
R^{flags}	The transition from the flags $(f_{\mathsf{bad},i}, f_{\mathsf{del},i}, f_{\mathsf{out},i})$ to the flags $(f_{\mathsf{bad},i}^*, f_{\mathsf{del},i}^*, f_{\mathsf{out},i}^*)$ is consistent with the values (η_i^*, v_i^*, c_i^*), as shown in the diagram in Fig. 4
R^{mtm}	A trader P_i is correctly marked to market, i.e. $m_i^* = m_i + \bar{p} \cdot v_i$

We must also store such a state in a way that after an order is accepted by the market (i.e. the ensemble of agents) it is not possible to link it to the next order by the same trader. This cannot just be the union of the individual (unopened) inventories. By looking at the unchanged inventories the traders could identify the trader who did the order. A global MPC step to update the entire state would be a solution but it would put an unnecessary burden on the other traders. A further challenge is that we must keep a fully *ordered* list (Matching orders must be executed according to arrival time).

First, we *augment* the private state of each trader with additional information besides the inventory m_i and v_i. We *memoize* the value of the estimation $\widehat{m_i}$ and $\widehat{v_i}$, and a counter c_i to track the number of pending orders. Each time a trader P_i posts an order (ℓ, v), the memoized values are updated as $\widehat{m_i} = m_i - p_\ell \cdot v$, $\widehat{v_i} = v_i + v$, and $c_i = c_i + \delta_c$ where $\delta_c = 1$. For order cancellations, or complete matches of pending orders, the reverse computation is performed ($\delta_c = -1$). The use of memoized values is a quick calculation to do and to verify cryptographically. Yet, the foremost reason for such device is that the values $\widehat{m_i}, \widehat{v_i}$ of a

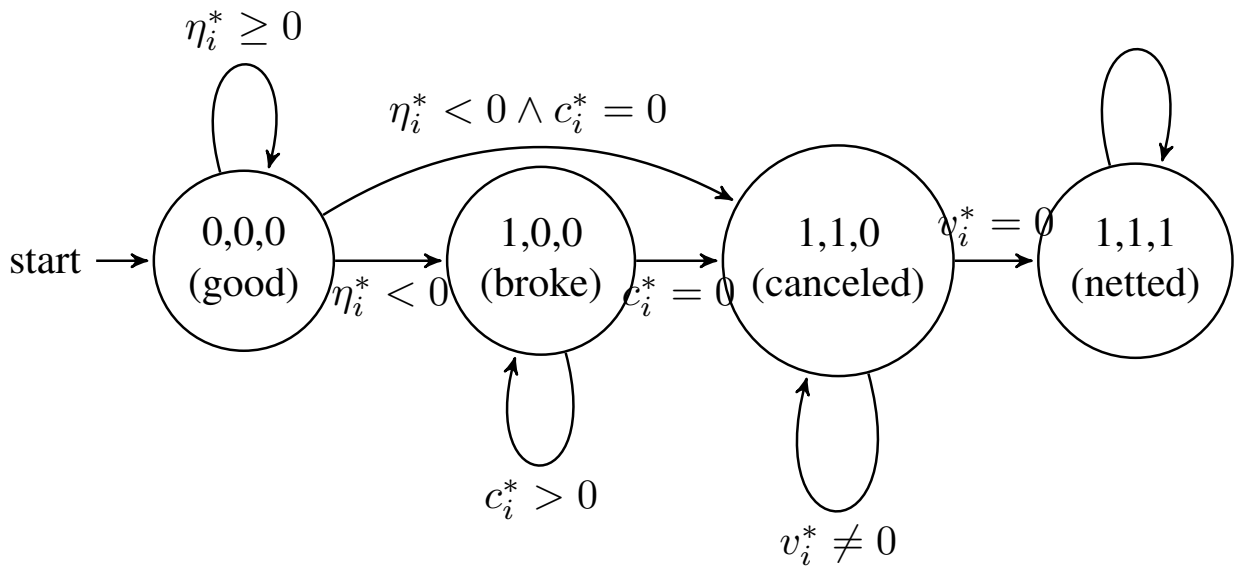


Fig. 4: Inventory flags state transition diagram

trader are needed to *prevent the linking of limit orders during the verification procedure*, while allowing the instantaneous computation of $\widehat{\eta_i}$.

Memoization avoids the use of MPC when addressing the conflicting requirements of i) providing a public trail of events, ii) publicly verifying a constraint on a private subset of such events as well as iii) showing that such private events are *all and only* applicable events. To meet (iii) Alice would have had to show which orders belonged to her to add them to her estimated net position. Since the full order book is visible (i), her full trading strategy would then be visible to the other players. In contrast, if we make sure that an order is private to trader Bob (ii), this very property does not allow Alice to prove that the order in question does not belong to her (and does not make her over budget), so failing (iii). A full MPC protocol would be a solution but this would force other traders to participate to the posting of any order from a third party. As mentioned, such burden would be considered unacceptable.

Next we introduce three flags to represent the status as a potentially broke trader. A trader's inventory is marked with a state represented by the three flags $f_{\mathsf{bad},i}, f_{\mathsf{del},i}, f_{\mathsf{out},i}$. We call an inventory with a non-negative instant net position a *good* inventory ($f_{\mathsf{bad},i}$=0, $f_{\mathsf{del},i}$=0, $f_{\mathsf{out},i}$=0), otherwise it is a *broke* inventory ($f_{\mathsf{bad},i}$=1, $f_{\mathsf{del},i}$=0, $f_{\mathsf{out},i}$=0). A good trader can do a normal post/cancel action, while a broke trader has to cancel a previous order in the **Margin Settlement** phase. Finally, we call an inventory *canceled* if it is a *broke* inventory with no pending order (after canceling all orders in the **Margin Settlement** phase) at the time of commitment ($f_{\mathsf{bad},i}$=1, $f_{\mathsf{del},i}$=1, $f_{\mathsf{out},i}$=0); an inventory is, instead, *netted* if it has a zero volume holding (after matching to an offset position during **Margin Settlement**) at the time of commitment ($f_{\mathsf{bad},i}$=1, $f_{\mathsf{del},i}$=1, $f_{\mathsf{out},i}$=1). The state transition diagram in Fig. 4 shows how the inventory switches from one state to another, as well as the condition causing the transition. This status will be key to capture the non-monotonic evolution of the validity of commitments and zk proofs once a valid order (of another trader) is accepted.

The overall state is then captured by a token τ_i that is a commitment of all values in the inventory (with fresh randomness r_i); initially, such value is only known to the trader itself. Each trader keeps the token secret and broadcasts a commitment to it in order to commit to a new inventory; such an inventory is considered as *unspent*. At a later point, a trader can reveal the token and retrieve a previously committed inventory, in which case we say the inventory is *spent*, as the corresponding token cannot be used anymore.

The anonymity of the inventory is guaranteed by relying on Merkle trees [44] in conjunction with the zero-knowledge proofs (as in [55]). Throughout the execution of the protocol, a Merkle tree $\mathcal{T}$ based on a collision-resistant hash function $\mathsf{H}: \{0,1\}^* \rightarrow \{0,1\}^*$, where the leafs are commitments, is maintained and updated. ρ denotes the root of the tree, and $path$ denotes the authentication path from a leaf $[\![v]\!]$ to

TABLE VIII: Merkle Tree's supported operations

Definition	Description
$\rho = \mathsf{Add}(\mathcal{T}, [\![v']\!])$	Adds a new leaf (the hash of $[\![v']\!]$) to the tree and generates a new root ρ.
$path = \mathsf{Path}(\mathcal{T}, [\![v']\!])$	Returns the authentication path from $[\![v']\!]$ to ρ.
$\{0,1\} \leftarrow \mathsf{Auth}(\rho, path, [\![v]\!])$	Authenticates $[\![v]\!]$ in $\mathcal{T}$ w.r.t. the authentication path $path$ (where output 1 means the authentication succeeded).

the root ρ. As in [55], [7], the number of leafs is not fixed a-priori, one can efficiently update a Merkle tree $\mathcal{T}$ by appending a new leaf, resulting in a new tree $\mathcal{T}'$ with root ρ; this can be done in time/space proportional to tree depth. Table VIII summarizes the supported ops Add, Path and Auth of a Merkle tree $\mathcal{T}$.

Preserving Traders' Anonymity. The commitment (the retrieval) of trader inventories to the Merkle Tree $\mathcal{T}$ is obtained by running a sub-protocols Π_{put} (resp. Π_{get}) as follows:

- Executing protocol Π_{put}, the trader broadcasts a commitment to the token corresponding to the current inventory:
  $\tau_i = \mathsf{Com}(m_i||v_i||\widehat{m_i}||\widehat{v_i}||c_i||f_{\mathsf{bad},i}||f_{\mathsf{del},i}||f_{\mathsf{out},i}; r_i)$
  Thus, the trader proves that the token is correctly constructed (using $\mathcal{F}^{\mathsf{token}}$) and appended into the Merkle tree $\mathcal{T}$ (with operation Add), before broadcasting the new root of the tree. The other traders will check that the new root is correctly computed before accepting it.
- In an execution of protocol Π_{get}, a trader can retrieve a previously committed inventory (say, at round $t' < t$), and *spend* it for posting or canceling an order (l, v), by revealing the secret *unspent* token τ_i' and proving that the newly committed values are consistent updates of the values committed at round t'; this is done using $\mathcal{F}^{\mathsf{inv}}$ (to retrieve the inventory) and then $\mathcal{F}^{\mathsf{uinv}}$ (to update the inventory); in particular, $[\![\tau_i']\!]$ is a leaf of the current tree and $m_i = m_i'$, $v_i = v_i'$, $\widehat{m_i} = \widehat{m_i}' - \delta_c \cdot p_l \cdot v$, $\widehat{v_i} = \widehat{v_i}' + \delta_c \cdot v$, and $c_i = c_i' + \delta_c$, while all the flags $f_{\mathsf{bad},i}, f_{\mathsf{del},i}, f_{\mathsf{out},i}$ stay the same. Every time an inventory is retrieved, two sets of commitments are generated corresponding to the inventory values before and after the update. The token τ_i' is now marked as *spent* and will not be usable for retrieving any inventory.

The main Merkle tree $\mathcal{T}$ can also be forked (via sub-protocol Π_{backup}, see below) into a backup tree $\mathcal{T}_{\mathcal{U}}$ to use during the **Mark to Market** phase in case there are still traders with a negative net position even after the **Margin Settlement** phase. We use this feature to challenge the non-monotonicity of security and go beyond security-with-abort.

VIII. Protocol Construction

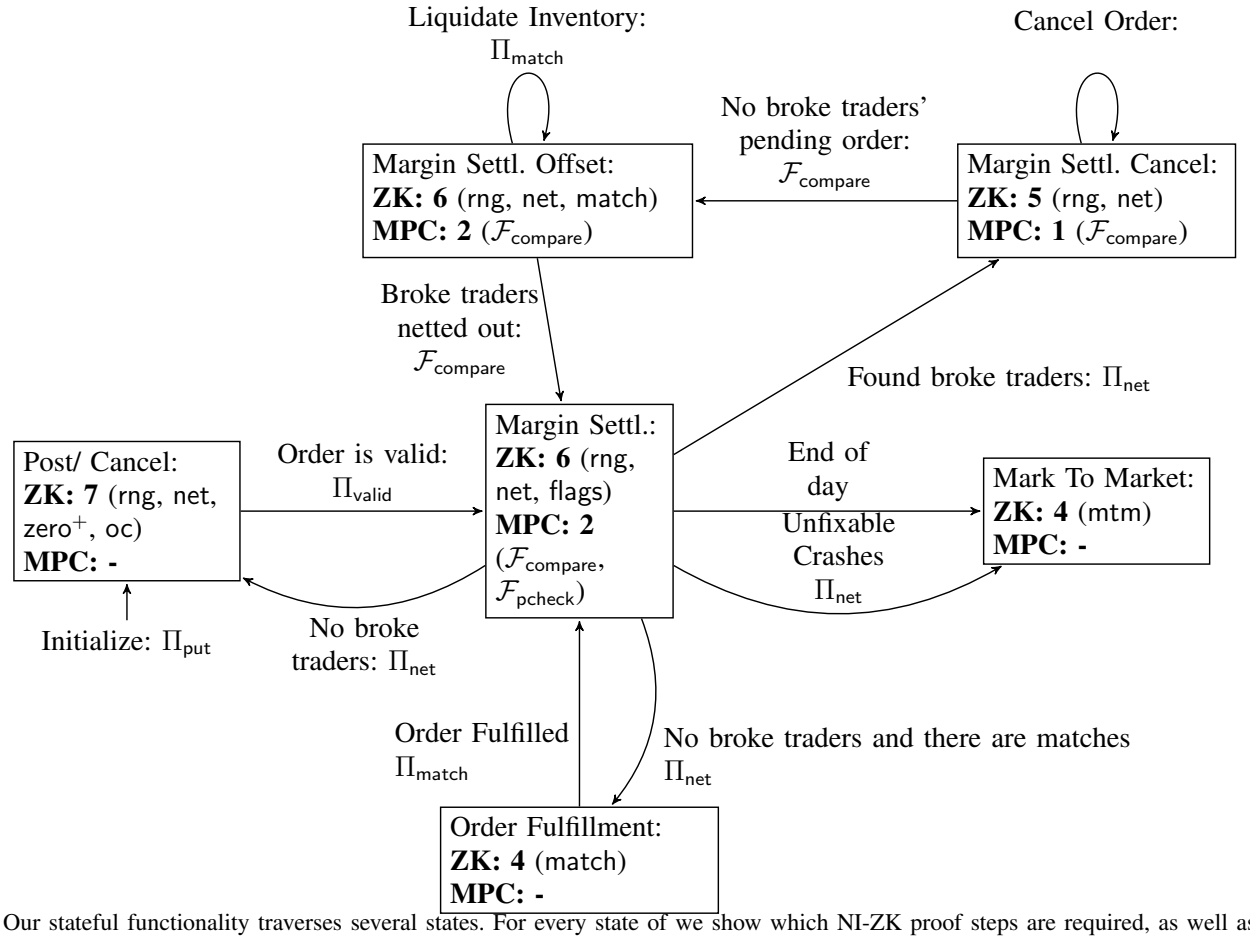
At this point an easy solution would be to just run the *entire* reactive functionality as a global MPC. As we mentioned, this would be unacceptable from the perspective of most traders: the burden of computation should be shifted to parties wishing to prove something (e.g. their good and bad standing). As such, only when global consistency of the market is at stake should all traders be involved. We illustrate this empirically in §XII.

Fig. 5 summarizes how the various security functionalities and sub-protocols have been used to implement each step of the global ideal functionality.

First we present a few useful sub-protocols that are extensively used during different phases of our main protocol.

a) Common Sub-Protocols: The protocols in Fig. 6 below are used extensively as sub-routines in our main protocol.

We denote by the superscript $\cdot^*$ the updated values, computed locally by P_i after an update of the order book, e.g. m_i^*, which are used as common inputs for the sub-protocols. We use it in particular on the commitments of the inventory values, e.g. $[\![m_i]\!]$, the related order information (δ_c, l, v) and the the Merkle Tree $\mathcal{T}$, where P_i additionally holds the committed values and the corresponding randomnesses.



Our stateful functionality traverses several states. For every state of we show which NI-ZK proof steps are required, as well as the MPC steps. The subprotocols Π_{get}, Π_{put} and their functionalities inv, uinv, token are always needed to interact with the trader's inventory.

Fig. 5: Hybrid Implementation of the Ideal Functionality

Π_{valid}: Every time a trader P_i posts or cancels an order, say (δ_c, l, v), the protocol has to check for its validity.

Π_{net}: Every time the order book is updated via (i) a post/cancel action of a trader P_i, or (ii) a match of two traders P_i and $P_{i'}$, all the traders (including P_i and $P_{i'}$), need to be checked for negative instant net position.

Π_{match}: for matching an order between P_i and $P_{i'}$.

Π_{backup}: fork a backup tree $\mathcal{T}_{\mathcal{U}}{}^*$ from the main tree $\mathcal{T}^*$ to serve as a starting point during the **Mark to Market** phase in the case there are still traders with a negative net position even after **Margin Settlement**. The protocol runs with the commitments of the inventory values, as well as the new net position of all traders, as the common inputs.

b) Protocol Description: The overall protocol runs in 4 phases, which we describe on a high-level below. Formal description can be found in §A of the appendix.

Initialization Every trader participating in the futures market has to commit to a valid initial inventory. This is done during the first round, by each trader individually, as follows.

- P_i holds an initial non-negative secret amount of cash (i.e., $m_i \geq 0$), zero volume holding (i.e., $v_i = 0$), an initial estimation of the cost to pay for pending orders (i.e., $\widehat{m_i} = m_i$), and an zero estimation of the volume holding for pending orders (i.e., $\widehat{v_i} = 0$) as well as all initial zero inventory flags.
- P_i commits to its initial inventory and proves in zero knowledge that such an inventory is valid (as defined above), using the functionalities $\mathcal{F}^{\mathsf{zero}^+}$ for m_i and $\mathcal{F}^{\mathsf{ec}}$ $\widehat{m_i}$; while simply decommitting v_i and $\widehat{v_i}$ and the flags is sufficient to prove that they are all zeros.
- The traders run protocol Π_{put} to commit the inventory of P_i; the backup tree $\mathcal{T}_{\mathcal{U}}$ is initially identical

Sub-protocol Π_{valid} is run by $(P_1, \ldots, P_N)$ in order to let P_i prove a valid **Post Order** or **Cancel Order** action w.r.t. (δ_c, l, v).

1) In case of **Cancel Order**, P_i sends $(P_i, [\![i]\!], (i, r))$ to $\mathcal{F}^{\mathsf{oc}}$, while $P_{j\neq i}$ sends $(P_j, [\![i]\!])$ for the ownership of the order.
2) All traders run Π_{get} then P_i proves that s/he can perform the action by decommitting the inventory flags to show:
   a) $f_{\mathsf{bad},i} = 0$ in a normal post/cancel action;
   b) $(f_{\mathsf{bad},i} = 1) \wedge (f_{\mathsf{del},i} = 0)$ for a cancel action during **Margin Settlement**;
   c) $(f_{\mathsf{bad},i} = 1) \wedge (f_{\mathsf{del},i} = 1) \wedge (f_{\mathsf{out},i} = 0)$ for a post action during **Margin Settlement**.
3) P_i proves it has a non-negative estimation for instant net position $\widehat{\eta}_i$ (only in a normal post/cancel action):
   a) Broadcast $([\![p_{\mathsf{lb}}]\!], [\![V_{\mathsf{lb}}]\!], [\![p_{\mathsf{ub}}]\!], [\![V_{\mathsf{ub}}]\!])$ and send $(P_i, x_i^{\mathsf{rng}}, w_i^{\mathsf{rng}})$ to $\mathcal{F}^{\mathsf{rng}}$, while $P_{j\neq i}$ sends $(P_j, x_j^{\mathsf{rng}})$.
   b) Broadcast $[\![\widehat{\eta}_i{}^*]\!]$ and send $(P_i, x_i^{\mathsf{net}}, w_i^{\mathsf{net}})$ to $\mathcal{F}^{\mathsf{net}}$, while $P_{j\neq i}$ sends $(P_j, x_j^{\mathsf{net}})$.
   c) Send $(P_i, [\![\widehat{\eta}_i{}^*]\!], \widehat{\eta}_i{}^*)$ to $\mathcal{F}^{\mathsf{zero}^+}$, while $P_{j\neq i}$ sends $(P_j, [\![\widehat{\eta}_i{}^*]\!])$.
4) All traders run Π_{put}.

Sub-protocol Π_{net} is run for checking new broke traders.

1) Repeat the following for each trader P_i to retrieve the inventory then update and check their new inventory flags.
   a) All traders run Π_{get}.
   b) Broadcast $([\![p_{\mathsf{lb}}]\!], [\![V_{\mathsf{lb}}]\!], [\![p_{\mathsf{ub}}]\!], [\![V_{\mathsf{ub}}]\!])$ and send $(P_i, x_i^{\mathsf{rng}}, w_i^{\mathsf{rng}})$ to $\mathcal{F}^{\mathsf{rng}}$, while $P_{j\neq i}$ sends $(P_j, x_j^{\mathsf{rng}})$.
   c) Broadcast $[\![\eta_i^*]\!]$ and send $(P_i, x_i^{\mathsf{net}}, w_i^{\mathsf{net}})$ to $\mathcal{F}^{\mathsf{net}}$, while $P_{j\neq i}$ sends $(P_j, x_j^{\mathsf{net}})$.
   d) Broadcast $[\![f_{\mathsf{bad},i}^*]\!], [\![f_{\mathsf{del},i}^*]\!], [\![f_{\mathsf{out},i}^*]\!]$ and send $(P_i, x_i^{\mathsf{flags}}, w_i^{\mathsf{flags}})$ to $\mathcal{F}^{\mathsf{flags}}$, while $P_{j\neq i}$ sends $(P_j, x_j^{\mathsf{flags}})$.
   e) Forward $f_{\mathsf{bad},i}$ and $f_{\mathsf{bad},i}^*$ to $\mathcal{F}_{\mathsf{compare}}$.
2) If $\mathcal{F}_{\mathsf{compare}}$ returns 1, all traders run Π_{put} (with $\mathcal{T}^*$) then run Π_{backup}.
3) Otherwise, the traders discard $\mathcal{T}^*$ and, for each trader P_i, all traders run Π_{put} (with $\mathcal{T}$).

Sub-protocol Π_{match} for updating the inventories upon a match of orders $(t, l, [\![i]\!], v)$ of P_i and $(t', l, [\![i']\!], v')$ of $P_{i'}$ $(v \cdot v' < 0)$.

1) P_i sends $(P_i, [\![i]\!], (i, r))$ to $\mathcal{F}^{\mathsf{oc}}$, while $P_{j\neq i}$ sends $(P_j, [\![i]\!])$, then all traders run Π_{get}.
2) In case $v > 0$, and for $\delta = \mathsf{min}(|v|, |v'|)$, P_i computes $m_i^* = m_i - p_l \cdot \delta$ and $v_i^* = v_i + \delta$.
3) (If $v < 0$, replace δ with $-\delta$. If $\delta = |v|$, let $c_i = c_i + \delta_c$ where $\delta_c = -1$.)
4) P_i broadcasts $[\![m_i^*]\!]$, $[\![v_i^*]\!]$, $[\![\widehat{m_i}^*]\!]$, $[\![\widehat{v_i}^*]\!]$, and $[\![c_i^*]\!]$. and sends $(P_i, x_i^{\mathsf{match}}, w_i^{\mathsf{match}})$ to $\mathcal{F}^{\mathsf{match}}$, while $P_{j\neq i}$ sends $(P_j, x_j^{\mathsf{match}})$.
5) All traders run Π_{put}.
6) $P_{i'}$ performs steps 1-5 (where $\delta := -\delta$, $i := i'$, and $\delta_c = -1$ if $\delta = |v'|$).

Sub-protocol Π_{backup} is run to fork a backup tree. Additionally, the common inputs include $[\![\eta_i^*]\!]$, and P_i also holds η_i^*.

1) P_i forwards η_i^* to $\mathcal{F}_{\mathsf{pcheck}}$.
2) If $\mathcal{F}_{\mathsf{pcheck}}$ returns 1, all traders run Π_{put} to obtain $\mathcal{T}_{\mathcal{U}}{}^*$.

Fig. 6: Sub-protocols Π_{valid}, Π_{net}, Π_{match} and Π_{backup}

to $\mathcal{T}$.

Post/Cancel Order A good trader can post a new order $(\delta_c = 1, l, v)$ or cancel a previous order $(\delta_c = -1, l', v')$.

- The traders run Π_{valid}.
- The traders run Π_{net}; this can lead to **Mark to Market**.
- The traders run Π_{match} for each match in the order book (only for **Post Order**).
- After the match, all traders run Π_{net} again.

Margin Settlement This phase is (re) started every time there is at least one new trader with a bad standing (i.e., $f_{\mathsf{bad},i} = 0$ and $\eta_i^* < 0$). It proceeds as described below; afterwards the protocol goes back to the previous phase (whatever it was).

- For each pending order (t', l', i, v') of a broke trader P_i:
  - The traders run Π_{get} with parameters $(-1, l, v)$, to retrieve P_i's two inventories: one before the cancellation of the pending order and one after that.

- The traders run Π_{valid}.
- All traders forward the necessary flags $f^*_{\mathsf{del},i}$ and $f_{\mathsf{bad},i}$ to $\mathcal{F}_{\mathsf{compare}}$ to check whether $\sum f^*_{\mathsf{del},i} = \sum f_{\mathsf{bad},i}$. If the check is successful, move to next step.

- The traders run Π_{net} to check and restart this phase if there are new broke traders.
- The broke traders offset their positions until all broke traders are netted out.
  - The traders run Π_{get} to retrieve their inventory.
  - The traders find matches on the order book at the current best price, say between P_i and $P_{i'}$; both traders locally update their inventory, commit to the new inventory, and prove in zero knowledge that the matching was done correctly (using $\mathcal{F}^{\mathsf{match}}$).
  - All traders forward the necessary flags $f^*_{\mathsf{out},i}$ and $f_{\mathsf{bad},i}$ to $\mathcal{F}_{\mathsf{compare}}$ to check whether $\sum f^*_{\mathsf{out},i} = \sum f_{\mathsf{bad},i}$, if the check is successful, go to next step.
- The traders run Π_{net} to check and restart this phase if there is a new broke trader.
- All traders run Π_{backup} to check if a backup tree can be forked, if not, go to **Mark to Market** phase.

Mark to Market This phase is invoked at the last round $t = T$, or during **Margin Settlement**.

- The traders run Π_{get} to retrieve their inventory.
- The traders locally update their inventory, commit to the new inventory, and prove in zero knowledge that the matching was done correctly (using $\mathcal{F}^{\mathsf{mtm}}$).
- Finally the new inventory is added back to the Merkle tree $\mathcal{T}$ by running Π_{put}.

The *Proportional Burden* is fulfilled for what is technically possible as we require traders posting/canceling an order to prove the validity of their actions before other traders prove the validity of their inventories according to the new order book. The latter is necessary for distributed risk management. It could be optimized by having a trader proving the validity of an inventory for a range of price values rather than just the current price (e.g. up/downward ticks as appropriate).

IX. Security Analysis (Sketch)

The theorem below states the security of protocol Π_{DFM}.

Theorem 1. *Let* Com *be a statistically hiding (and computationally binding) commitment scheme. Protocol* Π_{DFM} *securely realizes the ideal functionality* $\mathcal{F}_{\mathsf{CFM}}$ *in the* $(\mathcal{F}_{\mathsf{zk}}, \mathcal{F}_{\mathsf{compare}}, \mathcal{F}_{\mathsf{pcheck}})$*-hybrid model, where the zero-knowledge functionality* $\mathcal{F}_{\mathsf{zk}}$ *supports all* NP *relations defined in* §*VI.*

We sketch here the key step of the security proof (See Appendix B for details). As in standard simulation-based security proofs, we exhibit an efficient simulator interacting with the ideal functionality $\mathcal{F}_{\mathsf{CFM}}$ that is able to fake the view of any efficient adversaries corrupting a subset $I \subseteq [N]$ of the traders in an execution of protocol Π_{DFM}.[5].

Our protocol is designed in a "hybrid world" with several auxiliary ideal functionalities (mainly for zero-knowledge proofs and for running secure comparisons). Importantly, in such a world, there is no security issue when using these functionalities: a composition theorem ensures that our protocol is still secure when we replace the auxiliary ideal functionalities with sub-protocols securely realizing them. An advantage of working in the hybrid model is that the simulator gets to see the inputs that corrupted traders forward to the auxiliary ideal functionalities in the clear.

On a very high level, our simulator $\mathcal{S}$ works as follows. During the **Initialize** phase, it commits to zero values for each commitment forwarded by a honest trader in the real protocol; the commitments to the token of each inventory are added to a simulated Merkle Tree that is maintained internally by the simulator. During a **Post/Cancel Order** action, it relies on the ideal functionality $\mathcal{F}_{\mathsf{CFM}}$ to post/cancel the corresponding orders; afterwards, in the **Margin Settlement** phase, for each match notification received from the ideal functionality $\mathcal{F}_{\mathsf{CFM}}$, the simulator commits to zero for each commitment forwarded by a

[5]We assume the set I is fixed before the protocol execution starts.

honest trader in the real protocol execution. During the **Mark to Market** phase, it commits to zero values for each commitment forwarded by a honest trader in the real protocol.

The hiding property of the commitment scheme implies that the above simulation is indistinguishable to the view generated in a mental experiment where the simulator $\mathcal{S}$ is given the real inputs corresponding to each honest trader. The only difference between this mental experiment and a real protocol execution is that in the former experiment the market evolves using the inventories held at the beginning by each corrupted trader, whereas in the latter experiment the adversary can try to cheat and fake the inventory of a corrupted trader (e.g., by claiming an order pertaining to a honest trader). However, the binding property of the commitment scheme and the collision resistance of the Merkle Tree, ensure that such cheating attempts only succeed with a negligible probability.

This allows us to conclude that the view simulated in the ideal world (with the functionality $\mathcal{F}_{\mathsf{CFM}}$) is computationally indistinguishable from the view in a real execution of the protocol, thus establishing the security of Π_{DFM}.

X. Beyond Security-With-Abort

If every single party participates to the computation, the baseline protocol is secure. However, an adversary can refuse to join $\mathcal{F}_{\mathsf{compare}}$ or $\mathcal{F}_{\mathsf{pcheck}}$, or to match an order in Π_{match}, or refuse to cancel pending orders and liquidate her own inventory during the **Margin Settlement** phase. Therefore, if even a single party is byzantine and aborts, the base protocol cannot continue operating, leading to a clear scalability issue.

A preliminary observation is that in practice one cannot initialize a market with a self-claimed account. The cash that get deposited into the market must be backed by a verifiable source where a debit is acknowledged by every market participants, for instance ZeroCash. Hence, such source must be able to publicly verify the validity of the transactions resulting from the market's operation at the end of the day to credit each the account with the corresponding amount.

An approach is to penalize a faulty participant upon aborting in an MPC, hence make the adversary lose some digital cash in proportion to their actions. For instance, [36] and [37] require the adversary to make deposits and forfeit them upon dropping out. Unfortunately those protocols are not usable in our scenario. Technically the parties have to move *in a fixed order* since *order of revelation is important* (the see-saw mechanism, [36, p. 7]) for the aforementioned penalty mechanism to work. This fixed order conflicts with our protocol's anonymity requirement since this will reveal the identity of the trader who made a posting. Most importantly, those protocols are *not economically viable* as the baseline deposit would need to be progressively staggered in a see-saw fashion which is unachievable due to the anticipated variety in financial capability of traders. In a low-frequency market the trader going first would have to deposit assets 35x times the stake of the trader going last, and in large markets that increases to 500 times larger (See Table II, where a single Eurodollar contract has a notional value of 1M dollars and margins are measured in basis points).

Hence we opt towards the mechanism of Hawk [34, Appendix G, §B] in which private deposits are frozen and the identified aborting parties cannot claim the deposits back in the withdraw phase. This fits precisely with our scenario as the deposit can be made to match the initial margin (which is the largest amount[6] a trader can lose when being netted out).

At first we must show that honest participants can eventually move by themselves to a **Mark To Market** phase at least to cash their own inventory. Let us denote by Adv the set of adversaries who abort between time t and time $t+1$ given a backup tree $\mathcal{T}_{\mathcal{U}}$ with a solvable inventory of all traders (m_i, v_i) and the corresponding mid price $\bar{p}$. Since $\mathcal{T}_{\mathcal{U}}$ is a valid tree, it satisfies the constraints from Def. 1 & 2, and therefore $\eta_i \geq 0$. Since $\eta_i = m_i + \mathsf{cash}(v_i) \leq m_i + \bar{p} \cdot v_i$, we have

$$0 \leq \sum\nolimits_{i \notin Adv} (m_i + \bar{p} \cdot v_i) \leq \sum\nolimits_i m_i^0$$

[6] In practice traders can deposit additional funds when receiving a margin call. These incremental deposits could be easily incorporated into our setting by querying the deposit ledger for all deposited funds before time $t < T$ as opposed to checking the single deposited value at time 0.

This implies that there will be no unexpected loss to cover ($0 \leq \ldots$) nor additional money would be created ($\ldots \leq \sum_i m_i^0$). Then honest traders can proceed to the **Mark To Market** phase to split their own trade proceedings. If enough traders accept the move so that it ends into the public ledger this would be considered an acceptable solution. From an economics perspective, the adversary would be penalized with at least its initial cash margin which could be substantial.

Now we just need to extend our protocol to identify the aborting parties in various protocol steps and prevent them from claiming the deposit in **Mark To Market** phase by requiring each trader to present a proof of participation in the round where the abort happens. A further step can possibly be taken to divide the money of the adversary if at the end of the **Initialize** phase the total sum of money is computed (by an MPC protocol, i.e. $\mathcal{F}_{\mathsf{sum}}$ that receives m_i from each trader and computes $\sum_i m_i$). In the **Mark To Market** phase, by computing the sum of money of the honest traders after the updates of the inventories we can find the difference corresponding to the money of the adversary and shares it by updating the inventories again adding the shares.

Formally, in an abort, every honest traders maintain a set of *spent* tokens τ_i' of the participants. The **Mark To Market** phase now runs exactly the same as before except that a trader must prove in zk s/he knows the opening to a token τ_i' that was present in the last step with the relation R^{oinv} which takes as input the statement $x_i^{\mathsf{oinv}} = (\tau_i')$ and witness $w_i^{\mathsf{oinv}} = (r_i', m_i, v_i, \widehat{m_i}, \widehat{v_i}, c_i, f_{\mathsf{bad},i}, f_{\mathsf{del},i}, f_{\mathsf{out},i})$. The output of R^{oinv} is defined to be one if and only if $\tau_i' = \mathsf{Com}(m_i||v_i||\widehat{m_i}||\widehat{v_i}||c_i||f_{\mathsf{bad},i}||f_{\mathsf{del},i}||f_{\mathsf{out},i}; r_i')$. In the simplest settings, as in a joint step, the set of tokens from participants can be easily constructed. We discuss the disqualification of an adversary in a more complex case, where s/he refuses to match an order o':

1) All traders P_i cancel all orders o with $o \neq o'$.
2) All traders P_i prove they do not own o' by decommitting $[\![c_i]\!]$ and showing that $c_i = 0$.

Similarly, if the **Margin Settlement** phase is aborted,

1) In cancellation phase, all traders retrieve their inventory with a token τ_i', and prove that the inventory flags is not *broke*, i.e $(f_{\mathsf{bad},i}, f_{\mathsf{del},i}, f_{\mathsf{out},i}) \neq (1, 0, 0)$.
2) In liquidation phase, all traders retrieve their inventory with a token τ_i', and prove that the inventory flags is not *canceled*, i.e $(f_{\mathsf{bad},i}, f_{\mathsf{del},i}, f_{\mathsf{out},i}) \neq (1, 1, 0)$.

XI. Implementation

All phases of our protocol have been implemented and optimized. For the anonymous network and the distributed consensus protocol, off-the-shelf implementations will do[7].

a) Implementation of Components.: We use a digital cash network that supports a private payment scheme, e.g. zcash[8] for bootstrapping the market's initial cash. We extend zcash's POUR transaction to accept one more input/output: the commitment $[\![m_i]\!]$, to deposit and withdraw the market's digital cash from and back to the zcash network.

We follow [7] in the instantiation of our building blocks, at a security level of 128 bits. Let $\mathsf{H}(\cdot)$ be a collision-resistant compression hash function that maps l bits input ($l \geq 512$) into 256 bits output (e.g. SHA256). We use $\mathsf{H}(\cdot)$ to instantiate the commitment scheme Com and also the hash function for the binary Merkle Tree $\mathcal{T}$. The zero-knowledge functionality $\mathcal{F}^R$ is instantiated with zk-SNARKs[9] for arithmetic circuit satisfiability [10], while generic MPC is used for the hybrid ideal functionalities $\mathcal{F}_{\mathsf{compare}}$ and $\mathcal{F}_{\mathsf{pcheck}}$.

Our zk code is based on the libsnark[10] library. We split $\mathcal{F}^{\mathsf{inv}}$ into $\mathcal{F}^{\mathsf{invm}}$ to first check whether the token is one of the leaf of the Merkle Tree, and then $\mathcal{F}^{\mathsf{invt}}$ to check consistency of new commitments and old

[7]We use a distributed ledger, e.g. HyperLedger in PBFT mode (https://www.hyperledger.org) as a byzantine fault tolerant storage for each protocol step, i.e. each broadcast is replaced with a write into the distributed ledger. To communicate anonymously, the traders hide behind a Tor network. While we mention Tor, zcash, and HyperLedger in our implementation, we can replace any sub-protocol with other protocols for the same task, without affecting security. See [2], [59] for a comparison between different solutions.

[8]https://z.cash.

[9]While SNARKs are problematic in the setting of universal composability [35], they are still sufficient for sequential composition.

[10]https://github.com/scipr-lab/libsnark.

tokens[11]. Our MPC uses the SPDZ[12] library, along the construction in [11], [24], [49].[13]

b) Optimization: We also experimented with an optimized version to reduce the cost to only 30% and overcome the limit of 10 traders for the MPC. To improve the performance of the protocol we first streamline the number of the validations of the commitments by packing $f_{\mathsf{bad},i}$, $f_{\mathsf{del},i}$, $f_{\mathsf{out},i}$ into a single integer f_i, this improves the circuit $\mathcal{F}^{\mathsf{token}}$, $\mathcal{F}^{\mathsf{invt}}$ as well as $\mathcal{F}^{\mathsf{flags}}$. Then we can combine the circuits $\mathcal{F}^{\mathsf{invt}}$ and $\mathcal{F}^{\mathsf{uinv}}$, $\mathcal{F}^{\mathsf{rng}}$ and $\mathcal{F}^{\mathsf{net}}$. Proof generations can also be parallelized in a protocol step, e.g. in Π_{valid}, $\mathcal{F}^{\mathsf{invt}}$, $\mathcal{F}^{\mathsf{uinv}}$, $\mathcal{F}^{\mathsf{rng}}$, $\mathcal{F}^{\mathsf{net}}$, $\mathcal{F}^{\mathsf{zero}^+}$ and $\mathcal{F}^{\mathsf{token}}$ are independent of each other.

Further, functionalities $\mathcal{F}_{\mathsf{compare}}$ and $\mathcal{F}_{\mathsf{pcheck}}$ are used extensively throughout our protocol. As we shall see in Table X, the consistency check of the commitments slow down the whole protocol. We can replace $\mathcal{F}_{\mathsf{compare}}$ with a lighter functionality $\mathcal{F}_{\mathsf{dtc}}$ below to detect the flag in an unwanted state (without validating the consistency of the commitments) and randomly select a P_y owning an unwanted state to open the flag to check. This is not a problem as P_y cannot be traced to any traders from the previous steps or the subsequent ones due to the anonymity mechanism (Merkle Tree and ZK Functionalities). She is just an anonymous volunteer for this round. The functionality $\mathcal{F}_{\mathsf{pcheck}}$ can be similarly replaced.

The *Secure Detection* $\mathcal{F}_{\mathsf{dtc}}$ runs on common input (f) and interacts with a set of players $(P_1, \ldots, P_N)$ and receive (P_i, f_i, r_i) from each P_i. Upon receiving all inputs, let c_f be the number of pairs $(f_i = f)$, if $c_f > 0$ output $y = \sum r_i \bmod c_f$ to all players and $\perp$ otherwise. In the protocol, first each trader samples a random r_i and forwards P_i, f_i, r_i to $\mathcal{F}_{\mathsf{dtc}}$ with common input f to obtain y or $\perp$. In case of $\perp$, each trader P_i proves that her flag is different from the f (using $\mathcal{F}^{\mathsf{nec}}$). Otherwise trader P_y proves that the inventory flag is the same as the common input (by decommitting the flags). Any trader not able to prove this is considered as aborting.

XII. Evaluation

We evaluate our protocol in 3 steps. First we evaluate the performance of the cryptographic primitives that we use in our protocol, i.e. the zk-SNARK circuits and the MPC functionalities, both in the offline and online phase on a concrete implementation. Then, we use the obtained values to estimate the performance of each phase of our protocol. Finally, we evaluate the full protocol's performance by matching the above estimates with the public data available from the order books. Whilst network latencies are critical for high speed trading, we ignore them here since this issue is well understood by traders by either using known optimizations [47], [38], or by even buying dark fiber to cut delays between exchanges.[14]

All our experiments are run with an Amazon EC2 r4.4xlarge instance (Intel Xeon E5-2686 v4 @ 2.3Ghz, 16 cores, 122 GB RAM) so we exclude the communication cost[15]. The MPC protocol's off-line phase can also be pipelined with zk-SNARKs proof generation thus we exclude it as well.

a) zk-SNARK Circuits Performance: In Table IX, we report the performance metrics for the pre-processing steps: the key generation time (Key Gen), the size of the proving keys (PK), and the size of the verifying (VK) keys. We also report the time to generate a proof (Proving Time) and to verify it (Verify time), as well as the size of the proof during the actual trading execution described above. As shown, proving key size and proof generation time scale linearly with the number of commitments part of the relation.

b) Performance of the MPC functionalities: To gauge the effectiveness of our MPC components, we evaluate the performance for each functionality separately. Table X reports the size of the bytecode and the corresponding running times for 3, 5 and 10 traders. The memory requirement for the compilation of the

[11]Our prototype supports 32 bits signed integers (See footnote 3 on prices in §IV), a Merkle Tree of depth 10 and the net position range choices (i.e. $\mathcal{O}_{\mathsf{sell}}$ and $\mathcal{O}_{\mathsf{buy}}$) used in $\mathcal{F}^{\mathsf{rng}}$ are up to 10 .

[12]https://github.com/bristolcrypto/SPDZ-2.

[13]While libsnark is efficient and scalable, SPDZ hits the limit of 10 parties due to the complexity in implementing SHA256 with the library, i.e. right-shift is not natively supported for 32-bits word, and we had to implement it using left-shift and other bitwise operations.

[14]http://www.forbes.com/forbes/2010/0927/outfront-netscape-jim-barksdale- daniel-spivey-wall-street-speed-war.html

[15]However, each operation requires less than 20 commitments and 10 zk-SNARKs proof, thus per operation the data is less than 4KB. See Table IX.

TABLE IX: zk-SNARK Simple and Opt. Circuits Performance

	Pre-processing			On-Line Trading		
Circuit	KeyGen (ms)	| PK | (MB)	| VK | (KB)	Prove (ms)	| Proof | (B)	Verify (ms)
$\mathcal{F}^{\mathsf{rng}}$	8759	119	9	4752	287	31
$\mathcal{F}^{\mathsf{invm}}$	16778	210	2	8447	..	29
$\mathcal{F}^{\mathsf{token}}$	15925	189	..	7642	..	27
$\mathcal{F}^{\mathsf{flags}}$	12943	171	..	6115	..	..
$\mathcal{F}^{\mathsf{invt}}$	12954	171	..	6111	..	..
$\mathcal{F}^{\mathsf{uinv}}$	9650	116	..	4748	..	..
$\mathcal{F}^{\mathsf{match}}$	9644	116	..	4748	..	..
$\mathcal{F}^{\mathsf{net}}$	9638	115	..	4691	..	..
$\mathcal{F}^{\mathsf{mtm}}$	5456	57	..	2365	..	..
$\mathcal{F}^{\mathsf{ec}}$	3343	38	..	1429	..	..
$\mathcal{F}^{\mathsf{zero}^+}$	1639	19	..	739	..	26
$\mathcal{F}^{\mathsf{max}}$	1635	19	..	739	..	..
$\mathcal{F}^{\mathsf{oc}}$	1635	19	..	729	..	..
Optimized						
$\mathcal{F}^{\mathsf{token}}$	..	157	..	5691	..	..
$\mathcal{F}^{\mathsf{flags}}$	..	97	..	3908	..	..
$\mathcal{F}^{\mathsf{invt}}+\mathcal{F}^{\mathsf{uinv}}$	..	137	..	5193	..	..
$\mathcal{F}^{\mathsf{rng}}+\mathcal{F}^{\mathsf{net}}$	..	84	..	3509	..	..

TABLE X: MPC Performance

MPC Funct	Bytecode Size			On-Line Time		
#Traders	3	5	10	3	5	10
$\mathcal{F}_{\mathsf{compare}}$	425 MB	709 MB	1.4 GB	14s	24s	67s
$\mathcal{F}_{\mathsf{pcheck}}$	212 MB	354 MB	708 MB	7s	13s	36s

MPC functionalities using SHA-1 commitments crashed after 10 traders by exceeding 120 GB. We found that the dynamic memory requirement is typically 100x the final bytecode size. This was not reported before (e.g. [24]), and it is an important insight on the limit of the technology.

c) Overall Evaluation: In our experiment, we employ the futures trades in the first quarter 2017 for the Lean Hog futures market (See Table II) from the mentioned Thomson Reuters Tick History database[16]. For each day, we have five level limit orders (buy and sell, which we also chose for the $\mathcal{F}^{\mathsf{rng}}$) and transaction data at ticks level with millisecond timestamps. At a low end, we must be able to support a minimum of 10 traders and this is the limit we chose for illustrating our prototype. From the dataset we cannot determine the status of each trader (trader anonymity!), so we assume they have a large margin and never enter a broke state (i.e. we exclude Margin Settlement). We can combine the number of post, cancel and matched orders from market data (e.g. Table II and XI) to estimate the corresponding execution overhead throughout a day of trading. The final results are reported in Table XI. The actual timing of the protocol is still slow compared to the millisecond delay required by the CME but, if we compare the cost of our hardware (a $500 EC2 instance) to the "centralizedcompetitors" (CME's cost for IT infrastructure is *$30 millions* per year), we believe that a 10^3 delay (seconds vs milliseconds) with a 10^7 cheaper kit is acceptable.

Fig. 7 shows the overall record for the plain and optimized version as well as an estimation of a naive MPC implementation of the ideal functionality. For most of the entire quarter our distributed protocol can be optimized to have no overhead and executes all trades in the very same day.

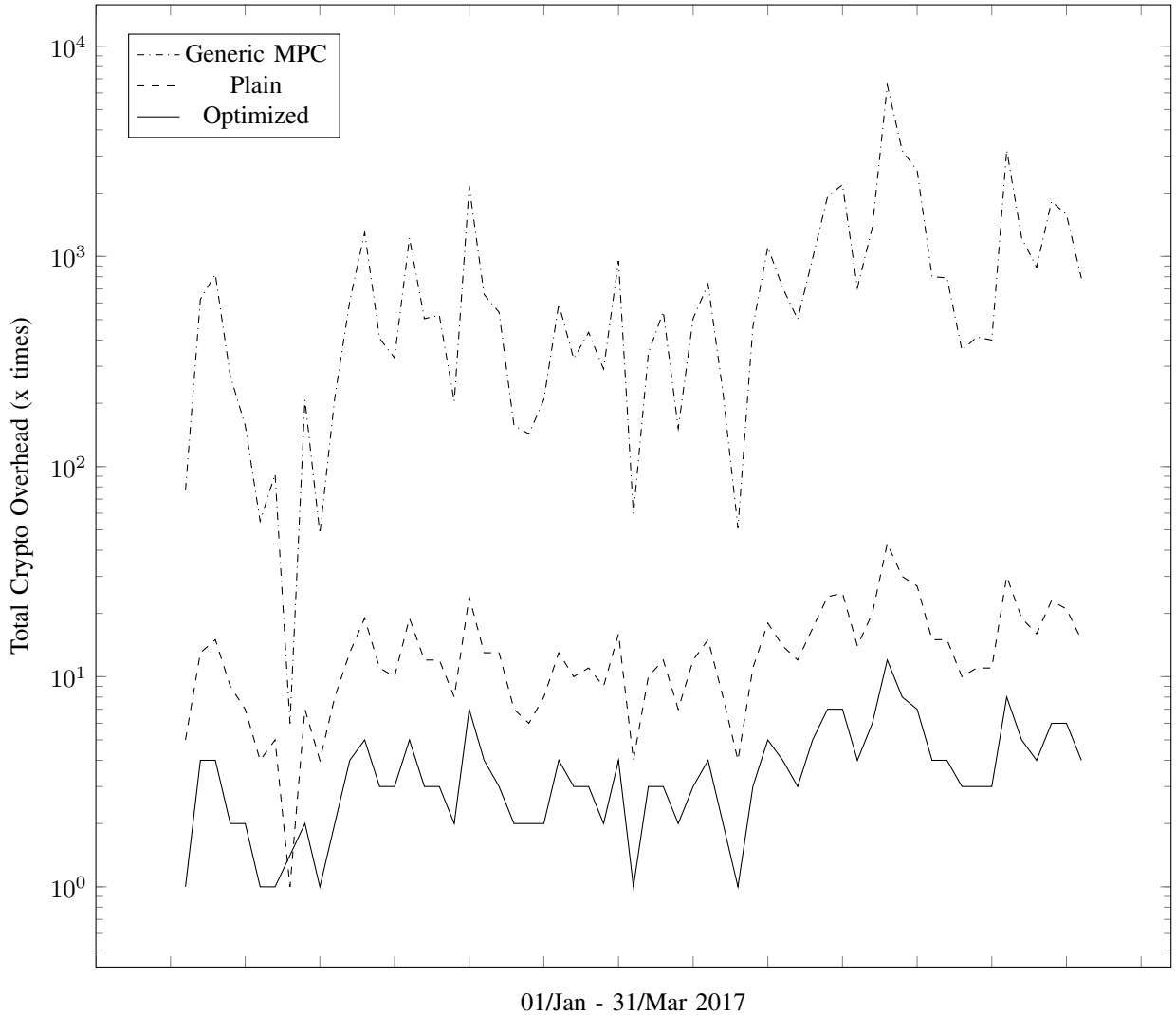
To estimate the cost of a naïve MPC implementation, we use as building block the simplest of our stateless MPC functionalities $\mathcal{F}_{\mathsf{dtc}}$ which is used to detect negative inputs and open one index. It costs only 0.2s for 10 traders. We then estimate the cost of a naïve MPC implementation of our stateful ideal

[16]https://tickhistory.thomsonreuters.com.

TABLE XI: Runtime of Individual Market Operations

Each individual operations can be done in few seconds for a market of 10 traders. With simple optimizations, we boost the performance and reduce the role of other traders in the computation which is a critical "ecological" constraint.

	Plain Prot. Runtime		Opt. Prot. Runtime	
Protocol	Trader	Others (%)	Trader	Others (%)
Initialize	11s	-	9s	-
Post Order	39s	148s (79%)	24s	27s (53%)
Cancel Order	40s	148s (79%)	25s	27s (52%)
Match Order	29s	148s (84%)	26s	27s (51%)
MarkToMarket	28s	-	25s	-

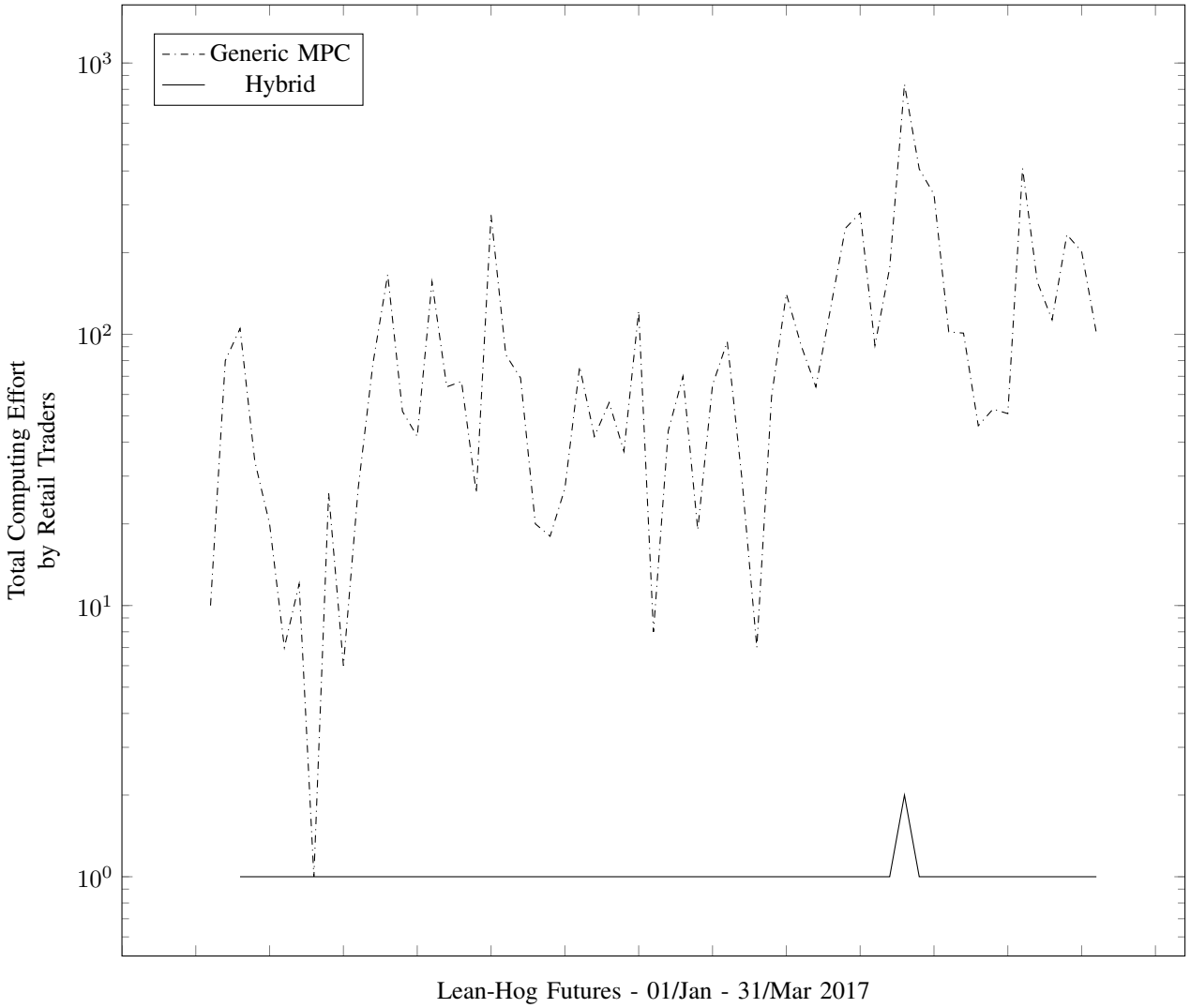


Performance in terms of execution overhead to the expected processing time (1 day). For the optimized version, only one day of trading exhibits overheads greater than 10x and only ten days greater than 5x. These overheads could be already offset by parallelizing the traders ZK-proofs (each trader has to do several of them) which would yield an improvement factor of 6x.

Fig. 7: Crypto Protocol Evaluation on Q1 of Lean-Hog

functionality by accumulating the steps in Fig. 2 and Fig. 5 under the favorable assumption (for MPC) that execution times accrue linearly with the number of steps.

Pure MPC impose a significant burden on retail traders (the overwhelming majority of the market) in particular during peak times when algorithmic traders frequently post and almost immediately cancel practically all orders (See [41]). Our hybrid approach shift the burden on computation on algorithmic



With MPC retail traders have to always participate whether they make an order or not (and they overwhelmingly don't [41]). They would be supplying to algorithmic traders some orders of magnitude of costly computing resources. With our approach the burden on retail traders is significantly smaller.

Fig. 8: Total Burden of Computation by Retail Traders

traders. As seen from Figure 8, retail traders would devote significant computational resources in a pure MPC implementation for allowing speculators to indeed speculate.

The optimized implementation can already break the barrier of ten traders and do the full 66 traders of the peak day. Parallelization can further reduce the runtime of the sub-protocol to just 8s (comparing to 24s of sequential proof generation). Furthermore, additional practical design decisions can further increase the protocol throughput, e.g. if traders can prove that their inventory is valid for a range of prices they would only need prove validity again when the price fluctuates out of that range, or by allowing multiple traders to post/cancel in one round, etc. We leave this for future investigation.

XIII. Related Work

Distributed Ledgers are ledgers maintained by a network of nodes. The most important property, e.g. for distributed payment networks, is consensus among the nodes, while still being fully decentralized. The most prominent example of a distributed payment network is Bitcoin [48], whose core components are the Proofs-of-Work and the Blockchain. The current bottleneck of Bitcoin is its low throughput in terms of transactions-per-second (TPS). (Roughly, 10 TPS compared to 2000 TPS achieved by, for instance,

Visa.) Several variants/extensions of Bitcoin appeared recently, including ZeroCoin [46], ZeroCash [7], and Ethereum [25].

Secure Multiparty Computation Seminal feasibility results in the theory of MPC established that any functionality is securely realizable via a distributed protocol in the computational (resp. information-theoretic) setting, assuming honest minority (resp. majority) [61], [29], [6], [19], [53]. The recent progress on efficient implementations of general-purpose MPC protocols (see, e.g., [24], [23], [3]) opened up the way to advanced applications, e.g. to privacy-preserving data mining [40]. See also [50] for an overview of applications of MPC. Privacy-preserving reputation systems only address half of our requirements (posting a public, yet anonymous, order, e.g. a rating of a service provider [62]), but not the other half (personally accruing the order's revenues).

SNARKs. The influential work of Micali on computationally sound proofs [45] gave the first zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK) for all of NP, relying on the random oracle heuristic [5]. Later work (starting with [22], [13]), showed that zk-SNARKs exist in the standard model, based on so-called knowledge-of-exponent assumptions; interestingly, these type of assumptions seem to be inherent for constructing zk-SNARKs [27]. An overview of these (and more) results can be found in [60]. In terms of implementation, the most efficient ones include [8], [10], [51].

XIV. Conclusions

This paper shows the first practical realization of distributed, secure, full financial intermediation without a trusted third party. Our chosen example has been one of the landmark institution of financial intermediation: a Futures Market Exchange. Besides the practical relevance of the application, such realization was interesting from a security perspectives as it requires to provide a rich security functionality with varied and potentially conflicting requirements. One needs to support a *public availability of information* about all actions performed by traders in the market (such as post or cancel orders) as well as *public verifiability of integrity of private information* from the traders. Further we need to provide *participants anonymity* and *public unlinkability* together with *global integrity guarantees and private linkability*.

Our hybrid protocol offers an efficient solution to the requirement of a *proportional burden of computation*. Traders infrequently making bids must only participate to the protocol to control market risk with a significant saving of the computational resources they would need to stake if general purpose MPC was used to implement the ideal functionality.

Our analysis of actual trading days using the Thomson-Reuters database, including a complete trading history at the level of milliseconds, have shown that the computation behind our protocol is within engineering reach: we simulated that on a *low frequency market* a secure protocol using a normal server (as opposed to traders' typical supercomputers) could still be executed within a day with an handful of exceptions. runs 10+ traders for 100+ transactions a day, so within our reach.

The first interesting avenue of future research is the choice of majority for the distributed consensus protocol. These solutions need further validation by financial economists: majority of traders or by weighted volumes? Should traders making offers far from the market price be considered as their offers might never be executed? Another direction is to analyze liveliness and robustness against collusion and price discrimination. For simplicity, our protocol by default goes to mark-to-market upon failure. Alternatives are possible, e.g. margin-call for additional funds, we leave them to future work.

In terms of practical implementation, we found the zk-SNARKs library to be pretty scalable, whilst the SPDZ library of our unoptimized implementation hit a hard limit at 10 traders due to the dynamic memory requirements (with the natural encoding of SHA256 in Python). This was surprising, as the final size of SPDZ bytecode was acceptable and consistent with the results from the literature [24]. We leave this issue for further investigations.

References

[1] F. Allen and A. M. Santomero, "The theory of financial intermediation," *J. of Banking & Fin.*, vol. 21, no. 11-12, pp. 1461 – 1485, 1997.

[2] M. Alsabah and I. Goldberg, "Performance and security improvements for tor: A survey," *ACM Comput. Surv.*, vol. 49, no. 2, pp. 32:1–32:36, 2016.

[3] D. W. Archer, D. Bogdanov, B. Pinkas, and P. Pullonen, "Maturity and performance of programmable secure computation," *Proc. of IEEE SSP*, vol. 14, no. 5, pp. 48–56, 2016.

[4] F. Baldimtsi and A. Lysyanskaya, "Anonymous credentials light," in *Proc. of ACM CCS*, 2013, pp. 1087–1098.

[5] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proc. of ACM CCS*, 1993, pp. 62–73.

[6] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract)," in *Proc. of ACM STOC*, 1988, pp. 1–10.

[7] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *Proc. of IEEE SSP*, 2014, pp. 459–474.

[8] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "Snarks for C: Verifying program executions succinctly and in zero knowledge," in *Proc. of CRYPTO*, 2013, pp. 90–108.

[9] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza, "Secure sampling of public parameters for succinct zero knowledge proofs," in *Proc. of IEEE SSP*, 2015, pp. 287–304.

[10] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *Proc. of USENIX Security*, 2014, pp. 781–796.

[11] R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias, "Semi-homomorphic encryption and multiparty computation," in *Proc. of Int. Conf. on the Theory and App. of Crypto. Tech.*, 2011, pp. 169–188.

[12] D. Bernhard and B. Warinschi, "Cryptographic voting - A gentle introduction," in *FOSAD*, 2013.

[13] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *Innov. in Theor. Comp. Sci.*, 2012, pp. 326–349.

[14] F. Boudot, "Efficient proofs that a committed number lies in an interval," in *Proc. of EUROCRYPT*, 2000, pp. 431–444.

[15] J. Camenisch, R. Chaabouni, and A. Shelat, "Efficient protocols for set membership and range proofs," in *Proc. of ASIACRYPT*, 2008, pp. 234–252.

[16] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proc. of IEEE FOCS*, 2001, pp. 136–145.

[17] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai, "Universally composable two-party and multi-party secure computation," in *Proc. of ACM STOC*, 2002, pp. 494–503.

[18] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM TOCS*, vol. 14, no. 5, pp. 398–461, 2002.

[19] D. Chaum, C. Crépeau, and I. Damgård, "Multiparty unconditionally secure protocols (extended abstract)," in *Proc. of ACM STOC*, 1988, pp. 11–19.

[20] J. Clark, J. Bonneau, E. W. Felten, J. A. Kroll, A. Miller, and A. Narayanan, "On decentralizing prediction markets and order books," in *Workshop on the Economics of Information Security, State College, Pennsylvania*, 2014.

[21] CoinDesk, "Understanding the dao attack," http://www.coindesk.com/understanding-dao-hack-journalists/, 2016.

[22] I. Damgård, S. Faust, and C. Hazay, "Secure two-party computation with low communication," in *Proc. of TCC*, 2012, pp. 54–74.

[23] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, "Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits," in *Proc. of ESORICS*, 2013, pp. 1–18.

[24] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Proc. of CRYPTO*, 2012, pp. 643–662.

[25] Ethereum, "A next-generation smart contract and decentralized application platform," 2015.

[26] Futures Industry Association, "Largest derivatives exchanges worldwide in 2015, by number of contracts traded (in millions)," 2016.

[27] C. Gentry and D. Wichs, "Separating succinct non-interactive arguments from all falsifiable assumptions," in *Proc. of ACM STOC*, 2011, pp. 99–108.

[28] O. Goldreich, *Foundations of Cryptography – Volume 2: Basic Applications*. Cambridge University Press, 2004.

[29] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or A completeness theorem for protocols with honest majority," in *Proc. of ACM STOC*, 1987, pp. 218–229.

[30] L. Harris, *Trading and exchanges: Market microstructure for practitioners*. Oxford University Press, 2003.

[31] J. Hull, S. Treepongkaruna, D. Colwell, R. Heaney, and D. Pitt, *Fundamentals of futures and options markets*. Pearson H. Edu, 2013.

[32] Y. Ishai, J. Katz, E. Kushilevitz, Y. Lindell, and E. Petrank, "On achieving the "best of both worlds" in secure multiparty computation," *SIAM J. on Comp.*, vol. 40, no. 1, pp. 122–141, 2011.

[33] A. Kiayias, T. Zacharias, and B. Zhang, "An efficient e2e verifiable e-voting system without setup assumptions," *IEEE S&P Mag.*, 2016.

[34] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. of IEEE SSP*, 2016, pp. 839–858.

[35] A. E. Kosba, Z. Zhao, A. Miller, Y. Qian, T. H. Chan, C. Papamanthou, R. Pass, A. Shelat, and E. Shi, "How to use snarks in universally composable protocols," *IACR Crypto. ePrint Ar.*, vol. 2015, 2015.

[36] R. Kumaresan, T. Moran, and I. Bentov, "How to use bitcoin to play decentralized poker," in *Proc. of ACM CCS*, 2015, pp. 195–206.

[37] R. Kumaresan, V. Vaikuntanathan, and P. N. Vasudevan, "Improvements to secure computation with penalties," in *Proc. of ACM CCS*, 2016, pp. 406–417.

[38] J. Labuszewski, J. Nyhoff, J. Boudreault *et al.*, "Disseminating floor quotes from open outcry markets," Oct. 23 2013, uS Patent App. 14/061,286.

[39] Y. Lindell, "How to simulate it – A tutorial on the simulation proof technique," *IACR Crypto. ePrint Ar.*, vol. 2016, p. 46, 2016.

[40] Y. Lindell and B. Pinkas, "Secure multiparty computation for privacy-preserving data mining," *IACR Crypto. ePrint Ar.*, vol. 2008, 2008.

[41] K. Malinova, A. Park, and R. Riordan, "Do retail traders suffer from high frequency traders," *Available at SSRN*, vol. 2183806, 2013.

[42] J. W. Markham, "Manipulation of commodity futures prices-the unprosecutable crime," *Yale J. on Reg.*, vol. 8, p. 281, 1991.

[43] F. Massacci, C. N. Ngo, J. Nie, D. Venturi, and J. Williams, "The seconomics (security-economics) vulnerabilities of decentralized autonomous organizations," in *Proc. of SPW*, 2017, pp. 171–179.

[44] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Proc. of CRYPTO*, 1987, pp. 369–378.

[45] S. Micali, "Computationally sound proofs," *SIAM J. on Comp.*, vol. 30, no. 4, pp. 1253–1298, 2000.

[46] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from bitcoin," in *Proc. of IEEE SSP*, 2013, pp. 397–411.

[47] M. Morano, I. Wall, S. Gaer, and K. Neumann, "Distributed trading bus architecture," Feb. 15 2011, uS Patent 7,890,412.

[48] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[49] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra, "A new approach to practical active-secure two-party computation," in *Proc. of CRYPTO*, 2012, pp. 681–700.

[50] C. Orlandi, "Is multiparty computation any good in practice?" in *IEEE ICASSP*, 2011, pp. 5848–5851.

[51] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: nearly practical verifiable computation," *ACM Comm.*, vol. 59, no. 2, pp. 103–112, 2016.

[52] C. Pirrong, "The economics of clearing in derivatives markets: Netting, asymmetric information, and the sharing of default risks through a central counterparty," *Asym. Info., & the Sharing of Default Risks Through a Central Counterparty*, 2009.

[53] T. Rabin and M. Ben-Or, "Verifiable secret sharing and multiparty protocols with honest majority (extended abstract)," in *Proc. of ACM STOC*, 1989, pp. 73–85.

[54] K. Sako, "An auction protocol which hides bids of losers," in *Pub. -Key Crypto.*, 2000, pp. 422–432.

[55] T. Sander and A. Ta-Shma, "Auditable, anonymous electronic cash," in *Proc. of IEEE ICC*, 1999, pp. 555–572.

[56] D. F. Spulber, "Market microstructure and intermediation," *J. of Econ. Persp.*, vol. 10, no. 3, pp. 135–152, 1996.

[57] US CFTC, "Mission & responsibilities," 2016.

[58] U.S. Securities and Exchange Commission, "Concept release on equity market structure," *Release No. 34-61458; File No. S7-02*, vol. 10, 2010.

[59] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. bft replication," in *iNetSec*, 2016, pp. 112–125.

[60] M. Walfish and A. J. Blumberg, "Verifying computations without reexecuting them," *ACM Comm.*, vol. 58, no. 2, pp. 74–84, 2015.

[61] A. C. Yao, "Protocols for secure computations (extended abstract)," in *Proc. of IEEE FOCS*, 1982, pp. 160–164.

[62] E. Zhai, D. I. Wolinsky, R. Chen, E. Syta, C. Teng, and B. Ford, "Anonrep: Towards tracking-resistant anonymous reputation." 2016, pp. 583–596.

APPENDIX

A. Supporting Material: Protocol Construction

Fig. 9 provides the formal description for Π_{put} and Π_{get} whereas Fig. 10 contains a formal description of our protocol.

B. Supporting Material: Security Analysis

Theorem 1 in §IX states the security of our protocol Π_{DFM} from §VIII. Below, we formalize security in the stand-alone setting with a malicious adversary. We refer to [28], [39] for a more extensive discussion of the standard formal definitions.

Proof: It is clear that Π_{DFM} computes $\mathcal{F}_{\mathsf{CFM}}$. We proceed to prove the security of Π_{DFM}. Let $\mathcal{A}$ be a non-uniform deterministic PPT adversary. The simulator $\mathcal{S}$ is given access to the ideal functionality $\mathcal{F}_{\mathsf{CFM}}$, and can also read the stored/updated values of the corrupted traders (that $\mathcal{A}$ controls) from $\mathcal{F}_{\mathsf{CFM}}$; recall that, since we prove only static security, the set of corrupted traders I is fixed before the protocol execution starts.

Sub-protocol Π_{put}**:** The protocol is run by $(P_1, \ldots, P_N)$ in order to let P_i commit to a new inventory by adding the commitment of the inventory token to a Merkle Tree $\mathcal{T}$ (resulting in a new root ρ^*).

1) P_i picks $r_i \leftarrow\!\!\$\ \{0,1\}^*$, computes $\tau_i = \mathsf{Com}(m_i||v_i||\widehat{m_i}||\widehat{v_i}||c_i||f_{\mathsf{bad},i}||f_{\mathsf{del},i}||f_{\mathsf{out},i}; r_i)$, and broadcasts $[\![\tau_i]\!] \leftarrow\!\!\$\ \mathsf{Com}(\tau_i)$.
2) P_i sends $(P_i, x_i^{\mathsf{token}}, w_i^{\mathsf{token}})$ and each other $P_{j\neq i}$ sends $(P_j, x_j^{\mathsf{token}})$ to $\mathcal{F}^{\mathsf{token}}$.
3) P_i runs $\rho^* = \mathsf{Add}(\mathcal{T}, [\![\tau_i]\!])$ and broadcasts ρ^*. The other traders accept ρ^* iff the token has been correctly added to the tree.

Sub-protocol Π_{get}**:** The protocol is used when a trader either to post a new order $(\delta_c = 1, l, v)$, or cancel a pending order $(\delta_c = -1, l, v)$, or simply retrieve an **unspent** inventory that was committed at round $t' \leq t$ into a Merkle Tree $\mathcal{T}$; any **spent** token is rejected.

1) P_i recovers $path = \mathsf{path}(\mathcal{T}, [\![\tau_i']\!])$ and broadcasts τ_i'.
2) P_i sends $(P_i, x_i^{\mathsf{inv}}, w_i^{\mathsf{inv}})$ and each other $P_{j\neq i}$ sends $(P_j, x_j^{\mathsf{inv}})$ to $\mathcal{F}^{\mathsf{inv}}$. The token τ_i' is now marked as **spent** by all traders.
3) If $(\delta_c, l, v) \neq (0,0,0)$, additionally P_i sends $(P_i, x_i^{\mathsf{uinv}}, w_i^{\mathsf{uinv}})$ and each other $P_{j\neq i}$ sends $(P_j, x_j^{\mathsf{uinv}})$ to $\mathcal{F}^{\mathsf{uinv}}$.

Fig. 9: Sub-protocols Π_{put} and Π_{get}

a) Sub-routines.: To simplify the simulator's description, we introduce sub-routines $\mathcal{S}_{\mathsf{put}}$, $\mathcal{S}_{\mathsf{get}}$ as well as $\mathcal{S}_{\mathsf{valid}}$, $\mathcal{S}_{\mathsf{backup}}$, $\mathcal{S}_{\mathsf{net}}$, $\mathcal{S}_{\mathsf{match}}$ that will call $\mathcal{S}_{\mathsf{put}}$, $\mathcal{S}_{\mathsf{get}}$ when it is related to commit and retrieve of inventories. The sub-routines will be later invoked by $\mathcal{S}$; while reading them, think of the simulator's behaviour as a simulation strategy for the corresponding protocols Π_{put}, Π_{get}, Π_{valid}, Π_{backup}, Π_{net}, and Π_{match}. Each sub-routine invokes $\mathcal{A}$ and receives messages from it. The sub-routines use $\mathcal{S}_{\mathsf{get}}$ and $\mathcal{S}_{\mathsf{put}}$ above. Since we are working in the hybrid model, whenever $\mathcal{A}$ interacts with an ideal functionality the simulator receives $\mathcal{A}$'s inputs to the functionality in the clear, and thus it can perfectly emulate the output of the hybrid functionality.

$\mathcal{S}_{\mathsf{put}}$**:** When a trader P_i commits to an inventory, it acts as a prover while the other traders act as verifiers. If P_i is corrupted, $\mathcal{S}$ needs to simulate the views of both the prover P_i and the corrupted verifiers P_j, by receiving the inputs from P_i and forwarding them to each corrupted verifier $P_{j\neq i}$. Otherwise, $\mathcal{S}$ only needs to simulate the view of the corrupted verifiers P_j, by forwarding $[\![\mathbf{0}]\!]$ and $\rho = \mathsf{Add}(\mathcal{T}, [\![\mathbf{0}]\!])$ to each corrupted verifier P_j. In both case $\mathcal{S}$ simulates the output of $\mathcal{F}^{\mathsf{token}}$ for each corrupted players, abort the simulation if any check fails.

$\mathcal{S}_{\mathsf{get}}$**:** Similar to $\mathcal{S}_{\mathsf{put}}$ but using $\mathcal{F}^{\mathsf{inv}}$ and $\mathcal{F}^{\mathsf{uinv}}$.

$\mathcal{S}_{\mathsf{valid}}$**:** Similar to $\mathcal{S}_{\mathsf{put}}$ but using $\mathcal{F}^{\mathsf{oc}}$, $\mathcal{F}^{\mathsf{rng}}$, $\mathcal{F}^{\mathsf{net}}$, and $\mathcal{F}^{\mathsf{zero}^+}$.

$\mathcal{S}_{\mathsf{backup}}$**:** Similar to $\mathcal{S}_{\mathsf{put}}$ but with $\mathcal{F}^{\mathsf{rng}}$, $\mathcal{F}^{\mathsf{net}}$ and $\mathcal{F}_{\mathsf{pcheck}}$.

$\mathcal{S}_{\mathsf{net}}$**:** When a trader P_i needs to be checked for a non-negative instant net position, it acts as a prover while the other traders act as verifiers. The steps are also similar to $\mathcal{S}_{\mathsf{put}}$ but with $\mathcal{F}^{\mathsf{rng}}$, $\mathcal{F}^{\mathsf{net}}$, $\mathcal{F}^{\mathsf{flags}}$ and $\mathcal{F}_{\mathsf{compare}}$.

$\mathcal{S}_{\mathsf{match}}$**:** When a trader P_i posts an order, it acts as a prover together with some other trader $P_{i'}$ with a matching order, while the other traders act as verifiers. We distinguish four cases for the honesty of P_i and $P_{i'}$. The steps are also similar to $\mathcal{S}_{\mathsf{put}}$ but with $\mathcal{F}^{\mathsf{oc}}$ and $\mathcal{F}^{\mathsf{match}}$.

b) Simulator description.: We are now ready to describe the simulator. In each round $t \leq T$, the simulator $\mathcal{S}$ runs as follows depending on the current phase the protocol.

Initialization: Let P_i be the trader committing to a **good** inventory. If P_i **is corrupted**:

1) Receive the commitments of inventory values and $[\![\tau_i]\!]$ from P_i; obtain the inputs that $\mathcal{A}$ sends to $\mathcal{F}^{\mathsf{zero}^+}$ and $\mathcal{F}^{\mathsf{ec}}$, and simulate the output of such ideal functionalities for each corrupted trader in I.
2) Forward $(\mathsf{init}, P_i, m_i)$ to $\mathcal{F}_{\mathsf{CFM}}$; if the ideal functionality returns 0 simulate an abort of the protocol.
3) Receive the decommitments corresponding to $[\![v_i]\!]$, $[\![\widehat{v_i}]\!]$, $[\![c_i]\!]$, $[\![f_{\mathsf{bad},i}]\!]$, $[\![f_{\mathsf{del},i}]\!]$, $[\![f_{\mathsf{out},i}]\!]$, and simulate an abort of the protocol in case such values are not valid openings.

Initialize Phase: This phase runs in the first round where each trader P_i commits to a **good** inventory with cash m_i.

1) Trader P_i commits to and broadcasts $[\![m_i]\!]$, $[\![v_i]\!]$, $[\![\widehat{m_i}]\!]$, $[\![\widehat{v_i}]\!]$, $[\![c_i]\!]$, $[\![f_{\mathsf{bad},i}]\!]$, $[\![f_{\mathsf{del},i}]\!]$, $[\![f_{\mathsf{out},i}]\!]$ where ($\widehat{m_i} = m_i$ and $v_i = \widehat{v_i} = c_i = f_{\mathsf{bad},i} = f_{\mathsf{del},i} = f_{\mathsf{out},i} = 0$).
2) Trader P_i broadcasts a token $[\![\tau_i]\!]$, where $\tau_i = \mathsf{Com}(m_i||v_i||\widehat{m_i}||\widehat{v_i}||c_i||f_{\mathsf{bad},i}||f_{\mathsf{del},i}||f_{\mathsf{out},i}; r_i)$ with $r_i \leftarrow\!\!\$\ \{0,1\}^*$.
3) P_i sends $(P_i, [\![m_i]\!], m_i)$ and each other $P_{j\neq i}$ sends $(P_j, [\![m_i]\!])$ to $\mathcal{F}^{\mathsf{zero}^+}$.
4) P_i sends $(P_i, ([\![\widehat{m_i}]\!], [\![m_i]\!]), (\widehat{m_i}, m_i))$ and each other $P_{j\neq i}$ sends $(P_j, ([\![\widehat{m_i}]\!], [\![m_i]\!]))$ to $\mathcal{F}^{\mathsf{ec}}$.
5) P_i decommits $[\![v_i]\!]$, $[\![\widehat{v_i}]\!]$, $[\![c_i]\!]$, $[\![f_{\mathsf{bad},i}]\!]$, $[\![f_{\mathsf{del},i}]\!]$, $[\![f_{\mathsf{out},i}]\!]$ to show the values are 0.
6) All traders run Π_{put}.
7) Set $\mathcal{T}_{\mathcal{U}} := \mathcal{T}$.

Post/Cancel Order Phase: This phase is run in order to post an order $o = (t, l, i, v)$ at round t (where $l \geq l_{\mathsf{buy}}$ for $v < 0$, and $l \leq l_{\mathsf{sell}}$ for $v > 0$) or to cancel an order $o = (t', l, i, v)$.

1) P_i broadcasts the values $(1, l, v)$ and a commitment $[\![i]\!]$ (resp. P_i broadcasts the values t' and $(-1, l, v)$ in **Cancel Order**).
2) All traders run Π_{valid} then Π_{net}.
3) In case of **Post Order**, starting from $t' = 1$, for each matching entry $(t', l, [\![i']\!], v') \in \mathcal{O}$ such that $v \cdot v' < 0$, until $v = 0$ or $t' = t$, all traders run Π_{match} then Π_{net}.

Margin Settlement Phase: This phase is run whenever at least one inventory was added to the Merkle Tree $\mathcal{T}$ during the sub-protocol Π_{net}. The protocol goes back to the invoking phase afterward.

1) For every **unspent** inventory of P_i in $\mathcal{T}$, each $o' = (t', l, [\![i]\!], v)$ has to be canceled:
   a) P_i broadcasts the values t' and $(-1, l, v)$.
   b) P_i sends $(P_i, [\![i]\!], r_i)$ to $\mathcal{F}^{\mathsf{oc}}$, while $P_{j\neq i}$ sends $(P_j, [\![i]\!])$.
   c) All traders run Π_{valid}.
   d) P_i forwards $f^*_{\mathsf{del},i}$ and $f_{\mathsf{bad},i}$ to $\mathcal{F}_{\mathsf{compare}}$; if $\mathcal{F}_{\mathsf{compare}}$ returns 1, proceed to next step, else go back to step (a).
2) The traders run Π_{net}, and return to step 1 if there is any new broke inventory.
3) For every broke **unspent** inventory of trader P_i in $\mathcal{T}$, offset the volume holding v_i until $v_i = 0$. In particular, P_i locally looks up the order book from $t' = 1$ to $t' = t$ for an order $o' = (t', l, [\![i']\!], v')$ (where $l = l_{\mathsf{sell}}$ for $v_i < 0$, and $l = l_{\mathsf{buy}}$ if $v_i > 0$), and then the following steps are performed for each o':
   a) The traders run Π_{match}.
   b) P_i forwards $f^*_{\mathsf{out},i}$ and $f_{\mathsf{bad},i}$ to $\mathcal{F}_{\mathsf{compare}}$. If $\mathcal{F}_{\mathsf{compare}}$ returns 1, proceed to next step, and else go back to step (a).
4) The traders run Π_{net}, and return to step 1 if there is any new broke inventory.
5) All traders run Π_{backup} to check and fork a backup tree, otherwise, the traders proceed to **Mark to Market** using the tree $\mathcal{T}_{\mathcal{U}}$.

Mark to Market Phase: This phase is either called during **Margin Settlement**, or in the last round, where every trader P_i retrieves and commits to a **good** inventory with new marked-to-market values.

1) All traders run Π_{get}.
2) P_i computes $m^*_i := \widehat{m_i}^* := m_i + \bar{p} \cdot v_i$ and set $v^*_i := \widehat{v_i}^* := c^*_i := f^*_{\mathsf{bad},i} := f^*_{\mathsf{del},i} := f^*_{\mathsf{out},i} := 0$.
3) P_i broadcasts $[\![m^*_i]\!]$, $[\![v^*_i]\!]$, $[\![\widehat{m_i}^*]\!]$, $[\![\widehat{v_i}^*]\!]$, $[\![c^*_i]\!]$ and $[\![f^*_{\mathsf{bad},i}]\!]$, $[\![f^*_{\mathsf{del},i}]\!]$, $[\![f^*_{\mathsf{out},i}]\!]$.
4) P_i decommits $[\![v^*_i]\!]$, $[\![\widehat{v_i}^*]\!]$, $[\![c^*_i]\!]$ and $[\![f^*_{\mathsf{bad},i}]\!]$, $[\![f^*_{\mathsf{del},i}]\!]$, $[\![f^*_{\mathsf{out},i}]\!]$. to show the values are 0.
5) P_i sends $(P_i, x^{\mathsf{mtm}}_i, w^{\mathsf{mtm}}_i)$ to $\mathcal{F}^{\mathsf{mtm}}$, while $P_{j\neq i}$ sends $(P_j, x^{\mathsf{mtm}}_j)$.
6) P_i sends $(P_i, ([\![\widehat{m_i}^*]\!], [\![m^*_i]\!]), (\widehat{m_i}^*, m^*_i))$ and each other $P_{j\neq i}$ sends $(P_j, ([\![\widehat{m_i}^*]\!], [\![m^*_i]\!]))$ to $\mathcal{F}^{\mathsf{ec}}$.
7) All traders run Π_{put}.

Fig. 10: The protocol Π_{DFM}

4) Run $\mathcal{S}_{\mathsf{put}}$ (for the case of corrupted P_i).

If P_i **is honest** we proceed as follows.

1) Forward commitments to zero for each of the values broadcast by P_i in the first step of the initialize phase; obtain the inputs that $\mathcal{A}$ sends to $\mathcal{F}^{\mathsf{zero}^+}$ and $\mathcal{F}^{\mathsf{ec}}$, and simulate the output of such ideal functionalities for each corrupted trader in I.
2) Open the commitments to zero corresponding to $[\![v_i]\!]$, $[\![\widehat{v_i}]\!]$, $[\![c_i]\!]$, $[\![f_{\mathsf{bad},i}]\!]$, $[\![f_{\mathsf{del},i}]\!]$, $[\![f_{\mathsf{out},i}]\!]$.
3) Run $\mathcal{S}_{\mathsf{put}}$ (for the case of honest P_i).

Post/Cancel Order: Let P_i be the trader posting an order or canceling a previous order. We distinguish two cases for Cancel Order and 4 cases for Post Order. If P_i **is corrupted:**:

1) In case of Post Order, receive the values $(1, l, v)$ and $[\![i]\!]$ from P_i; forward $(\mathsf{post\_order}, P_i, t, l, v)$ to $\mathcal{F}_{\mathsf{CFM}}$. Otherwise, receive the values t' and $(-1, l, v)$ from P_i; forward $(\mathsf{cancel\_order}, P_i, t')$ to $\mathcal{F}_{\mathsf{CFM}}$.
2) Run $\mathcal{S}_{\mathsf{valid}}$ and $\mathcal{S}_{\mathsf{net}}$ (for the case of corrupted P_i).
3) In case of Post Order, for each command $(\mathsf{match}, t', l, v')$ received from $\mathcal{F}_{\mathsf{CFM}}$ run $\mathcal{S}_{\mathsf{match}}$ and then $\mathcal{S}_{\mathsf{net}}$ (for the case of corrupted P_i).

If P_i **is honest:** we proceed as follows.

1) In case of Post Order, receive $(\mathsf{post\_order}, t, l, v)$ from $\mathcal{F}_{\mathsf{CFM}}$. Otherwise receive $(\mathsf{cancel\_order}, t')$ from $\mathcal{F}_{\mathsf{CFM}}$.
2) Run $\mathcal{S}_{\mathsf{valid}}$ and $\mathcal{S}_{\mathsf{net}}$ (for the case of honest P_i).
3) In case of Post Order, for each command $(\mathsf{match}, t', l, v')$ received from $\mathcal{F}_{\mathsf{CFM}}$ run $\mathcal{S}_{\mathsf{match}}$ and then $\mathcal{S}_{\mathsf{net}}$ (for the case of honest P_i).

If P_i **is honest and** $P_{i'}$ **is corrupted (or viceversa):** proceed as above, depending on who is honest/corrupted.

Margin Settlement: Note that during this phase $\mathcal{S}$ always learns the list of pending orders that are canceled, as a public output of $\mathcal{F}_{\mathsf{CFM}}$. If P_i **is corrupted** we proceed as follows.

1) For each order to be canceled in an unspent inventory:
   a) Receive the values t' and $(-1, l, v)$ from P_i; obtain the inputs that $\mathcal{A}$ sends to $\mathcal{F}^{\mathsf{oc}}$, and simulate the output of such ideal functionality for each corrupted trader in I.
   b) Run $\mathcal{S}_{\mathsf{valid}}$ and $\mathcal{S}_{\mathsf{net}}$ (for the case of corrupted P_i).
   c) Obtain the inputs that $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{compare}}$, and simulate the output of such ideal functionality for each corrupted trader in I.
2) For each broke unspent inventory:
   a) Run $\mathcal{S}_{\mathsf{match}}$ and $\mathcal{S}_{\mathsf{net}}$ (for the case of corrupted P_i).
   b) Obtain the inputs that $\mathcal{A}$ sends to $\mathcal{F}_{\mathsf{compare}}$, and simulate the output of such ideal functionality for each corrupted trader in I.
3) Run $\mathcal{S}_{\mathsf{backup}}$ (for the case of corrupted P_i).

If P_i **is honest:** Same as above, except that the values (l, v') for each order to be canceled are obtained from $\mathcal{F}_{\mathsf{CFM}}$.

Mark To Market: Let P_i be the trader committing to a good inventory with marked-to-market values. If P_i **is corrupted:**

1) Run $\mathcal{S}_{\mathsf{get}}$ (for the case of corrupted P_i).
2) Receive the commitments and obtain the inputs that $\mathcal{A}$ sends to $\mathcal{F}^{\mathsf{mtm}}$, and simulate the output of such ideal functionality for each corrupted trader in I.
3) Run $\mathcal{S}_{\mathsf{put}}$ (for the case of corrupted P_i).

If P_i **is honest:** Proceed as follows.

1) Run $\mathcal{S}_{\mathsf{get}}$ (for the case of honest P_i).
2) Forward commitments to zero for each of the values broadcast by P_i in the second step of the **Mark to Market** phase; obtain the inputs that $\mathcal{A}$ sends to $\mathcal{F}^{\mathsf{mtm}}$, and simulate the output of such ideal functionality for each corrupted trader in I.
3) Run $\mathcal{S}_{\mathsf{put}}$ (for the case of honest P_i).

c) Indistinguishability of the simulation.: We need to show that for all PPT adversaries $\mathcal{A}$, all $I \subseteq [N]$, and every auxiliary input $z \in \{0,1\}^*$, the following holds:

$$\mathtt{REAL}_{\Pi_{\mathsf{DFM}}, \mathcal{A}(z), I} \approx_c \mathtt{IDEAL}_{\mathcal{F}_{\mathsf{CFM}}, \mathcal{S}(z), I}.$$

We start by considering a hybrid experiment $\mathtt{HYBRID}^1_{\mathcal{A}(z), \mathcal{S}_1, I}$ with a simulator $\mathcal{S}_1$ that runs exactly the same as $\mathcal{S}$, except that $\mathcal{S}_1$ also plays the role of the ideal functionality $\mathcal{F}_{\mathsf{CFM}}$ on its own. This means that $\mathcal{S}_1$ directly receives the inputs of other honest traders that are not under control of $\mathcal{A}$. Clearly, for all adversaries

$\mathcal{A}$, all subsets I, and every auxiliary in put $z \in \{0,1\}^*$, we have that $\texttt{HYBRID}^1_{\mathcal{A}(z),\mathcal{S}_1,I} \equiv \texttt{IDEAL}_{\mathcal{F}_{\mathsf{CFM}},\mathcal{S}(z),I}$, as there is no difference in generating the view of $\mathcal{A}$ in the two experiments.

Next, we consider another hybrid experiment $\texttt{HYBRID}^2_{\mathcal{A}(z),\mathcal{S}_2,I}$ with a simulator $\mathcal{S}_2$ that runs exactly the same as $\mathcal{S}_1$, except that whenever $\mathcal{S}_1$ committs to zero values when dealing with dishonest verifiers, $\mathcal{S}_2$ commits to the real values received from the honest provers. The lemma below shows that the two experiments are statistically close.

Lemma 1. *For all (unbounded) adversaries $\mathcal{A}$, all $I \subseteq [N]$, and every $z \in \{0,1\}^*$:* $\texttt{HYBRID}^1_{\mathcal{A}(z),\mathcal{S}_1,I} \approx_s \texttt{HYBRID}^2_{\mathcal{A}(z),\mathcal{S}_2,I}$

Proof: The proof is down to the statistical hiding property of the non-interactive commitment Com.

We consider a variant of the statistical hiding property where a distinguisher $\mathcal{D}$ is given access to a left-or-right oracle $O_{\mathsf{lr}}(b,\cdot)$, parametrized by a bit $b \in \{0,1\}$, that upon input $v \in \{0,1\}^*$ returns $[\![v]\!]$ (if $b = 0$) or $[\![\mathbf{0}]\!]$ (if $b = 1$), where $|\mathbf{0}| = |v|$; hence, we have Com is statistically hiding if for all computationally unbounded $\mathcal{D}$,

$$\left|\Pr\left[\mathcal{D}^{O_{\mathsf{lr}}(0,\cdot)}(1^\lambda) = 1\right] - \Pr\left[\mathcal{D}^{O_{\mathsf{lr}}(1,\cdot)}(1^\lambda) = 1\right]\right| \leq \nu(\lambda),$$

for a negligible function $\nu : \mathbb{N} \to [0,1]$. By a standard hybrid argument, as long as $\mathcal{D}$ makes a polynomial (in λ) number of oracle queries, the above flavor of statistical hiding is equivalent to that of Com.

Assume there exists a distinguisher $\mathcal{D}'$ and a polynomial $p(\lambda)$, such that, for some $I \subseteq [N]$ and $z \in \{0,1\}^*$, and for infinitely many values of $\lambda \in \mathbb{N}$, we have that

$$\Bigg|\Pr\left[\mathcal{D}'(\texttt{HYBRID}^1_{\mathcal{A}(z),\mathcal{S}_1,I}) = 1\right] - \Pr\left[\mathcal{D}'(\texttt{HYBRID}^2_{\mathcal{A}(z),\mathcal{S}_2,I}) = 1\right]\Bigg| \geq 1/p(\lambda).$$

We can construct a distinguisher $\mathcal{D}$ breaking the statistical hiding property of Com as follows. $\mathcal{D}$ runs $\mathcal{A}$ and simulates an execution of protocol Π_{DFM} exactly as $\mathcal{S}_1$ does, except that whenever $\mathcal{S}_1$ forwards a commitment to zero, $\mathcal{D}$ asks a query to the left-or-right oracle and sends the output of the oracle to $\mathcal{A}$; the value v for each oracle query is equal to the value $\mathcal{S}_2$ would commit to (instead of committing to zero).

In case $\mathcal{D}$ receives always commitments to zero, the view of $\mathcal{A}$ when run by $\mathcal{D}$ is identical to the view in the first hybrid experiment; on the other hand, in case $\mathcal{D}$ receives always commitments to the values queries to the left-or-right oracle, the view of $\mathcal{A}$ when run by $\mathcal{D}$ is identical to the view in the second hybrid experiment.

Thus, $\mathcal{D}$ retains the same advantage of $\mathcal{D}'$. This concludes the proof. ■

The lemma below says that the view of the adversary in the last hybrid experiment is computationally indistinguishable from the view in the real experiment.

Lemma 2. *For all PPT adversaries $\mathcal{A}$, all $I \subseteq [N]$, and every $z \in \{0,1\}^*$, it is* $\texttt{HYBRID}^2_{\mathcal{A}(z),\mathcal{S}_2,I} \approx_c \texttt{REAL}_{\Pi_{\mathsf{DFM}},\mathcal{A}(z),I}$.

Proof: Fix $I \subseteq [N]$, and $z \in \{0,1\}^*$. Consider the following events, defined over the probability space of the last hybrid experiment.

Event $\texttt{Bad}_{\texttt{inv}}$: The event becomes true whenever $\mathcal{A}$ can modify the inventory of a corrupted trader P_i, by finding two distinct valid openings for a token τ_i. The computational binding property of Com implies that $\Pr[\texttt{Bad}_{\texttt{inv}}]$ is negligible.

Event $\texttt{Bad}_{\texttt{spend}}$: The event becomes true whenever $\mathcal{A}$ can double spend the inventory of a corrupted trader P_i, by finding two distinct valid openings for $[\![\tau_i]\!]$. The computational binding property of Com implies that $\Pr[\texttt{Bad}_{\texttt{inv}}]$ is negligible.

Event $\texttt{Bad}_{\texttt{forge}}$: The event becomes true whenever $\mathcal{A}$ forges an inventory of a trader P_i, by finding two distinct valid authentication paths for a leaf $[\![\tau_i]\!]$ of the Merkle Tree. The computational binding property

of the Merkle Tree, which follows from the collision resistance of the underlying hash function, implies that $\Pr[\mathtt{Bad}_{\mathtt{forge}}]$ is negligible.

Event $\mathtt{Bad}_{\mathtt{swap}}$**:** The event becomes true whenever $\mathcal{A}$ claim a pending order of an honest trader $P_{i'}$, by finding two valid openings for the commitment $[\![i']\!]$. The computational binding property of Com implies that $\Pr[\mathtt{Bad}_{\mathtt{swap}}]$ is negligible.

Define $\mathtt{Bad} := \mathtt{Bad}_{\mathtt{inv}} \vee \mathtt{Bad}_{\mathtt{spend}} \vee \mathtt{Bad}_{\mathtt{forge}} \vee \mathtt{Bad}_{\mathtt{swap}}$. It is not hard to see that conditioning on Bad not happening, the view of $\mathcal{A}$ is identical in the two experiments. This is because the only difference between the last hybrid and the real experiment is that in the former experiment the values $m_i, v_i, \widehat{m_i}, \widehat{v_i}, c_i, f_{\mathsf{bad},i}, f_{\mathsf{del},i}, f_{\mathsf{out},i}$ are read from the internal storage of $\mathcal{S}_2$ (playing the role of $\mathcal{F}_{\mathsf{CFM}}$), whereas in the latter experiment these values are specified by the attacker. Hence, by a standard argument, for all PPT distinguishers $\mathcal{D}$:

$$\Big|\Pr\left[\mathcal{D}(\mathtt{HYBRID}^{2}_{\mathcal{A}(z),\mathcal{S}_2,I}) = 1\right] - \Pr\left[\mathcal{D}(\mathtt{REAL}_{\Pi_{\mathsf{DFM}},\mathcal{A}(z),I}) = 1\right]\Big| \leq \Pr[\mathtt{Bad}].$$

The proof of the lemma now follows by a union bound. ■

Combining the above two lemmas, we obtain that the real and ideal experiment are computationally close, as desired. ■

C. Estimating the Cost of Computation

To estimate the burden of computation, we observe that the summary of each data point from the THTR is described by a tuple $\langle d, n_p, n_c, n_m, n_t\rangle$ where:

- d is the trading date,
- n_p is the number of post orders (# increases),
- n_c is the number of cancelled orders (# decreases),
- n_m is the number of matched orders (# actual trades),
- and n_t is the number of traders.

As the plain implementation cannot go beyond 10 traders we have assumed that only 10 traders could actually participate (so we cap n_t at 10). With this cap made, we estimate the required computation in a naïve MPC implementation according to Fig. 2 as follows.

- For **Post/Cancel Order**, an order requires 3 sub-steps per trader which yields $3n_t$ sub-steps to process an order.
- Similarly, for **Match Order**, an order requires 2 sub-steps per trader hence $2n_t$ sub-steps to match a trade.
- In each sub-step, one phase must walk through $\frac{n_p}{2}$ orders in average (These operations contribute to most of the generic MPC overhead).

They are then multiplied by the time $\tau^{mpc}(n_t)$ required by the elementary MPC operation $\mathcal{F}_{\mathsf{dtc}}$.

Differently, while generic MPC requires *all traders to compute for one trader*, the hybrid protocol allows traders to produce and verify the proof *by themselves at the same time* so the cost is not affected by n_t. The proof generation time $c^{hyb}_{i,gen}$ and the proof verification time $c^{hyb}_{i,ver}$ are actually performed by different traders. This is is not important to calculate the overall crypto-overhead of operating the market in a distributed fashion (before moving to the next order both operations must be done) but will be important for the calculation of the proportional burden.

Therefore the total time to process a trading day d reported in Fig. 7 of the single trader follows the equations below:

$$\begin{aligned} T^{mpc}_d &= \frac{n_p}{2}\left(\sum_{i=p,c} n_i 3n_t + n_m 2n_t\right)\tau^{mpc}(n_t) \\ T^{hyb}_d &= \sum_{i=p,c,m} n_i\left(\tau^{hyb}_{i,gen} + \tau^{hyb}_{i,ver} + \tau^{hyb}_{i,mpc}(n_t)\right) \end{aligned}$$

and similarly for the optimized version where the costs have estimated according to Table XI.

To estimate the fraction of retail and institutional traders ρ_t we use the data from [41] as well as the fraction of orders performed by retail traders ρ_o ($\rho_t = 0.71$, $\rho_o = 0.18$). Albeit TSX and CME are different exchanges, the skeweness against retail traders might be even more pronounced the more active the market. For the MPC naïve computation a party has to participate to the computation irrespectively to whether she actually made any order. So the overall burden of computation by retail traders for naïve MPC (Fig. 8) is as follows:

$$R_d^{mpc} = \rho_t n_t T_d^{mpc}$$

In the hybrid approach, a retail trader only needs to verify proofs when an institutional trader has to generate proof, hence the computation by retail traders (Fig. 8) is determined by the following equation:

$$R_d^{hyb} = \rho_t n_t \sum_{i=p,c,m} n_i \left(\rho_o \tau_{i,gen}^{hyb} + (1-\rho_o)\tau_{i,ver}^{hyb} + \tau_{i,mpc}^{hyb} \right)$$