Multi-Party Computation Mechanism for Anonymous Equity Block Trading: A Secure Implementation of Turquoise Plato Uncross

John Cartlidge[1], Nigel P. Smart[1,2], and Younes Talibi Alaoui[2]

[1] University of Bristol, Bristol, UK.
[2] KU Leuven, Leuven, Belgium.
john.cartlidge@bristol.ac.uk, nigel.smart@kuleuven.be, younes.talibialaoui@kuleuven.be

Abstract. Dark markets for share trading provide an important function in the financial services industry, however the operators of such markets are not operating in the dark; leaving the possibility open for the market operator to gain a market advantage. Multi-Party Computation (MPC) is a method to remove the need for such trusted third parties. In this paper we show how MPC can be used to implement such a dark market. Importantly we base our solution on the throughputs and algorithms needed in real markets. In particular we base our implementation on the Turquoise Plato market used by the London Stock Exchange.

1 Introduction

Dark pools consist of private trading venues, where placed orders remain hidden from the public, and exposed only after being matched. Dark pools have emerged in the past few years, offering an alternative for traders seeking to avoid the market impact, i.e., the impact on the price by which an order will be matched, caused by the fact that this was publicly announced.

In the other hand, while orders remain private throughout the auction, the dark pool operator still has full access to the trading system, and thus can see these orders. As a consequence, the transparency of the trading process is put into question, as traders are trusting the dark pool operator to behave honestly, while nothing can stop him from using the confidential information he holds, to his own advantage, i.e, what we call insider trading.

In [7], authors demonstrated that Multi-Party Computation (MPC), can be a potential solution to this issue, by emulating the dark pool operator as a set of organizations, that jointly match orders in a secure manner. They considered three general algorithms used in auctions, to trade one instrument, i.e. one asset being traded in the market, and benchmarked these three algorithms with respect to latency and throughput.

In this paper, we address a more realistic scenario for trading within dark pools, by first, considering more than one instrument traded instead of just one. That is, a stock exchange in general, and particularly, a dark pool is usually dealing with multiple instruments and not only one. Second, we targeted one of the actual algorithms used in dark pool auctions (with minor modifications in order to make it MPC friendly), and not only a general one. The one we considered is the Turquoise Plato Uncross (TPU) used in the London Stock Exchange, that we will describe in the next section. Third, we came up with a protocol to securely send orders from traders to organizations, in such a way that nothing is leaked about these orders, and if a trader or an organization is cheating, this will be detected

To achieve what we are aiming, we will have to face several problems.

First, a logical choice when dealing with many instruments is to consider that the organizations emulating the dark pool will use many MPC engines, in order to keep a separation of the instruments to distribute efficiently the workload of matching orders, making each engine dedicate all its computing power to match the associated instruments to it. This is needed as one MPC engine per instrument would be impractical from an organizational stand point, and one MPC engine for all instruments would be impractical from an MPC stand point. However, this needs to be done carefully without resulting in any leakage of information, particularly, we should not leak any information about the instrument of an order even though we are sending

it to a specific engine, that handles only a set of instruments. This requirement impacts our small divergences from the actual market operation mechanism done in the real Plato exchange.

Second, by emulating the dark pool as a set of organizations, these will perform an MPC protocol on inputs coming from other parties, i.e. traders. Those inputs must remain hidden to the organizations during computation. This specific use case is different from what we usually find in the literature, where inputs come from the same parties doing the MPC computation. Therefore, we need to address this carefully in order to provide a secure protocol.

2 Background

2.1 Dark Liquidity

Successful trading in the financial markets requires balancing the conflicting objectives of finding a counterparty with whom to trade, while not disclosing one's trading intention. The majority of exchange venues simplify the process of finding a counterparty by maintaining a *public limit order book* (PLOB), which displays all orders currently available in the market. However, in these public (or *lit*) exchange venues, as soon as a trader submits a new order, information about the trader's intention to trade (the desire to buy or sell some quantity at a particular price) is immediately disclosed. This can be a problem if the order size (or *volume*) is relatively large, as other traders in the market are likely to react to this information by moving the price in the adverse direction. For example, a large volume sell order signals to the market that there is an excess supply, and traders will quickly reduce their own order prices in anticipation of a subsequent fall in price. This reaction to the discovery of a large order is known as *price impact*, or *market impact*, and the costs to a trader can be severe, far outweighing commission fees and other trading charges. It has been estimated that market impact increases approximately with the square root of volume [15], although accurate calculations of market impact from empirical trading data are notoriously difficult and there is no consensus on the exact functional relationship between volume and impact (for a review of price impact, see [6]; for a technical discussion, see [5, Chapters 11 and 12]). Nevertheless, it is universally accepted that publicly exposing one's intention to trade, particularly when trading in large volume, is extremely costly and best avoided.

Traditionally, to avoid market impact when attempting large trades, traders would pay a trusted broker to find a counterparty within their network of contacts. As long as a broker network keeps all order information secret (as long as there is no information *leakage*), then a trade can occur with little or no market impact, since the wider market is unaware of trading intention until *after* the trade executes. However, there is a strong financial incentive for brokers to misuse the *privileged* information of their clients' confidential orders. By selling a client's order information to a third party, or by using the information directly to *front run* a client's trade, brokers can profit at the direct expense of their client. Although often difficult to prove, such (illegal) activity is not uncommon. In 2005, twenty specialists on the New York Stock Exchange (NYSE) were charged with committing thousands of illicit front running trades between 1999 and 2003, causing millions of dollars of customer losses [49].

Front running describes acting in advance on confidential trading information for one's own gain. For example, let us assume broker, B, is instructed by client, C, to buy 20,000 shares in XYZ, and the current PLOB is displaying the following offers to sell: $5,000$@\$49; $15,000$@\$50. If acting honestly, B will execute C's order by purchasing $5,000$ shares at \$49 and $15,000$ shares at \$50, for a total cost to C of \$$995,000$. However, since B knows that C's buy order will move the market (i.e., the large buy order will have a *positive* price impact), B decides to *misuse* C's intention to trade by front running the purchase. To this end, on their own account, B buys $5,000$ shares at \$49 and simultaneously posts an offer to sell $5,000$@\$50. The order book for XYZ now displays sell offers: $20,000$@\$50, allowing B to execute C's request to purchase $20,000$ shares at \$50, for a total cost of \$1 million to the client. Broker B has immediately sold their shares (to the client, and at the direct expense of the client) for a risk-free profit of \$$5,000$. This practice is illegal but can be extremely difficult to detect, particularly if B uses a third party (with no obvious connection to B or C) to front run the trade.

Table 1. SEC Penalty Settlements for US Dark Pool Operators (2011-2019).

Company and Settlement	Illegal Activity
ITG (POSIT) Nov 2018, $12m [47]	**Information misuse:** ITG disclosed confidential trading information on dark pool, POSIT, by offering reports on the prior day's trading activity to HFT firms. **Mechanism misuse:** ITG secretly split POSIT into two separate non-interacting dark pools.
Citigroup (Citi Match) Sep 2018, $12m [46]	**Mechanism misuse:** Nearly half of Citi Match orders were secretly routed to other venues, with trade confirmation messages edited to indicate these orders executed on Citi Match. **Mechanism misuse/misleading customers:** Citigroup misled users with assurances that HFT were not allowed to trade in Citi Match dark pool, when two of Citi Match's most active users reasonably qualified as HFT and executed more than $9bn of orders through the pool.
Merrill Lynch (Instinct X) Jun 2018, $42m [48]	**Mechanism misuse:** Merrill Lynch (a broker-dealer) secretly routed customer orders to external venues, with trade confirmation messages edited to indicate these orders had executed in-house: a process they called *masking*. In total, $141bn of transactions masked.
Deutsche Bank (SuperX+) Dec 2016, $18.52m [45]	**Mechanism misuse:** Coding error in the Dark Pool Ranking Model of dark pool order router, SuperX+, caused two dark pools to receive inflated rankings and consequently millions of orders that should have been routed elsewhere. (SuperX+ is a dark router, not a dark pool.)
Barclays Capital (LX) Jan 2016, $35m [42]	**Mechanism misuse:** Barclays failed to police its dark pool, LX, from predatory trading and lied when stating LX only used direct data feeds from exchanges (to deter latency arbitrage), after inquiries were generated by the publication of Michael Lewis's book, *Flash Boys*.
Credit Suisse (Crossfinder) Jan 2016, $54m [43, 44]	**Information misuse:** Credit Suisse transmitted confidential order information in Crossfinder to internal system Crosslink, which alerted HFT firms to the existence of Crossfinder orders. **Mechanism misuse:** 117 million illegal sub-penny orders were executed in Crossfinder dark pool.
ITG (POSIT) Aug 2015, $20.3m [40]	**Information misuse:** ITG's secret trading desk, Project Omega, accessed live feeds of order information on dark pool, POSIT, and used it to implement HFT strategies, including one that traded against POSIT subscribers.
UBS (ATS) Jan 2015, $14.4m [41]	**Mechanism misuse:** UBS offered secret PrimaryPegPlus orders to HFT firms, which enabled HFT to jump ahead of other participants in the dark pool by placing illegal sub-penny orders.
LavaFlow (ECN) Jul 2014, $5m [38]	**Information misuse:** LavaFlow allowed an affiliate to access and use confidential trading information in their dark pool, through a smart order routing service, ColorBook, which had access to the non-displayed orders of LavaFlow ECN.
Liquidnet (Liquidnet) Jun 2014, $2m [39]	**Information misuse:** Liquidnet sought to find additional sources of liquidity for its dark pool by offering Liquidnet subscribers' intentions to buy or sell securities to firms looking to execute large equity capital markets transactions.
eBX (LeveL) Oct 2012, $0.8m [37]	**Information misuse:** eBX allowed a third party Order Routing Business (ORB) to access confidential trading information in their dark pool, LeveL. The ORB used unexecuted order information on LeveL to route orders for its own benefit.
Pipeline (Pipeline) Oct 2011, $1m [36]	**Information misuse:** The majority of shares traded in the dark pool were executed by a wholly owned subsidiary, which used the side and price of Pipeline subscribers' orders to front-run them by trading on the same side in other venues before filling them on Pipeline.

To circumvent these malicious and predatory practices of human brokers, the first *dark pool* crossing networks emerged in the 1980s. These alternative electronic trading exchanges automatically match orders in *private*. Unlike the visible orders entered into the PLOB of a lit exchange, orders entering a dark pool remain invisible, and details of trades are only published *after* execution. As trading intention remains secret, even large orders can execute in a dark pool with little or no market impact. The attraction of dark pools is clear, and the demand from traders is strong. Over the last decade, largely driven by regulation changes (RegNMS, USA, 2005; MiFID, EU, 2007) and the rise of high-frequency trading (HFT), the number of dark pool venues and the volume they trade has ballooned. In the US, around 40 dark pool venues now operate with approximately 15-18% market share of equities trading (a quadrupling since 2005); while in the EU, the volume traded on the fifteen major dark pools accounts for over 8% of total value traded in equities (a rise from less than 1% in 2009) [30].

However, where there is trust, there is the possibility of abuse. Although dark pools offer trading in secrecy away from prying eyes, the operators of dark pools are trusted to maintain data integrity and not misuse the confidential information inside. For many operators, temptation has proven too great. As detailed in Table 1, between October 2011 and November 2018, US dark pool operators paid more than $217 million in penalty

settlements to the Securities and Exchange Commission (SEC) for illegal practices, including: (a) directly misusing customers' information for front running (*Pipeline*, 2011 [36]; *LavaFlow*, 2014 [38]; *ITG POSIT*, 2015 [40]); (b) selling customers' confidential information to third parties (*eBX*, 2012 [37]; *Liquidnet*, 2014 [39]; *Credit Suisse*, 2016 [43]; *ITG POSIT*, 2018 [47]); and (c) selling preferential access to customers' orders to predatory traders (*UBS*, 2015 [41]; *Credit Suisse*, 2016 [44]; *Barclays Capital*, 2016 [42]; *Deutsche Bank*, 2016 [45]; *Merrill Lynch*, 2018 [48]; *Citigroup*, 2018 [46]). We can consider cases (a) and (b) as *information misuse* (misusing customers' confidential information for the dark pool providers' own gains), and case (c) as *mechanism misuse* (misusing the dark pool trading algorithm in a way that is detrimental to customers, such that customers would be unlikely to use the dark pool if they were aware of how it was being operated).

2.2 Secure Auctions: Related Work

Given the significant financial and regulatory incentives for finding a commercially viable solution to counter the problems of mechanism and information misuse in financial markets (and, more generally, in online auction venues for e-commerce), it is perhaps little surprise that there is now more than two decades of research dedicated to securing auction integrity through cryptographic protocols. This research can be roughly categorised into two themes [28]:

Secrecy-preserving correctness: an auction operator can prove outputs (i.e., trades) are correct given the rules of the market and inputs (i.e., orders), without revealing any information about those inputs. The operator publishes an encrypted audit trail that enables observers to validate the correctness of the auction mechanism. These protocols combat mechanism misuse. However, information misuse is possible.

Strong secrecy: an auction operator is unable to release any additional information about inputs (i.e., orders) other than that implied by the outputs (i.e., trades). As the operator has no access to internal information (i.e., orders), these protocols guarantee no information misuse. Strong secrecy, as the name suggests, provides greater security than secrecy-preserving correctness, but poses a much greater computational challenge to achieve.

Secrecy-preserving correctness: ensuring mechanism integrity The majority of work on secrecy-preserving correctness follows the *Evaluator-Prover* (EP) framework [28], where: (i) traders secretly submit encrypted input order values $x_1, \ldots, x_n$ to the EP (the auction operator); (ii) the EP executes the auction by computing a function $y = f(x_1, ..., x_n)$, before outputting value y, and engaging in a proof of the correctness of the result; finally (iii) the proof of correctness is made publicly available for anybody to verify. To ensure secrecy-preservation, proofs must not reveal anything about the input (i.e., the *orders*) except for the information implied by the output value (i.e., the *trade*). To achieve this, proof protocols use Paillier's homomorphic encryption for zero-knowledge proofs [26], which allows computation on ciphertexts and generates an encrypted result which, when decrypted, matches the result of the operations as if they had been performed on the plaintext.

In 2007, Thorpe and Parkes introduced an EP model for a continuous double auction (CDA) mechanism with limit order and market order types [33]. Before posting order $O(p, q, d)$, the trader encrypts order price, p, quantity, q, and direction, d (bid/buy or ask/sell), using the market operator's public key.[1] The encrypted order, $E(p, q, d)$, is then sent to the exchange, whereby the market operator privately decrypts E to obtain O. Order O is entered into a limit order book (LOB) and matching is performed *in the clear* on the plaintext O. Post-execution, trades are published in encrypted form, such that observers can validate the correctness of the market operation by proof checking that encrypted inputs (orders) match the expected encrypted output (trades) given the published CDA auction mechanism. An empirical evaluation of the proposed protocol running on low-end contemporary commodity hardware suggested an implementation of the system would have operational costs of approximately 5 cents (US) to place and verify an order. Later extensions by the authors and their colleagues included a combinatorial auction mechanism (trading *baskets* of stocks) [34] and

[1] For market order types, value p is not needed.

the ability to enter more sophisticated *conditional rule-based* order types [35]. The EP model has also been applied to simpler sealed-bid auctions, with calculation times reported at: approximately 1 minute per order (using homomorphic encryption for proofs) [27]; 500 milliseconds per order (using a *random representation* protocol for proofs, which is faster than homomorphic encryption but has the drawback of allowing the proof to be performed only once) [32]; and 0.02 milliseconds per order (using an improved random representation protocol, in which the proof can be performed any number of times) [31]. However, in all cases the underlying encryption protocol remained unchanged: traders are required to post orders encrypted using the operator's public key, and the operator matches orders *in the clear*. Therefore, while the post-trade audit trail is secure, the real-time market information is not; thus enabling the possibility of information leakage, or front-running traders' order flow.

To increase information security of the EP model, several extensions have been proposed. For sealed bid auctions, a *delayed private key revelation service* (DPKRS) is used to ensure that the operator cannot decrypt incoming orders (and therefore cannot access order information) before the time of auction close [27].[2] After close, keys are revealed to the operator via the automated DPKRS, and the auction is performed on plaintext orders, as usual. This approach guarantees pre-trade information privacy in a one-shot auction, useful for a unique high-valued item such as a work of art, but when there are a series of repeated auctions with returning participants (a characteristic of financial markets), information leakage is still possible; as Hal Varian (Chief Economist at Google since 2002) explains, *"Even if current information can be safeguarded, records of past behaviour can be extremely valuable, since historical data can be used to estimate willingness to pay"* [50]. To further address information leakage in the EP model (and to approach strong secrecy guarantees), in 2015, Parkes et al. proposed that operators could host auction software on *Trusted Computing* infrastructure; essentially a secure *"computer in a cage"* installed in a physically secure location, with digitally signed software, a secure processor, and with ongoing and publicly observable automated monitoring [28]. However, this approach essentially pushes the issue of trust from the operator to a third party (the Trusted Computing infrastructure), rather than provably guaranteeing strong secrecy, and the authors themselves describe a possible security attack vector and note that Trusted Computing infrastructure is still in its infancy. For these reasons, we conclude that the EP model *does not* provide a credible opportunity for strong secrecy in financial markets; a necessity for guaranteeing no information leakage in dark pool trading venues.

Strong secrecy: ensuring information integrity The problem of avoiding information leakage in financial markets has motivated several studies investigating the potential of multi-party computation (MPC) to achieve auction protocols with strong secrecy. MPC approaches enable $n > 1$ parties to jointly compute a function over their inputs (i.e., orders) while keeping those inputs private (i.e., orders remain in encrypted form, such that no individual party is able to access the plaintext without *colluding* with other parties). Using MPC to operate a trading venue requires computation to be distributed across n servers and guarantees secrecy as long as *at most* t servers are corrupt. We refer to t as the *threshold trust* or *fault tolerance* of the system.

Early work on MPC for secure auctions focused on simple sealed-bid auctions, commonly used online. In 1998, an MPC protocol was proposed with fault tolerance $t < n/3$, therefore ensuring that with $n = 4$ parties no single party can cheat or violate privacy [16]. Shortly afterwards, a novel two-party protocol was proposed ($t = 1$), making use of *garbled circuits* to significantly reduce the required rounds of communication between parties [25]; followed by a two-party scheme without garbled circuits [20], shown to be an order of magnitude faster than [25], but offering a lower level of confidentiality. However, despite progress, none of this work was implemented as a practical application.

Bogetoft and colleagues proposed MPC protocols for one-shot double auctions, with fault tolerance $t < n/2$. These protocols were the first to demonstrate practical success (protocol design [4]; application [3]; commercialisation [29]). The double auction has two periods: (i) open period, where limit orders are submitted in encrypted form; and (ii) close period, where trades are executed at the market clearing price,

[2] The EP model does not enable operators to perform computation on encrypted inputs (i.e., orders). Therefore, DPKRS is unsuitable for continuous double auctions as computation occurs immediately upon order entry. There is no *before* and *after* time for key revelation in an asynchronous, continuous market.

calculated to minimise excess demand and supply.[3] In the first real-world application [3], a system was developed for Danish farmers to trade contracts for sugar beet production on a nation-wide market. The role of the auctioneer is played by $n = 3$ parties ($t = 1$): (i) Danisco, the only sugar beets processor on the Danish market; (ii) DKS, the sugar beet growers' association; and (iii) SIMAP, the research project team. During the open period, 1229 farmers submitted orders. Auction computation was performed on 14 Jan 2008, and approximately 25,000 tons of production rights changed ownership. Timings for the live auction computation were not presented, but on contemporary commodity hardware, tests showed computations for 1000 traders took around 30 minutes; and for 3000 traders around 75 minutes. The protocols have since been commercialised through a private company, Partisia [29], which continues to offer a double auction mechanism with single clearing price using MPC. The one-shot double auction mechanism is particularly suited to a one-off high stake auction with sealed bids and long auction durations. According to Partisia's website, their platform has been tailored for the Norwegian Spectrum Auction to trade spectrum rights for a total of NOK 877.983.276 (approximately USD $100 million) over the course of 7 days and 83 bidding rounds in December 2015.[4]

Protocols for double auctions are further developed by Jutla, to enable repeated (periodic) auctions [18]. In Jutla's proposal, the market protocol is a secure $n = 5$ party computation, run by four brokers and one regulating authority (e.g., the SEC), which can audit saved computations and validate if they were performed according to the protocol. During each auction period, traders can enter limit and market orders. At the end of each auction round, the market is cleared (at a single price) and price and volume information is revealed. Uncleared orders remain in the market for the following auction period. Jutla argues that current (2015) MPC technology *is* capable of running repeated periodic double auctions for financial markets, using a 30 minutes opening auction, followed by a succession of 15 minute auctions, with 5 minute gaps in-between for processing and information digestion.[5] However, Jutla's protocol is not implemented or empirically tested in this work, and to date the work has not been published.[6]

Recently, in 2019, the current authors introduced 2-party and 3-party MPC protocols for three auction mechanisms most commonly used in financial markets: (i) continuous double auction (CDA); (ii) periodic double auction (with clearing price calculated to maximise volume traded); and (iii) periodic volume match (a simple auction protocol with size priority and no price discovery, where volume is cleared at a price determined by some reference value, such as the current mid-price on the London Stock Exchange) [7]. Empirical evaluation of a simple market containing a single traded instrument demonstrated that the CDA protocol can process between 10 and 50 orders per second (depending on the size of the order book); the periodic double auction protocol can process around 500 orders per second; and finally, the simple periodic volume match protocol can process around 800 orders per second, a throughput that may be viable for some real-world dark pools. These results are promising, suggesting that MPC may finally be ready for real-world application in financial markets.

Finally, MPC has also been used by Massacci et al. (2018) to implement a secure futures market using distributed ledger technology, with traders hiding behind a Tor network to communicate anonymously [24]. Designed to replicate the functionality of the Chicago Mercantile Exchange (CME), the system uses a CDA mechanism for order matching. However, unlike previous approaches, discussed above, the focus of this work is on enabling anonymity of *who* is executing a trade, rather than securing *what* and *how much* is being traded; with MPC only used for a small subset of the operations to enable this privacy. Whilst addressing part of the security problem, the methodology still requires a trusted third party with access to secret inputs of all participating traders, and therefore *does not* ensure information integrity. A proof of concept implementation

[3] Clearing price minimises excess demand and supply. At price p, if aggregate demand ΣD_p is greater than aggregate supply ΣS_p then excess demand $ED_p = \Sigma D_p - \Sigma S_p > 0$ and excess supply $ES_p = 0$. If $\Sigma D_p - \Sigma S_p < 0$ then we have excess supply $ES_p = \Sigma S_p - \Sigma D_p > 0$ and $ED_p = 0$.

[4] For details, see: `https://partisia.com/spectrum-auctions/`

[5] These timings follow the open-auction period (30 minutes) of specialists on the New York Stock Exchange (NYSE).

[6] Personal communication with the author, Oct 2018.

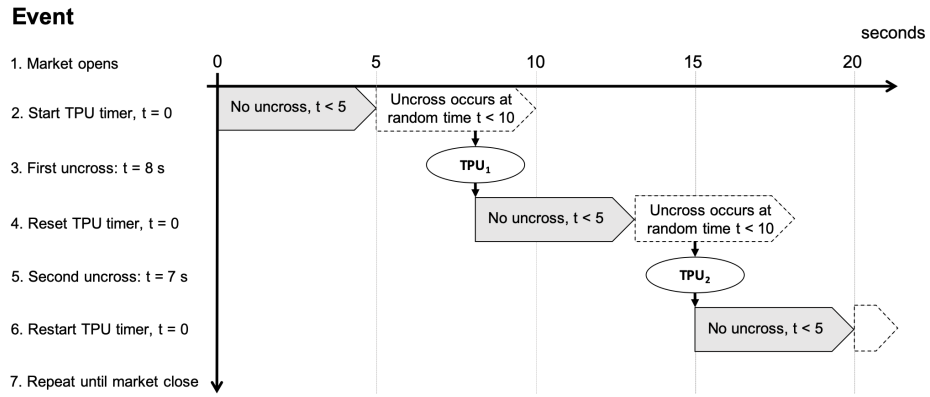

Fig. 1. Turquoise Plato Uncross. Uncrossing occurs at random intervals with period t drawn from a uniform distribution with minimum 5 seconds and maximum 10 seconds, i.e., $t = T \sim U(5, 10)$. Here, the first uncrossing TPU_1 occurs after 8 seconds. The second uncrossing TPU_2 occurs 7 seconds later. This process repeats until close.

is demonstrated, containing a population of 10 traders and an order book with five levels. Results show that individual operations (e.g., post order, cancel order, etc.) can be performed in around 24s.[7]

In summary, with such significant (albeit *illicit*) financial rewards on offer, information and mechanism misuse by dark pool operators is likely to continue until the opportunity for misuse is removed. However, the only way to *guarantee* that there is no information misuse is to *ensure* that *nobody*, not even the dark pool operator, can gain access to the data inside the system. One mechanism to achieve this data privacy is to apply a multi-party computation (MPC) technique to the underlying dark pool algorithm, such that internal order data is held in secret-shared form, and is processed by a set of servers. If a given ratio (depending on the precise MPC system) of the servers remains *honest* then the internal algorithm variables do not leak, and privacy is thus preserved. All orders can be entered into using a protocol to convert an external user's order into a secret shared form. MPC can then be used to perform computation (order matching) on the secret shared data, such that no order information is ever *in the clear*. In addition, if one of the MPC servers is run by a financial regulator (such as the SEC or FCA), then not only is information privacy guaranteed (assuming the regulator remains honest), but the regulator can also guarantee that a specific algorithm was used to perform the matching. These dual guarantees ensure that an MPC-driven dark pool is *provably secure* from both information misuse and mechanism misuse, of the kind detailed in Table 1. In this paper, we introduce an MPC implementation of *Turquoise Plato Uncross* (TPU), one of Europe's largest dark pool trading venues, to demonstrate the potential for MPC to ensure dark pool integrity in financial markets.[8]

2.3 LSEG's Turquoise Plato Uncross

Turquoise Plato (TP) is a suite of non-displayed (i.e., dark) services offered by the *London Stock Exchange Group* (LSEG). Within TP, there are three related and interacting services: *Turquoise Plato Uncross* (TPU), *Turquoise Plato Block Discovery* (TPBD), and *Turquoise Plato Lit Auctions* (TPLA). TPU provides randomised uncrossings during the trading day and is designed for larger and less time sensitive orders to trade in private to avoid market impact. TPBD offers a service for matching clients' undisclosed large block order indications, which are then sent to TPU for execution. Finally, TPLA is an intraday series of periodic random

[7] For comparison, CME Globex report an average median latency for order entry of 200 microseconds during 2017 [10, p.2].

[8] We choose to implement TPU because of its dominant market position and well-defined matching mechanism. The authors are not suggesting that TPU are engaging in information or mechanism misuse.

auction uncrossings based on TPU. To simplify our implementation, we focus our attention on TPU only, which is the main execution logic underlying the Turquoise Plato platform.

TPU executes orders using an uncrossing mechanism that is performed at random intervals throughout the day, with minimum period of 5 seconds and maximum period of 10 seconds between each uncross. We present this visually in Figure 1, where time from market opening is shown on the x-axis, and events are listed from top to bottom on the y-axis. At the start of the day, the market opens (top) and the TPU timer is initialised at $t = 0$. While $t < 5$ (represented by the grey box) we are guaranteed that no uncross will take place as this is shorter than the minimum interval of 5 seconds. Then, for the next 5 seconds, i.e., while $5 \leq t \leq 10$, there is a window, during which uncrossing can occur at any time (represented by the white box with dashed borders). In the example shown, the timer is randomly stopped at $t = 8$ and the first uncross TPU_1 occurs 8 seconds after opening (shown as a white ellipse). The timer is reset to $t = 0$, and the process repeats. First, there is a guaranteed period of 5 seconds with no uncross (grey bar), followed by a 5 second window when a cross will randomly (white bar). This time, the timer is stopped randomly at $t = 7$ and the second uncross TPU_2 occurs $t = 8 + 7 = 15$ seconds after market opening. This process repeats throughout the day until market close.

Each order entered into TPU (requests to buy or sell a particular quantity of an instrument) contains an *instrument* (the stock to trade) a *direction* (buy or sell), a *size* (the quantity to buy/sell) and a *minimum execution size* (MES), which is the quantity below which the order will not execute. For each of U instruments traded within TPU, orders entered during the order insertion phase will rest in a *hidden* limit order book for that instrument. Orders in the book are prioritised by size and then time of order entry (if two orders have the same size, the first order entered into the system will take priority, i.e., it will be positioned higher in the limit order book). An example is shown in Figure 2. In Figure 2(a), we see that six orders are entered during the order insertion phase; three *bids* (orders to buy) and three *asks* (orders to sell). These orders enter the limit order book with size priority, such that in Figure 2(b), we see that the largest bid (id=5) with size $100,000$ and the largest ask (id=6) with size $50,000$ are at the top. The first sell order entered (id=1) has the smallest size and so is placed at the bottom of the order book. This is the state of the order book at the end of the order insertion phase, immediately before the uncrossing phase. During the uncrossing phase, Figure 2(c), orders execute subject to MES thresholds, such that a buy order of MES b (resp. a sell order of MES c) can only be matched with sell orders of volume w s.t $b \leq w$ (resp. buy orders of volume v s.t $c \leq v$). We see that two trades execute: a trade of size $50,000$ executes between bid id=5 and ask id=6, and a further trade of size $1,000$ executes between bid id=4 and ask id=2. The transaction price for these trades is set as the instantaneous mid-price on the primary reference exchange (for example, the London Stock Exchange, for shares listed in the UK). In Figure 2(d) we see the order book immediately after the uncrossing phase, with executed orders removed. In cases where an order is partially matched and the remaining volume becomes smaller than the MES, then the order is removed as it can no longer be matched. Other orders that are fully or partially unexecuted remain in the order book for the next insertion phase, and new orders are inserted in the order book as they arrive.

2.4 Turquoise Plato Uncross: Trading Data and Statistics

In this section, we aim to elaborate the trading dynamics of TPU, so that we are in a position to validate our later MPC implementation against real-world commercial demands. Trading data for TPU, and dark pools in general, is commercially sensitive and accurate high-resolution data is therefore difficult to obtain. However, all trading venues report aggregated data, and from this we are able to estimate some trading statistics of TPU.

The European Central Bank (ECB) reported in 2017 that the share of European equity trading conducted on dark pools has expanded rapidly in recent years, growing from less than 1% in 2009 to over 8% in 2016 [30, p.5]. In June 2016, Turquoise Dark traded 1.37% of total equity volume traded in Europe [30, p.25], equivalent to approximately 17% of all volume traded in 15 dark pools active in Europe at the end of 2016 [30, p.22]. This makes TPU one of the largest (by trade volume) and most successful dark pool trading venues in Europe.

ID	Direction	Size	MES
#1	Sell	100	50
#2	Sell	1,000	1,000
#3	Buy	10,000	5,000
#4	Buy	1,000	100
#5	Buy	100,000	50,000
#6	Sell	50,000	10,000

(a) Orders entered

Bids			Offers		
ID	MES	Size	Size	MES	ID
#5	50,000	100,000	50,000	10,000	#6
#3	5,000	10,000	1,000	1,000	#2
#4	100	1,000	100	50	#1

(b) Order book before uncross

TradeID	BuyerID	SellerID	Size
#T1	#5	#6	50,000
#T2	#4	#2	1,000

(c) Trades executed by uncross

Bids			Offers		
ID	MES	Size	Size	MES	ID
#5	50,000	50,000	100	50	#1
#3	5,000	10,000			

(d) Order book after uncross

Fig. 2. Example Turquoise Plato Uncross: (a) six orders are entered during the *insertion phase*, with order ID incremented each time; (b) orders enter the order book, sorted by size priority such that the largest buy order (ID=5) and the largest sell order (ID=6) are positioned at the top; (c) during the *uncrossing phase*, two trades are executed. A trade of size 50,000 between buyer ID=5 and seller ID=6, and a trade of size 1,000 between buyer ID=4 and seller ID=2; (d) after uncrossing, three orders remain in the order book and a new insertion phase begins.

Table 2. LSEG monthly trading reports for Turquoise Plato. Reproduced from [22, 23].

Month	Trading Days	Total Number of Trades (Avg Daily)	Value Traded (Avg Daily)	Value Traded (Month)	Mean Trade Size
Feb 2017	20	66,307	€ 651 million	€ 13,029 million	€ 9,818
Feb 2020	20	71,416	€ 1,237 million	€ 24,733 million	€ 17,321

Table 3. Turquoise Plato data for Feb. 2017, reproduced from *LiquidMetrix: Guide to European Dark Pools* [21].

Turquoise Plato (Feb 2017)	Month	Avg. Daily
Total Number of Trades	1 326 140	66 307
Total Value Traded (EUR millions)	13 037	652
Number of Unique Instruments Traded (N)	1927	1258
Effective Number of Instruments Traded (E)	91	55
Mean Trade Size (EUR)	9831	
Median Trade Size (EUR)	4114	

Table 2 summarises Turquoise Plato trading data from LSEG's monthly trading reports. We see that over the last three years, the total number of trades per day has only increased by 7.7%. However, the total value traded each day has increased by 90%, which has produced an almost doubling in mean trade size over three years. Yet, it is pertinent to note that the mean trade size on TPU is a misleading statistic, as it is heavily skewed by a relatively small number of very large in scale trades, which are particularly encouraged by the Turquoise Plato Block Discovery (TPBD) service. In February 2017, TPBD had an average daily value traded € 199 *million*, average trade size of € 768,783, and average daily number of trades of 154.8. [11, p.12]. LSEG report that the record trade size on TPU is € 17.33 *million*, executed on 28 June 2018. Further details of large in scale trades are reported in the 2018 *Parliamentary Review* [2] for trading on 16 Mar 2018 in Spanish company, Santander. On that day, Santander's top ten trades by value all executed on

Turquoise Plato, with a combined share of trading (SoT) of 6.4%; i.e., only 10 trades on TPU accounted for more than 6% of the value of all Santander trading, across all market venues, executed that day.

More detailed statistics for TPU trading are shown in Table 3 for February 2017. Comparing with Table 2 for the same month, we can see that these figures are consistent with LSEG trading reports, but also offer further insight into the skewed nature of trading towards a relatively small number of stocks. While in March 2020, LSEG report that Turquoise Plato enables clients to trade a broad universe of $U = 4,500$ stocks across 19 major European and emerging markets (note: this number was likely lower in Feb 2017, with estimations from public reports giving $U \approx 3,000$), we see that only $1,927$ stocks traded during the month of Feb 2017, and on average only $1,258$ stocks traded each day. Therefore, the majority of instruments available to trade on TPU have no executions during each trading day.

Table 3 presents the *effective number of instruments traded*, E, over a given period. This is calculated as the reciprocal of the Hirfindahl index, H:

$$H = \sum_{i=1}^{N} s_i^2 \tag{1}$$

where s_i is the proportion of trading in each stock/instrument, i. Note that, if the proportion of trading is uniform across all stocks, i.e., $\forall i : s_i = 1/N$, then $H = 1/N$, and the effective number of stocks $E = 1/H = N$. When the proportion of trading is skewed heavily towards a small number of stocks, then the effective number of stocks traded, E, will be much lower than the actual number of stocks traded, i.e., $E \ll N$. From Table 3, we see that $E = 90.7 \ll 1,927 = N$ for the month of February and $E = 54.8 \ll 1,258 = N$ for average daily trading. Therefore, it is clear that trading is heavily focused in a small number of instruments. Each day, the majority of instruments on TPU trade rarely, or not at all.

2.5 TPU Implementation Requirements

Here, we capture the requirements necessary for an MPC implementation of TPU, in order to ensure that the technology can be applied commercially. We use real-world TPU trading data, presented in Section 2.4, to ensure that the MPC system can handle similar order throughput and trading activity. Assumptions and simplifications are described, below.

TPU trades for 8.5 hours per day, between 08:00-16:30 (UK time). Since auction uncrossing occurs at random every 5 to 10 seconds, each day there are a minimum of $3,060$ uncrossings (every 10 seconds) and maximum of $6,120$ uncrossings (every 5 seconds) per instrument. To ensure that the system is capable of real-world throughput, we consider the *worst case* scenario, such that auction uncrossing occurs every 5 seconds exactly. Therefore, we assume there are $6,120$ auctions per day per instrument, and the MPC system has a maximum of 5 seconds to handle order entry, order book insertion, and order book uncrossing.

From Table 3, we see that in Februrary 2017, TPU executed an average of $66,307$ trades per day. The intra-day trading volume on TPU (not shown) has two peaks, the first at open (08:00) and the second shortly before close (15:00), with average volume traded during these peak hours roughly twice the size of volume traded during off-peak hours [21]. We simplify to assume uniform intra-day trading volume, with an average of $66307/6120 \approx 10$ trades per auction interval, across all instruments. As discussed previously, Table 3 also shows that most trading occurs in a very small number of instruments, with the majority of instruments trading very rarely, or not at all.

Since TPU is designed for larger, less time-sensitive orders, we assume that all orders, given long enough, will eventually trade. Since TPU matches on volume alone (with trade price taken using mid-price of the primary exchange as reference value), an order will execute as soon as a counterparty enters an order for the opposite direction with volume greater than MES. Therefore, we estimate the average number of new orders entered into TPU to be twice the number of trades, i.e., approximately 20 orders per interval, across all instruments.

To ensure no information leakage, the MPC implementation requires that after each uncrossing, all uncompleted orders are wiped from the order book and must be re-entered into the system to take part in the next uncross. In the worst case scenario (although *unlikely*), we have a situation where every instrument has multiple orders on one side of the order book only (e.g., all orders are bids, or all orders are asks), such

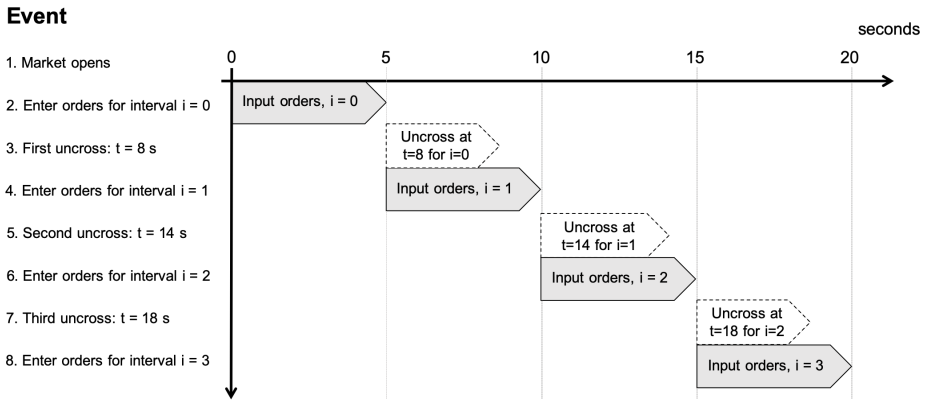

Fig. 3. Modified Turquoise Plato Uncross. Orders are entered in five second intervals and the next set of orders are entered while the uncrossing occurs. Note, the order book is cleared on every uncrossing, with unmet and partially executed orders re-entered in the following interval.

that none are able to execute, and all have to be re-entered in the next interval. In Table 3 we see that the average number of unique instruments trading each day is $1,258$, and the number of instruments trading each month is $1,927$. Therefore, we consider a situation where $2,000$ instruments each have one order in the order book as a worst case scenario for the system to handle.

In summary, we assume that any MPC implementation of TPU that can be applied commercially must be capable of offering trading in a Universe of $U = 4,500$ instruments, and within 5 seconds be able to handle $2,000$ orders entered and an average of 10 trades per uncross, without leaking any information. In the following section, we introduce an MPC implementation to meet these requirements.

3 Preliminaries

3.1 Auction Modifications

To enable the required throughput to be achieved via an MPC system we need to make a minor modification to the way the Plato system works. Instead of the operation given in Figure 1, we adopt the methodology given in Figure 3. In particular we divide the day into five second time intervals. In time interval i orders are entered, then in time interval $i+1$ the uncrossing occurs for the orders entered in time interval i, and new orders for time interval $i+1$ are entered. An important difference, to maintain security of our solution, is that the order book is flushed on every uncrossing. Thus, unmet and partially executed orders at interval i need to be re-entered into the system at interval $i+1$.

3.2 Architecture

The setup we will consider is that of n organizations $\mathsf{Org} = \{\mathsf{O}_1, \ldots, \mathsf{O}_n\}$, which wish to run the dark-pool in a distributed manner via MPC. The n organizations do this by creating $L+1$ entities $\{\mathsf{E}_0, \ldots, \mathsf{E}_L\}$ each of which is instantiated as an MPC 'engine'. The engine E_i consists of $\{\mathsf{P}_i^1, \ldots, \mathsf{P}_i^n\}$, where party/server P_i^j is run by organization O_j. We will refer to the parties/servers $\{\mathsf{P}_i^1, \ldots, \mathsf{P}_i^n\}$ constituting E_i as $\mathcal{P}_i$. All entities P_0^j and P_i^j for $i \neq 0$ are connected by pairwise secure channels, meaning secret shared values held by P_0^j can be passed securely to P_i^j.

The engine E_0 is a special *gateway engine*, whilst engines $\mathsf{E}_1, \ldots, \mathsf{E}_L$ deal with the actual auction, so we refer to these L engines as *auction engines*. The reason for requiring multiple engines to deal with the

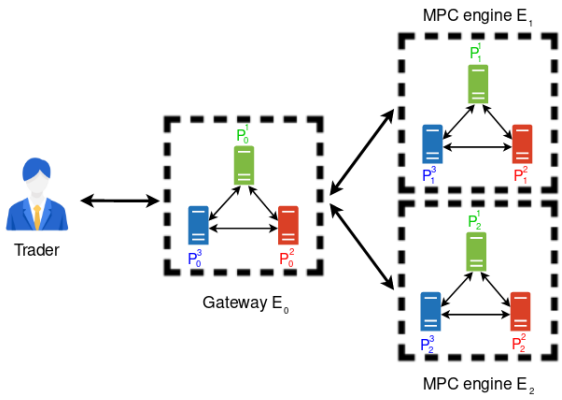

Fig. 4. Example setup for $L = 2$ auction engines, and $n = 3$ dark pool operators. A trader submits an order for instrument i to Gateway E_0 in secret shared form. The order is allocated to the auction engine running TPU for i.

auction is to enable a high enough throughput to be reached when the market is dealing with a large number of instruments. We do this by distributing the instruments between the different engines, however we need to do so in a way which avoids linkage between orders between different time intervals. It is to enable this that we utilize L auction engines, which itself gives rise to the complications coming in sub-protocols Π_{prep} and Π_{inp} below.

A trader T from the set of potential traders Tr places his order into the auction through secure channels with $\mathcal{P}_0$. As an example instantiation consider Figure 4 which depicts the setup for the case where $L = 2$, and $n = 3$.

The instruments (for example specific stocks) are pulled from a given fixed set $S = \{1, \ldots, U\}$. In any one time interval we will divide the set S up into disjoint subsets $S = S_1 \cup \ldots \cup S_L$ and assign the set of instruments S_i to engine E_i. We let R denote the size of S_i, where we assume for ease of exposition that $|S_i| = R$ for all i. This division is carried out by the gateway engine E_0 in such a way that a qualified set of the organizations do not learn which instrument is assigned to which engine. This is vital to stop organizations learning information about unmet orders; by changing the allocation in each time interval we also stop correlations being obtained by the organizations. However, in one time interval the assignment of an order to an engine will leak *some* information (which we explicitly model below).

3.3 Protocol overview:

In what follows, we will give a high level overview of the protocol that organizations and traders need to execute. In later sections we will show how it is implemented. We focus on the operations in a given time interval. These consist of four distinct operations:

- Before the insertion phase:
  1. Assign the instruments to the engines; this we call sub-protocol Π_{prep}.
- During the insertion phase:
  2. Take as input an order from a trader and get the gateway engine E_0 to pass it on to the correct auction engine E_i; we call this sub-protocol Π_{inp}.
  3. Each E_i now needs to insert each incoming order into its order book; we call this sub-protocol Π_{ins}.
- During the uncrossing phase:
  4. Finally each engine needs to implement the uncrossing phase; we call this sub-protocol Π_{unc}.

The sub-protocols Π_{ins} and Π_{unc} are essentially taken from [7], thus for now we concentrate on the sub-protocols Π_{prep} and Π_{inp}. The protocol Π_{prep} is pre-processing, and thus we assume this is done before the specific time interval, for example over night. For all incoming orders in a specific time interval the set of steps

in Π_{inp} need to be executed for all incoming orders in the five second time window. During the uncrossing phase we need Π_{ins} and Π_{unc} also to be completed within the permitted five second interval. However, this is an overestimate as Π_{ins} can be run in parallel with Π_{inp}, especially as Π_{inp} puts the main computational strain on engine E_0, whilst Π_{ins} is purely an operation on E_i for a given $i \geq 1$.

The sub-protocol Π_{prep} obliviously computes an assignment from the set of instruments to the set of engines. For ease of notation we write this assignment as $\phi : S \longrightarrow \{1, \ldots, L\}$, where an instrument $r \in S_i$ if and only if $\phi(r) = i$. We emphasise that this mapping ϕ is not known to any of the organizations, thus $\mathcal{P}_0$ will be oblivious to the sets $S_1, \ldots, S_L$. Hence organizations will not know which engine is dealing with which instruments in this given time interval, however the organizations will learn which orders get assigned to which set (but not the orders specific instrument). In practice $\mathcal{P}_0$ will engage in a sub-protocol at the end of which they will obtain the sets S_i as secret shared indices in $S = \{1, \ldots, U\}$.

The sub-protocol Π_{inp} needs to take an ord of instrument r from a trader T and then secret share it to the auction engine $\mathsf{E}_{\phi(r)}$. However, this needs to be done without $\mathcal{P}_0$ learning the instrument r in this particular order, or the mapping $\phi(r)$. One way to address this, is to simply send ord to every engine, and thanks to the equality test implicit with Π_{ins}, the order ord will be inserted in exactly one engine which is the right one. However, this will clearly degrade the performance of our solution as each engine will have to deal with every order, and we actually introduced the gateway to filter orders in order to send them to the right engine.

An alternative consists of revealing to E_0 the instruments that every set S_i contains. However, from this information, parties in E_0 can determine the corresponding engine without having to open the instrument of ord. Thus we consider this an unacceptable leak of information, since we can determine sets of instruments among which trades will occur, i.e, instruments of S_i if the gateway is sending orders to E_i.

Therefore to implement Π_{prep} and Π_{inp} we need to be a little more involved. Our solution consists of having $\mathcal{P}_0$ obliviously construct a vector of encryptions of the $\phi(j)$, given by $\mathbf{c} \leftarrow (\mathsf{Enc}_{\mathsf{Pk}}(\phi(1)), \ldots, \mathsf{Enc}_{\mathsf{Pk}}(\phi(U)))$, with a homomorphic encryption scheme which supports distributed decryption and re-randomization of ciphertexts. In our instantiation we do this using Paillier encryption [26], see Section 3.5 below. This vector needs to be constructed without leaking anything about the sets S_i. This is the output of our sub-protocol Π_{prep}.

For the Π_{ins} sub-protocol this vector is sent by $\mathcal{P}_0$ to the trader. He will select the r-th component of this vector $\mathbf{c}_r$, and he re-randomizes it to obtain $\mathbf{c}'_r$, before sending it back to $\mathcal{P}_0$. Finally, $\mathcal{P}_0$ will perform a distributed decryption on $\mathbf{c}'_r$ to find $\phi(r)$. Note that re-randomization from the trader is necessary here so as to avoid correlating orders belonging to the same instrument.

3.4 Modeling the leakage:

Before going any further in this work, we need to define which events will be considered as a leakage of information and thus break the security of our protocol, and which events will not. The best case scenario is to consider that any information leaked about an order that is still not matched is a leakage of information. However, while this is theoretically possible, it will result in a protocol that is not efficient, while our goal is to come up with a protocol that is practical to implement for the real-world TPU. To this end, we formally define in Figure 5, how we modeled the leakage of information. Therefore, the security evaluation of our work follows this model, i.e., our protocol is deemed secure as long as no leakage of information occurs, aside what we permitted in Figure 5.

Informally the leakage when $L = 1$ is what one would expect for such an auction; one can determine which orders are buy and sell orders and one knows at the end which orders have been matched, with all other data remaining hidden. This is captured by items 1–4 in Figure 5. When $L \geq 1$ there is the additional leakage of information captured in item 5; this additional leakage comes from our distribution of the orders into U/L buckets corresponding to each engine. Note, in the extreme case of $L = U$, i.e. when we have one engine per instrument, the adversary will learn which instrument corresponds to which order. Thus the larger L is then the more information leaks, but the less computational resources are needed to run the auction, whereas when L is small less information leaks, but the auction may not be able to be completed

For an order ord that enters to the auction within time interval t, the adversary can:

1. Determine whether ord is a buy or a sell order.
2. If ord is a sell order (resp. a buy order), the adversary can determine whether ord satisfies the matching requirement with every order ord$'$ from the buy list (resp. the sell list).
3. If ord is matched with ord$'$, the adversary will end up knowing the name, MES, instrument and the matched volume of both orders after the orders enterred in time interval t are uncrossed. However, the adversary is not allowed to learn which orders were partially matched and which orders were completely matched.
4. Determine a bound (upper and/or lower bound) on the volume of ord.
5. In time interval t the instruments are divided into U/L subsets, which are unknown to the adversary. However, the adversary can learn to which subset each order is associated with.

Any other information that leaks about ord will be considered as an unexceptable leakage of information.

Figure 5. Leakage model.

in enough time. Thus we obtain a form of $R = U/L$-anonymity on the instruments associated to an order[9]. One question we aim to answer is what is a good value for L in such an auction.

3.5 Cryptographic Background

We will assume that the parties in $\mathcal{P}_i$, for $i = 0, \ldots, L$ are probabilistic polynomial time Turing machines. We will refer to sampling uniformly at random an element r from a set X by $r \leftarrow X$. We also denote by $a \leftarrow b$ normal variable assignment, i.e. assigning the value of variable b to the variable a. If $\mathcal{D}$ is a probability distribution over a set X, we denote by $a \leftarrow \mathcal{D}$ sampling from X with respect to the distribution $\mathcal{D}$. We denote by $\mathbf{v}^1 \odot \mathbf{v}^2$, the component-wise multiplication of two vectors, i.e. $\mathbf{v}^3 \leftarrow \mathbf{v}^1 \odot \mathbf{v}^2$, where $\mathbf{v}^3_i = \mathbf{v}^1_i \cdot \mathbf{v}^2_i$. Finally, $\underline{b}^a$ will denote a vector of size a where each component of it is equal to b.

Paillier Scheme Our solution for sending orders to the corresponding engines is inspired from the Helen system [51]; where they needed to convert data back and forth between secret shared form and encrypted form using a partially homomorphic with distributed decryption procedure. Such a scheme can be instantiated using Paillier encryption [26]. In the Helen protocol to verify that parties behaved honestly during these conversions, the parties run the MAC check protocol on encrypted data, by using the homomorphic property of Paillier encryption, and by performing a distributed decryption on the result of the MAC check protocol to check if it is correct.

In this work we also need to convert data from secret shared form to encrypted form as we will later, and for this purpose we will use the Paillier scheme as well. However, our approach to detect cheating is completely different, we actually avoid running the MAC check protocol on encrypted data, by taking advantage of the nature of the computations we are performing.

In what follows, we will formally define the Paillier scheme, along with the properties it satisfies.

Encryption Scheme: A probabilistic public key encryption scheme is a set of algorithms (KeyGen, Enc, Dec), such that:

- $\mathsf{KeyGen}(1^\lambda)$ generates a public key and a private key $(\mathsf{Pk}, \mathsf{Sk})$, with respect to some security parameter λ.
- $\mathsf{Enc}_{\mathsf{Pk}}(m, r)$ outputs a ciphertext c encrypting the message m with randomness r, under the key Pk.
- $\mathsf{Dec}_{\mathsf{Sk}}(c)$ outputs the decryption of the ciphertext c under the key Sk.

For correctness we require that the decryption algorithm satisfies the following:

[9] It is not quite the same as R-anonymity as, whilst you know an order is for an instrument from a set of R possible instruments, no party knows what R instruments are in the set.

- $\mathsf{Dec}_{\mathsf{Sk}}(\mathsf{Enc}_{\mathsf{Pk}}(m, r)) = m$ for any randomness r.

Throughout the paper, we may also refer to the encryption of a message m under the key Pk by $\mathsf{Enc}_{\mathsf{Pk}}(m)$, dropping from the notation the randomness used. We will also abuse notation, by referring to a vector that contains encryptions under Pk of the components of a vector $\mathbf{v}$ by $\mathsf{Enc}_{\mathsf{Pk}}(\mathbf{v})$, and to a matrix that contains encryptions under Pk of the entries of a matrix M by $\mathsf{Enc}_{\mathsf{Pk}}(M)$. The scheme is considered secure (in the IND-CPA sense) if no adversary can distinguish whether a given ciphertext c is the encryption of message m_0 or message m_1.

Partial Homomorphic Encryption: A partially homomorphic encryption scheme is an encryption scheme as defined above, with an extra requirement:

- For an operation $\oplus$ that defines a group over the plaintext space $(G, \oplus)$, and an operation $\otimes$ that defines a group over the ciphertext space $(G', \otimes)$, $\mathsf{Dec}_{\mathsf{Sk}}(\mathsf{Enc}_{\mathsf{Pk}}(m_1, r_1) \otimes \mathsf{Enc}_{\mathsf{Pk}}(m_2, r_2)) = m_1 \oplus m_2$, for any two messages m_1 and m_2, and any randomnesses r_1 and r_2.

This basically means that we can perform computation on the ciphertexts without having to decrypt them. And it also means that we can re-randomize ciphertexts. For instance, for the case where $\oplus$ consists of addition, and $\otimes$ consists of multiplication, we can re-randomize a ciphertext c by multiplying it by $\mathsf{Enc}_{\mathsf{Pk}}(0_G, r)$ for some r drawn at random, where 0_G is the identity element of $(G, \oplus)$.

Encryption Scheme with Distributed Decryption: For a set of parties $\{\mathsf{P}^1, \ldots, \mathsf{P}^n\}$, an access structure $\mathcal{A}$ is (monotonically increasing) subset of $2^{\{\mathsf{P}^1, \ldots, \mathsf{P}^n\}} \setminus \{\emptyset\}$, i.e., $\mathcal{A}$ is a collection of non empty sets $\mathcal{C}_j$ of $\{\mathsf{P}^1, \ldots, \mathsf{P}^n\}$. The sets $\mathcal{C}_j = \{\mathsf{P}^{1j}, \ldots, \mathsf{P}^{|C_j|j}\}$ for j in $1, \ldots, |\mathcal{A}|$ are called the authorized sets. A public encryption scheme, is said to have distributed decryption over parties $\mathsf{P}^1, \ldots, \mathsf{P}^n$, with respect to an access structure $\mathcal{A}$, if we can provide two protocols:

- Π_{KeyGen}: a protocol that securely implements the KeyGen algorithm, i.e. it outputs a public key Pk, and for every authorized set $\mathcal{C}^j$, it outputs to every P^{ij}, a share Sk^{ij} of the private key Sk.
- Π_{Dec}: a protocol that securely implements the $\mathsf{Dec}_{\mathsf{Sk}}$ algorithm for every authorized set $\mathcal{C}_j$. It takes as a public input a ciphertext c, and as a private input, the shares of the secret key of one of the authorized sets $\mathcal{C}^j$, i.e., Sk_{ij} of the parties P^{ij}, then it outputs $m = \mathsf{Dec}_{\mathsf{Sk}}(c)$ to the parties in $\mathcal{C}^j$.

The security requirement is that a ciphertext should remain semantically secure, i.e. it reveals no information, to any subset of $\{\mathsf{P}^1, \ldots, \mathsf{P}^n\}$ which is not contained in the access structure $\mathcal{A}$. For instance, for the case where $\mathcal{A}$ contains only one authorized set, namely all parties in $\{\mathsf{P}^1, \ldots, \mathsf{P}^n\}$, then we require all parties to implement the decryption algorithm.

The Paillier Scheme: Paillier is an encryption scheme that is both partially homomorphic and has distributed decryption. We will consider here the Damgård-Jurik variant of it [13], as it offers flexibility regarding the size of the plaintext space. To show how Damgård-Jurik works, we will explain it through the simplified version given in [13]; note if we take $e = 1$ then we obtain Paillier's original scheme.

- KeyGen: Take N as the product of two prime numbers q_1 and q_2. Set $\mathsf{Pk} \leftarrow N$ and $\mathsf{Sk} \leftarrow \mathsf{lcm}((q_1 - 1), (q_2 - 1))$.
- Enc: For a message m in $\mathbb{Z}_{N^e}$, where $e > 0$ is an integer, take randomness $r \leftarrow \mathbb{Z}^*_{N^{e+1}}$, and compute $\mathsf{Enc}_{\mathsf{Pk}}(m, r) \leftarrow (1 + N)^m \cdot r^{N^e}$.
- Dec: For a ciphertext c in $\mathbb{Z}^*_{N^{e+1}}$, compute d such that $d = 1 \mod N^e$ and $d = 0 \mod \mathsf{Sk}$. Then compute $c^d \mod N^{e+1}$ to obtain $(1 + N)^m \mod N^{e+1}$. We can then extract the discrete logarithm of this value to obtain m, which is feasible for this case.

The security of rests on the DCR assumption, which itself is believed to be as hard as finding the factorization of N. Thus one selects N of around 2048 bits to obtain a suitable security.

Distributed versions of this scheme are available, against active adversaries, for both honest and dishonest majority, see [13], [17]. In addition it is easy to see that this scheme is partially homomorphic, i.e.

$$\mathsf{Enc}_{\mathsf{Pk}}(m_1, r_1) \cdot \mathsf{Enc}_{\mathsf{Pk}}(m_2, r_2) = (1 + N)^{m_1 + m_2} \cdot (r_1 \cdot r_2)^{N^e} = \mathsf{Enc}_{\mathsf{Pk}}(m_1 + m_2, r_1 \cdot r_2).$$

Protocol $\mathcal{F}^{\mathcal{P}}[\mathsf{MPC}]$

The functionality runs with $\mathcal{P} = \{\mathsf{P}^1, \ldots, \mathsf{P}^n\}$ and an ideal adversary $\mathcal{A}$, that statically corrupts a set A of parties. Given a set I of valid identifiers, all values are stored in the form $(varid, x)$, where $varid \in I$.

Initialize: On input $(init, p)$ from all parties, the functionality stores $(domain, p)$,

Input: On input $(input, \mathsf{P}^i, varid, x)$ from P^i and $(input, \mathsf{P}^i, varid, ?)$ from all other parties, with $varid$ a fresh identifier, the functionality stores $(varid, x)$.

Add: On command $(add, varid_1, varid_2, varid_3)$ from all parties (if $varid_1, varid_2$ are present in memory and $varid_3$ is not), the functionality retrieves $(varid_1, x)$, $(varid_2, y)$ and stores $(varid_3, x + y)$.

Multiply: On input $(multiply, varid_1, varid_2, varid_3)$ from all parties (if $varid_1, varid_2$ are present in memory and $varid_3$ is not), the functionality retrieves $(varid_1, x)$, $(varid_2, y)$ and stores $(varid_3, x \cdot y)$.

Output: On input $(output, varid, i)$ from all honest parties (if $varid$ is present in memory), the functionality retrieves $(varid, y)$ and outputs it to the environment. The functionality waits for an input from the environment. If this input is $\mathsf{Deliver}$ then y is output to all players if $i = 0$, or y is output to player i if $i \neq 0$. If the adversarial input is not equal to $\mathsf{Deliver}$ then $\varnothing$ is output to all players.

Figure 6. Operations for Secure Function Evaluation.

This property is extremely useful, for instance, for an encrypted matrix $C \leftarrow \mathsf{Enc}_{\mathsf{Pk}}(M)$ in $\mathcal{M}_{\{R,R\}}$ and a vector $\mathbf{v}$ of length R, we can compute $\mathbf{c} \leftarrow \mathsf{Enc}_{\mathsf{Pk}}(M \cdot \mathbf{v}^{(t)})$ by simply computing $\mathbf{c}_i \leftarrow \Pi_{j=1}^{j=R} C_{i,j}^{\mathbf{v}_j}$. And from vectors of ciphertexts $\mathbf{c}^1, \ldots, \mathbf{c}^R$, where $\mathbf{c}^i = \mathsf{Enc}_{\mathsf{Pk}}(\mathbf{v}^i)$, we can compute $\mathsf{Enc}_{\mathsf{Pk}}(\sum_i \mathbf{v}^i)$, by computing $\mathbf{c}^1 \odot \ldots \odot \mathbf{c}^R$.

Note, that if we consider the plaintext elements as integers of given size, then we need to ensure that the homomorphic operations we apply do not produce wrap-around, i.e. produce values which exceed N^e in absolute value. Namely, the result of a homomorphic operation is only meaningful if the modulus N^e is chosen to avoid such wrap around in the application.

Multi-Party Computation We consider Secret Sharing based Multi-Party Computation (MPC) protocols with abort against active adversaries. This means that inputs of the parties remain private throughout the execution of the protocol, and when a set of adversaries deviate from the protocol honest parties will catch this with overwhelming probability and then abort from the protocol. This should be compared to passively secure protocols which offer a much weaker guarantee that security is only preserved if all parties follow the precise protocol steps correctly. We present in Figure 6 the base MPC functionality.

The SCALE-MAMBA framework: As in [7], we will consider the SCALE-MAMBA system [1] to run our experiments. SCALE implements multiple MPC protocols realizing $\mathcal{F}^{\mathcal{P}}[\mathsf{MPC}]$. In the secret sharing based ones, computation is taking place in a prime field $\mathbb{F}_p$, a value $x \in \mathbb{F}_p$ secret shared among the parties in $\mathcal{P}$ will be denoted as $\langle x \rangle$. SCALE works in the pre-processing model, which means that there are two phases within SCALE, an offline and an online phase:

- In the offline phase, input independent data are produced, such as random values, i.e. $\langle r \rangle$ such that $r \leftarrow \mathbb{F}_p$. Beaver triples, which are random multiplication triples of the form: $\{\langle a \rangle, \langle b \rangle, \langle c \rangle\}$ such that $a \leftarrow \mathbb{F}_p$, $b \leftarrow \mathbb{F}_p$ and $c = a \cdot b$. And random bits, i.e. $\langle b \rangle$ such that $b \leftarrow \{0, 1\}$. This data will be further consumed in the online phase when a multiplication of secret shared values is required.
- In the online phase, actual computation takes place on inputs of the parties. In SCALE, addition is a local operation, and multiplication requires communication between parties, consuming Beaver triples generated from the offline phase.

The nature of the MPC protocol is such that a value x held in secret shared form $\langle x \rangle$ is authenticated. This means that if a party attempts to change it, then this will be detected. In fact, the probability that a

party cheats without being caught is equal to $\frac{1}{p}$, thus, p has to be big enough to ensure an overwhelming probability. Further in this section we will see how we selected p.

The way authentication is achieved depends on the underlying secret sharing scheme, the two schemes we will be considering for this work are Shamir Secret Sharing based MPC following the methodology of [19], and the SPDZ protocol of [14] and its follow-up papers. These two protocols are fairly different, in terms of the access structures they support. We will briefly outline these two approaches, and explain why the probability of cheating without being caught corresponds to $\frac{1}{p}$.

Shamir Secret Sharing Based MPC: In Shamir Secret Sharing based MPC each entity holds a share $x_i \in \mathbb{F}_p$ of a secret x where we have that x is the constant term of a polynomial $f_x(X)$ of degree t such that $x_i = f(i) \pmod p$, i.e. $x = f_x(0)$. We write $\langle x \rangle$ to denote a sharing of x in this way. Clearly, if $t+1$ parties come together they can recover the polynomial $f_x(X)$ via interpolation, and then recover x from $f_x(0)$. It is also clear that parties can compute arbitrary *linear* functions of their shares without interaction.

To produce the multiplication triples in the offline phase one requires that $t < n/2$, in which case the parties generate two random sharings $\langle a \rangle$ and $\langle b \rangle$ and then each party produces the product of their local shares, the parties then reshare the results, and compute a specific linear function of the resulting n sharings. On its own this only provides passive security, but the basic protocol can be made actively secure with abort with very little additional overhead, see [19] for example.

SPDZ Based MPC: The above system works when $t < n/2$, when $n/2 \leq t < n$ we require a different methodology, which is what the SPDZ system provides. Here each party P^i holds a share $\alpha_i \in \mathbb{F}_p$ of a global Message Authentication Code (MAC) key. The MAC key is defined as $\alpha = \sum_i \alpha_i$. A value $x \in \mathbb{F}_p$ is then secret shared among the organizations as the tuple $\{x_i, \gamma_i\}_{i \in [n]}$, such that $x = \sum_i x_i$ and $\sum \gamma_i = \alpha \cdot x$. We call $\alpha \cdot x$ the MAC value on x. We use the notation $\gamma_i[x]$ (resp. $\gamma[x]$), if we want to refer to a MAC share γ_i of γ (resp. the MAC γ), specifying at the same time the value x on which γ is a MAC. We again write $\langle x \rangle$ for this sharing so as to unify notation and allow us to treat both situations at the same time.

Linear computation on shared values is again straightforward to perform, in particular, given two secret shared values x and y and three field constants $a, b, c \in \mathbb{F}_p$ we can compute the sharing of $z = a \cdot x + b \cdot y + c$ locally by each player computing:

$$\begin{aligned} z_1 &\leftarrow a \cdot x_i + b \cdot y_i + c && \text{for } i = 1 \\ z_i &\leftarrow a \cdot x_i + b \cdot y_i && \text{for } i \neq 1 \\ \gamma_i[z] &\leftarrow a \cdot \gamma_i[x] + b \cdot \gamma_i[y] + \alpha_i \cdot c && \text{for all } i. \end{aligned}$$

The production of the multiplication triples is now much more involved, and makes use of a Homomorphic Encryption scheme; for more details about SPDZ, we refer to [14]

Comparison between SPDZ and Shamir Based MPC: For Shamir Secret Sharing based MPC, we assume that there is a honest majority, i.e. among the n parties participating in the protocol, at least a threshold of $\lceil \frac{n+1}{2} \rceil$ parties are honest. If this is not ensured, the security of the protocol collapses and the protocol no longer provides any security guarantees. Whereas for SPDZ, it works with a Full Threshold setting, i.e. we tolerate up to $n-1$ parties to be malicious. This means that, all the claimed security guarantees hold, as long as there is at least one honest party.

Authentication of the shared values is done in different ways in the two familes. For Shamir Secret Sharing based MPC, we have an honest majority which provides a direct method to provide authentication on the underlying secret shared values; essentially using the error-detecting properties of the underlying Reed-Solomon encoding. In fact, detecting cheating for an opened value is done by simply checking the consistency of the shares. Roughly speaking, if the shares were correctly computed, recombining them will yield the secret, otherwise, no value from $\mathbb{F}_p$ can be the combination of those shares. Thus, a malicious party can guess a value that was not opened only with probability $\frac{1}{p}$.

For SPDZ, given that it does not assume a honest majority, detecting cheating is not as straightforward as for Shamir Sharing. To obtain the same form of authentication the SPDZ protocol introduced MACs into the secret sharing. That is, each opened value goes through the MAC check protocol. The soundness of this protocol comes from the fact that it is hard to forge a MAC. This means that an opened value y can only have a valid MAC if there was no cheating while computing it. The probability of cheating without being

Table 4. Costs of operations in SCALE, with parameters $\mathsf{sec} = 40$, $k = 64$, and $\log_2 p = 128$.

Operation	Open	$\langle a \rangle \cdot \langle b \rangle$	$\langle a \rangle < \langle b \rangle$ $\langle a \rangle < b$	$\langle a \rangle = \langle b \rangle$ $\langle a \rangle = b$
Triples	0	1	120	63
Bits	0	0	105	104
Rnds of Comm.	1	1	7	7

caught corresponds to guessing the MAC Key α. As α is drawn at random from $\mathbb{F}_p$, this probability is equal to $\frac{1}{p}$.

Arithmetic using SCALE-MAMBA and the size of p: Addition, multiplication and comparison of secret shared values $\langle x \rangle$ and $\langle y \rangle$ will be denoted by $\langle z \rangle \leftarrow \langle x \rangle + \langle y \rangle$, $\langle z \rangle \leftarrow \langle x \rangle \cdot \langle y \rangle$, $\langle z \rangle \leftarrow \langle x \rangle > \langle y \rangle$. Which means form a sharing $\langle z \rangle$ of the result of the operation on the sharings $\langle x \rangle$ and $\langle y \rangle$. We also denote revealing a value to parties by Open $\langle x \rangle$. We will also abuse notation by referring to a vector that contains the secret sharing of the components of a vector $\mathbf{v}$ by $\langle \mathbf{v} \rangle$, and to a matrix that contains the secret sharing of the entries of a matrix M by $\langle M \rangle$.

Whilst computation in our MPC engines takes place over a prime field $\mathbb{F}_p$, however, we actually need to perform computation over integers to emulate matching orders, i.e, to execute Π_{ins} and Π_{unc}. In particular we will need to compute on k-bit integers, and in particular compare them. We will encode an integer in $[-2^{k-1}, \ldots, 2^{k-1}]$ as its standard representative modulo p. Then we need to make sure that no wrap around takes place all along the computation, i.e., the range $[-2^{k-1}, \ldots, 2^{k-1}]$ is big enough to catch all the computation. This is an easy task as in our algorithms, we are performing a number of conditional summations, and hence the maximum size of all values can be known ahead of time.

To perform a comparison operation, such as $\langle b \rangle \leftarrow \langle x \rangle < \langle y \rangle$, SCALE follows the methods of [8, 9, 12], where the comparison operator is implemented using only additions, multiplications and output to all. What is important to mention about this method is that it requires to take a shared value $\langle x \rangle$ for x in $[-2^{k-1}, \ldots, 2^{k-1}]$, and mask it by a value $\langle r \rangle$ by computing $\langle x + r \rangle \leftarrow \langle x \rangle + \langle r \rangle$, then opening $\langle z \rangle \leftarrow \langle x + r \rangle$ so that the parties obtain $x + r$. However, this would be problematic if r is not big enough, as it reveals information about x. In fact, the statistical distance of z from the uniform distribution is $2^{-\mathsf{sec}}$, if r was chosen from the interval $[-2^{\mathsf{sec}+k-1}, \ldots, 2^{\mathsf{sec}+k-1}]$, i.e., r chosen from an interval that is 2^{sec} times larger than the range of x. This parameter sec is called the *statistical security parameter for arithmetic*.

To run experiments, we selected $\mathsf{sec} = 40$ and $k = 64$. Then to ensure valid arithmetic, and an acceptable soundness, we need to select p such that $k + \mathsf{sec} < \log_2 p$, and $1/p$ is a negligible probability. To this end, we picked p such that $\log_2 p = 128$. Considering these parameters, we present in Table 4, the costs of the basic operations within SCALE of the basic algorithms for multiplication, comparision and equality testing that we will need.

4 Emulating the dark pool operator

We now treat each of the sub-protocols realising the phases in turn:

4.1 Allocating instruments to an engine: the Π_{prep} sub-protocol

We will assume from now on that the players in $\mathcal{P}_0$ hold a Paillier key pair $(\mathsf{Pk}, \mathsf{Sk})$, where Sk is distributed among the organizations, and that they have access to a protocol Π_{Dec} to decrypt which is actively secure for the specified access structure $\mathcal{A}$ equivalent to the access structure for the underlying MPC protocol (i.e. threshold (t, n) for Shamir and full-threshold for SPDZ). We assume the Paillier key (N, e) is chosen so that $N^e > n \cdot p$, where n is the number of parties and p is the underlying modulus of the MPC system.

The main idea to build Π_{prep} consists of obliviously generating a permutation π in order to construct the map $\phi : S \longrightarrow \{1, \ldots, L\}$, that will be used to assign the instruments to the engines. Namely, ϕ will be constructed as follows:

$$\pi(1 + R \cdot i) = \ldots = \pi(R + R \cdot i) = \phi^{-1}(i + 1) \text{ for } i = 0, \ldots, L - 1.$$

In order to achieve this, the parties generate an $U \times U$ secret shared permutation matrix M of a random permutation π, by having each party contributing to the construction of M, then producing the same permutation in encrypted form under the Paillier key $(\mathsf{Pk}, \mathsf{Sk})$. The secret shared form will be used to produce the sets $S_1, \ldots, S_L$. That is, $S_{i+1} = \{\pi(1 + R \cdot i), \ldots, \pi(R + R \cdot i)\}$, for $i = 0, \ldots, L - 1$, and the encrypted form to produce the vector $\mathbf{c}$. Recall that we want $\mathbf{c} \leftarrow (\mathsf{Enc}_{\mathsf{Pk}}(\phi(1)), \ldots, \mathsf{Enc}_{\mathsf{Pk}}(\phi(U)))$.

To do so, $\mathcal{P}_0$ in step 1 in Figure 7, produces $\langle M \rangle$, and in step 2 assigns instruments to S_i and send them to E_i. Then parties reproduce M in encrypted form C, by having each party encrypt their share of the matrix M and broadcast it to the other parties. Then, $\mathcal{P}_0$ obtain the vector $\mathbf{c}$ in 5 from the transpose of C. A formal description of this sub-protocol is given in Figure 7.

As Π_{prep} does not deal with any orders it is clear no information about orders can leak at this stage. We however need to prove that, while executing Π_{prep}, the matrix M provided by the parties corresponds to a permutation matrix, and the map ϕ used to compute the vector $\mathbf{c}$, is indeed the same as the one used to produce the sets S_i. These two requirements are crucial for the remaining sub-protocols. Ensuring this was the main reason behind adding the steps 1.III, 1.IV and 4 into Π_{prep}. That is, it is easy to see that if the checks in 1.III and 1.IV go through, a sufficient and necessary condition about M being a permutation matrix is then satisfied. Therefore, proving that Π_{prep} is secure boils down to proving that the check in 4 guarantees that the same map ϕ was used. To prove this, we will need the following trivial Lemma.

Lemma 1. *For the random variables, X, Y and Z, where X follows the uniform distribution over $\mathbb{F}_p$, Y follows some distribution $\mathcal{D}_1$ over $\mathbb{F}_p$, and Z follows some distribution $\mathcal{D}_2$ over $\mathbb{F}_p \setminus \{0\}$, we have:*

- *The variable $H \leftarrow (X + Y)$ follows the uniform distribution over $\mathbb{F}_p$.*
- *The variable $G \leftarrow (X \cdot Z)$ follows the uniform distribution over $\mathbb{F}_p$.*

Theorem 1. *If $N^e > n \cdot p$, the check in step 4.V in Figure 7 is correct and sound, i.e, if $\langle M \rangle$ and C correspond to the same permutation and organizations were honest during the execution of this sub-protocol, then we will have $\mathbf{t}' = \mathbf{m}'$ mod p, and if $\langle M \rangle$ and C do not correspond to the same matrix, we will have $\mathbf{t}' \neq \mathbf{m}'$ mod p except with negligible probability.*

Proof. **Correctness:** Basically, the aim here is to prove that although computation in the secret shared domain is modulo p, and computation is the plaintext domain of Paillier is with a different modulo ($N^e \gg p$), if parties are honest, in step 4.V in Figure 7, we will have $\mathbf{t}' = \mathbf{m}'$ mod p.

If parties are honest, they will provide the same matrix M in secret shared form $\langle M \rangle$ and encrypted form $C = \mathsf{Enc}_{\mathsf{Pk}}(M)$, along with vectors $\mathbf{t}^i$ in secret shared form $\langle \mathbf{t}^i \rangle$ and encryptions $\mathbf{c}^i$ of $M \cdot (\mathbf{t}^i)^t$. Note that $\mathbf{c}^i$ contains the same elements in $\mathbf{t}^i$ shuffled with respect to the permutation matrix M. Having the components of $\mathbf{c}^i$ smaller than p, the ciphertext $\mathbf{c}'$ which encrypts the sum of the plaintexts of $\mathbf{c}^i$, will contain elements that are smaller than $p \cdot n$. Therefore, the plaintext of $\mathbf{c}'$ mod p is equal to the secret shared vector $M \cdot (\mathbf{t})^t$, as there was no wrap around mod N^e in the plaintext domain, given that $N^e > n \cdot p$.
Soundness: To prove then the soundness of Theorem 1, let us assume that an adversary controlling a set of parties, lie about their shares. This can be modeled as having $\langle M \rangle$ and $\mathbf{t}$ in secret shared domain, and $C = \mathsf{Enc}_{\mathsf{Pk}}(M + A)$ and $\mathbf{c}'' = \mathsf{Enc}_{\mathsf{Pk}}(\mathbf{t} + \mathbf{a})$, where $A \neq 0$. If the check in step 4.V goes through, this means that:

$$\begin{aligned}(M + A) \cdot (\mathbf{t} + \mathbf{a})^t &= M \cdot \mathbf{t}^t \\ M \cdot \mathbf{a}^t + A \cdot \mathbf{t}^t + A \cdot \mathbf{a}^t &= 0 \\ A \cdot \mathbf{t}^t &= -M \cdot \mathbf{a}^t - A \cdot \mathbf{a}^t\end{aligned}$$

For a set of instruments $\{1,..,U\}$, parties $\mathcal{P}_0$ of E_0 execute the following:

(1) To generate a secret shared random permutation matrix $\langle M \rangle$. $\mathcal{P}_0$ execute the following:
  (I) For $i = 1, \ldots, n$:
    (A) P_0^i generates a permutation matrix M^i of size $U \times U$ and secret shares it in E_0 to obtain $\langle M^i \rangle$.
  (II) $\mathcal{P}_0$ compute $\langle M \rangle \leftarrow \Pi_1^n \langle M^i \rangle$
  (III) $\mathcal{P}_0$ compute the sum of each row and column of $\langle M \rangle$ and open them. If any of these values is different than 1, the organizations abort.
  (IV) $\mathcal{P}_0$ also computes $\langle M_{j,k} \rangle \cdot (1 - \langle M_{j,k} \rangle)$ for $j \in \{1, \ldots, U\}$ and $k \in \{1, \ldots, U\}$. If any of these values is different than 0, the organizations abort.

(2) To obliviously assign instruments to engines, $\mathcal{P}_0$ execute the following:
  (I) $\mathcal{P}_0$ compute $\langle \mathbf{v} \rangle \leftarrow \langle M \rangle \cdot (1, 2, \ldots, U)^t$
  (II) For $i = 0, \ldots, L-1$:
    (A) $S_{i+1} \leftarrow \{\langle v_{1+R \cdot i} \rangle, \ldots, \langle v_{R+R \cdot i} \rangle\}$
    (B) The set S_{i+1} is sent to parties in the engine E_{i+1}.

(3) Each party P_0^i encrypts its share M_i of $\langle M \rangle$ under Pk, and publish it. Organizations then reconstruct $C \leftarrow \mathsf{Enc}_{\mathsf{Pk}}(\sum_i M_i)$.

(4) To check if C corresponds to $\langle M \rangle$, $\mathcal{P}_0$ execute the following:
  (I) For $i = 1, \ldots, n$:
    (A) P_0^i generates a random vector $\mathbf{t}^i \in \mathbb{F}_p^U$.
    (B) P_0^i inputs $\mathbf{t}^i$ in E_0 to obtain $\langle \mathbf{t}^i \rangle$.
    (C) P_0^i computes $\mathbf{c}^i \leftarrow \mathsf{Enc}_{\mathsf{Pk}}(M \cdot (\mathbf{t}^i)^t)$ (from C and $\mathbf{t}^i$, as in 3.5) and publishes $\mathbf{c}^i$.
  (II) $\mathcal{P}_0$ compute $\langle \mathbf{t} \rangle \leftarrow \sum_i \langle \mathbf{t}^i \rangle$
  (III) $\mathcal{P}_0$ compute $\langle \mathbf{t}' \rangle \leftarrow \langle M \rangle \cdot \langle \mathbf{t} \rangle^t$
  (IV) $\mathcal{P}_0$ compute $\mathbf{c}' \leftarrow \mathsf{Enc}_{\mathsf{Pk}}(M \cdot \mathbf{t}^t)$ (from $\mathbf{c}^i$ as in 3.5).
  (V) $\mathcal{P}_0$ open $\langle \mathbf{t}' \rangle$, then perform a distributed decryption on $\mathbf{c}'$ to obtain $\mathbf{m}'$. If $\mathbf{t}' \neq \mathbf{m}' \bmod p$, organizations abort.

(5) $\mathcal{P}_0$ compute $\mathbf{c} \leftarrow C^{(t)} \cdot \mathsf{Enc}_{\mathsf{Pk}}(\mathbf{v})$, where $\mathbf{v}$ is a vector of size U that is equal to $(\underline{1}^R, \ldots, \underline{L}^R)$.

Figure 7. The Π_{prep} sub-protocol used for allocating R instruments to L engines.

where A and a are chosen by the adversary, and M is a uniformly random permutation matrix, and $\mathbf{t}$ is uniformly random from $\mathbb{F}_p$.

Given that $A \neq 0$, there will be at least one entry $A_{i,j}$ in A, such that $A_{i,j} \neq 0$. Let us consider the i^{th} component of the resulting vectors of the left and right sides of this equation. From the results of 1, the i^{th} component in $A \cdot \mathbf{t}^t$ is uniformly at random, as it is the sum of variables following some distribution over $\mathbb{F}_p$, plus a variable, which is the product of $A_{i,j}$ that follows some distribution and the i^{th} component of $\mathbf{t}$, which is uniformly at random. Thus, the adversary ends up with an equation he needs to satisfy, where the left side (the i^{th} component in $A \cdot \mathbf{t}^t$) is a uniformly random value, and the right side consists of a sum of some variables that follows some distribution $\mathcal{D}$. The probability Pr that the adversary can produce values to make this equation be satisfied is

$$\mathsf{Pr} = \frac{1}{p} \cdot \mathsf{Pr}(0 \leftarrow \mathcal{D}) + \ldots + \frac{1}{p} \cdot \mathsf{Pr}(p-1 \leftarrow \mathcal{D}) = \frac{1}{p}$$

Therefore, the adversary can succeed with at most the (negligible) probability $\frac{1}{p}$. □

4.2 Inputting orders into the system: the Π_{inp} sub-protocol

Inputting an order $\langle \mathsf{ord} \rangle = (\langle \mathsf{name}_0 \rangle, \langle r \rangle, \langle b_0 \rangle, \langle v_0 \rangle)$ from trader T is done as follows: $\mathcal{P}_0$ sends to T the vector $\mathbf{c}$, the trader then re-randomizes the r^{th} component of this vector to obtain c, and sends it back to $\mathcal{P}_0$, along with $\langle \mathsf{ord} \rangle$. Parties in E_0 perform then a distributed decryption on c to obtain $\phi(r)$, and re-share $\langle \mathsf{ord} \rangle$, among engine $\mathsf{E}_{\phi(r)}$.

One issue that needs to be addressed in this scenario, consists of providing a way to traders to securely input their orders into the auction. Basically, the trivial way to perform this consists of drawing a random $\langle r \rangle$ from E_0, and open it to T by having parties in E_0 send their shares of r to T. Then, if T wishes to input m, he computes $d = m + r$ and sends it back to parties in E_0. Parties in E_0 then compute $\langle m \rangle = d - \langle r \rangle$. This reveals nothing about m as d is a random value. However, nothing can stop parties in E_0 to send wrong shares of r to T. This is due to the fact that r did not go through any procedure to check its correctness, unlike what happens when we open an authenticated value to the parties.

The fix for this depends on the underlying protocols we are considering, namely Shamir Sharing or SPDZ. For the case of Shamir, simply having parties in E_0 send their shares of $\langle r \rangle$ will be sufficient, as T will detect cheating thanks to the error detecting property of Shamir. The sub-procedure Send[Shamir]-Sub for this is in Figure 8.

As for SPDZ, the fix is more complex. The MAC check protocol takes use of the fact that $\langle r \rangle$ is opened to all parties within an MPC engine, (there is also a variant of it that allows to check the correctness of a value opened to only one party). However, we cannot execute this protocol on $\langle r \rangle$ in our case as we intentionally do not open the value to any party within the engine, and we cannot reveal the MAC key α to the trader so that he runs himself the MAC check protocol. To cope with this, we introduce The sub-procedure Send[SPDZ]-Sub in Figure 8, which guarantees that

- The MAC α of SPDZ is not compromised.
- If the check in step 4 goes through, the trader is convinced that parties in E_0 sent him correct shares.

This is due to the fact that r is a random value, and α' reveals nothing about α, as it is the product of α with another random value. As for whether parties in E_0 sent the correct shares, the intuition behind why the protocol ensures this, comes from the fact that we let the trader run the MAC check protocol on r, by sending him all the necessary material, where r is MAC'd with the MAC key $\alpha' = \alpha \cdot s$.

The formal description of the Π_{inp} sub-protocol is given in Figure 8. The main complexity comes in the sub-protocol Send[SPDZ]-Sub, which we analyse in the following theorem.

Theorem 2. *The sub-procedure* Send[SPDZ]-Sub *in Figure 8 is correct and sound, i.e, for a honest trader, he accepts the check in step 4 if the shares r_i sent to him correspond to $\langle r \rangle$, otherwise, he rejects except with a negligible probability.*

Proof. **Correctness:** If the parties are honest while executing Send[SPDZ]-Sub, i.e. the shares $\{r_i, \alpha'_i, \gamma_i[\gamma']\}$ correspond to $\{r, \alpha', \gamma'\}$. We will have $\alpha' \cdot r = \alpha \cdot s \cdot r$ and $\gamma' = \alpha \cdot s \cdot r$. This means that the shares sent to T of $\langle r \rangle$ are correct.
Soundness: Let us assume that an adversary controlling a set of parties, lie about their shares, i.e. the shares they sent along with the shares of honest parties sum up to: $\{r'', \alpha'', \gamma''\}$, where $r'' \leftarrow r + \epsilon_1$, $\alpha'' \leftarrow \alpha' + \epsilon_2$, $\gamma'' \leftarrow \gamma' + \epsilon_3$, and $\epsilon_1 \neq 0$, where ϵ_1, ϵ_2, and ϵ_3 are known to the adversary. If the check in step 4 goes through, this means that:

$$\begin{aligned}
\alpha'' \cdot r'' &= \gamma'' \\
(\alpha' + \epsilon_2) \cdot (r + \epsilon_1) &= (\gamma' + \epsilon_3) \\
(\alpha \cdot s + \epsilon_2) \cdot (r + \epsilon_1) &= (\alpha \cdot s \cdot r + \epsilon_3) \\
(\alpha \cdot s) \cdot (r + \epsilon_1) &= (\alpha \cdot s \cdot r + \epsilon_3) - \epsilon_2 \cdot (r + \epsilon_1) \\
\alpha \cdot s &= \epsilon_3 \cdot \epsilon_1^{-1} - \epsilon_2 - \epsilon_2 \cdot r \cdot \epsilon_1^{-1}
\end{aligned}$$

which is an equation of the form $r_1 = f(\epsilon_1, \epsilon_2, \epsilon_3, r_2)$, where r_1, r_2 are uniformly independent random numbers from $\mathbb{F}_p$, and $\epsilon_1, \epsilon_2, \epsilon_3$ are chosen by the adversary. The right side of the equation follows some distribution $\mathcal{D}$. The probability Pr that the adversary can produce values to make this equation be satisfied is

$$\mathsf{Pr} = \frac{1}{p} \cdot \mathsf{Pr}(0 \leftarrow \mathcal{D}) + \ldots + \frac{1}{p} \cdot \mathsf{Pr}(p-1 \leftarrow \mathcal{D}) = \frac{1}{p}$$

Therefore, the adversary can succeed with only the (negligible) probability $\frac{1}{p}$. □

Send[Shamir]-Sub:

Parties in E_0 and T execute the following sub-procedure, so that T inputs a value m into E:

(1) Trader T asks for randomness.
(2) Parties in E_0 draw a random value $\langle r \rangle$.
(3) For $i = 1, \ldots, n$:
  (I) P_0^i sends his shares r_i of r to T.
(4) T checks consistency of the shares r_i.
(5) If the check goes through (i.e. T claims $\mathcal{P}_0$ did not cheat):
  (I) T reconstructs r and sends back $d \leftarrow r + m$.
  (II) Parties in E_0 compute $\langle m \rangle = d - \langle r \rangle$
(6) Else (i.e. T claims $\mathcal{P}_0$ are cheating):
  (I) T sends back to every party in E_0 the signed shares he received from every every party.
  (II) Parties in E_0 check the consistency of those shares.
  (III) If this check goes through, this trader is excluded from the auction (i.e. T is lying and in fact $\mathcal{P}_0$ did not cheat).
  (IV) Else, the organizations abort (i.e. some parties within $\mathcal{P}_0$ are cheating).

Send[SPDZ]-Sub:

Parties in E_0 and T execute the following sub-procedure, so that T inputs a value m into E:

(1) Trader T asks for randomness.
(2) Parties in E_0 draw two random values $\langle r \rangle$ and $\langle s \rangle$ and compute $\langle \alpha' \rangle \leftarrow \langle s \rangle \cdot \langle \alpha \rangle$, and $\langle \gamma' \rangle \leftarrow \langle \alpha' \rangle \cdot \langle r \rangle$, where α is the MAC key of E.
(3) For $i = 1, \ldots, n$:
  (I) P_0^i sends his shares $\{r_i, \alpha'_i, \gamma_i[\gamma']\}$ of $\{r, \alpha', \gamma'\}$ to T.
(4) T recombines r, α', γ' and checks whether $\alpha' \cdot r = \gamma'$.
(5) If the check goes through (i.e. T claims $\mathcal{P}_0$ did not cheat):
  (I) T sends back $d \leftarrow r + m$.
  (II) Parties in E_0 compute $\langle m \rangle = d - \langle r \rangle$
(6) Else (i.e. T claims $\mathcal{P}_0$ are cheating):
  (I) T sends back to every party in E_0 the signed shares he received from every other party.
  (II) Parties in E_0 check whether $\alpha' \cdot r' = \gamma'$.
  (III) If this check goes through, this trader is excluded from the auction (i.e. T is lying and in fact $\mathcal{P}_0$ did not cheat).
  (IV) Else, the organizations abort (i.e. some parties within $\mathcal{P}_0$ are cheating).

The sub-protocol Π_{inp}:

For each order ord with instrument r to be input into the auction by trader T from Tr.

(1) $\mathcal{P}_0$ send T the vector $\mathbf{c}$ computed in Π_{prep}
(2) Trader T chooses the r^{th} component in $\mathbf{c}$. Then re-randomizes it to obtain c. Note if the trader cheats in this phase the worst that can happen is that his order is ignored.
(3) T sends c, and inputs $\langle \mathsf{ord} \rangle$ in E_0, using one of the sub-procedures above depending on the setting considered.
(4) $\mathcal{P}_0$ perform distributed decryption on c to obtain $\phi(r)$, i.e. the index of the engine to which $\langle \mathsf{ord} \rangle$ will be sent.
(5) $\mathcal{P}_0$ re-share $\langle \mathsf{ord} \rangle$ in $\mathsf{E}_{\phi(r)}$.

Figure 8. The Π_{inp} sub-protocol used for inputting orders to the gateway engine E_0.

Thanks to the sub-procedures Send[Shamir]-Sub and Send[SPDZ]-Sub, each order ord is sent to $\mathcal{P}_0$ without being revealed. Using the vector $\mathbf{c}$, the engine E_i to which ord is sent is determined without revealing the corresponding instrument. And thanks to how Π_{prep} was implemented, $\mathcal{P}_0$ are oblivious to the instruments that E_i deals with. Thus, the fact that ord was sent to E_i does not reveal the corresponding instrument.

Given a new buy order $(\langle \mathsf{name}_0^b \rangle, \langle r \rangle, \langle b_0 \rangle, \langle v_0 \rangle)$, input in engine E_l. this algorithm inserts the new order into the sorted list (with respect to the volume) L_r^b of existing buy orders for instrument r.
To do this, we insert first a dummy element at the end of the buy list for all instruments of the form $(\langle 0 \rangle, \langle \mathsf{MAX} \rangle, \langle 0 \rangle)$ (except for instruments that are already of size M), i.e. a buy order of MES MAX, and volume 0.

(1) For $\langle i \rangle \in S_l$
  (I) $\langle d \rangle \leftarrow (\langle r \rangle = \langle i \rangle)$.
  (II) If $\mathsf{size}(L_i^b) < M$ then
    (A) Add $(\langle 0 \rangle, \langle \mathsf{MAX} \rangle, \langle 0 \rangle)$ to the end of the list.
  (III) $\langle f_0 \rangle \leftarrow (\langle v_0 \rangle < \langle v_1 \rangle)$.
  (IV) For $j = 1, \ldots, \mathsf{size}(L_i^b)$ do
    (A) $\langle e \rangle \leftarrow \langle v_0 \rangle \leq \langle v_j \rangle$.
    (B) $\langle f_j \rangle \leftarrow \langle d \rangle \cdot (\langle e \rangle - 1) + 1$.
    (C) $\langle f'_j \rangle \leftarrow (1 - \langle f_j \rangle) \cdot \langle f_{j-1} \rangle$.
    (D) $\langle f''_j \rangle \leftarrow (1 - \langle f_j \rangle) \cdot (1 - \langle f'_j \rangle)$.
    (E) $\langle \mathsf{name}_j^{b\prime} \rangle \leftarrow \langle f_j \rangle \cdot \langle \mathsf{name}_j^b \rangle + \langle f'_j \rangle \cdot \langle \mathsf{name}_0^b \rangle + \langle f''_j \rangle \cdot \langle \mathsf{name}_{j-1}^b \rangle$.
    (F) $\langle b'_j \rangle \leftarrow \langle f_j \rangle \cdot \langle b_j \rangle + \langle f'_j \rangle \cdot \langle b_0 \rangle + \langle f''_j \rangle \cdot \langle b_{j-1} \rangle$.
    (G) $\langle v'_j \rangle \leftarrow \langle f_j \rangle \cdot \langle v_j \rangle + \langle f'_j \rangle \cdot \langle v_0 \rangle + \langle f''_j \rangle \cdot \langle v_{j-1} \rangle$.
  (V) Output $[(\langle \mathsf{name}_j^{b\prime} \rangle, \langle b'_j \rangle, \langle v'_j \rangle)]_{j=1}^{\mathsf{size}(L_i^b)}$ as the buy list of instrument i.

Figure 9. The Π_{ins} sub-protocol for inserting a new buy order into the buy list (sell order insertion is identical).

Otherwise, any other leakage falls in what the adversary is allowed to know, in particular, events 1 and 5 from Figure 5.

4.3 Inserting orders into the order book: the Π_{ins} sub-protocol

We describe here the Π_{ins} sub-protocol, that is executed in every engine E_k for every order ord received. The basic protocol was given and analyzed in [7], the only difference in this paper is that instead of having one instrument per engine we have R instruments per engine. We assume that at any point during the auction, the buy list (resp. the sell list) of each instrument can contain at most M orders (resp. N orders). We will assume as well that no volume can reach a maximum bound MAX. A buy order (resp. a sell order) for instrument r submitted to the auction will be of the form $(\langle \mathsf{name}_0^b \rangle, \langle r^b \rangle, \langle b_0 \rangle, \langle v_0 \rangle)$ (resp. $(\langle \mathsf{name}_0^s \rangle, \langle r^s \rangle, \langle c_0 \rangle, \langle w_0 \rangle)$).

In order to hide how many orders were submitted per instrument, and at the same time to keep lists of instruments separated, we filter in a secure manner each order entering when sending it to the corresponding instrument using an equality check. Therefore, in the insertion phase, we insert $(\langle \mathsf{name}_0^b \rangle, \langle b_0 \rangle, \langle v_0 \rangle)$ of instrument r into the list
$L_r^b = [(\langle \mathsf{name}_j^b \rangle, \langle b_j \rangle, \langle v_j \rangle)]_{j=1}^{\mathsf{size}(L_r^b)}$: The list of buy orders submitted during this round for instrument r. Similarly, we insert $(\langle \mathsf{name}_0^s \rangle, \langle c_0 \rangle, \langle w_0 \rangle)$ in the list $L_r^s = [(\langle \mathsf{name}_k^s \rangle, \langle c_k \rangle, \langle w_k \rangle)]_{k=1}^{\mathsf{size}(L_r^s)}$: The list of sell orders submitted during this round for instrument r. See Figure 9 for a formal description of inserting a new buy order into the buy list. An identical algorithm can be used to insert a sell order.

Note that, for every new order entered to the auction, we add a dummy order $(\langle 0 \rangle, \langle \mathsf{MAX} \rangle, \langle 0 \rangle)$ to every list (unless the list is already of the maximal size), then we obliviously replace a dummy order by the new order in its corresponding list. Therefore, this hides how many orders are submitted to every instrument. For a formal analysis of the protocol see [7]. Also note that this sub-protocol leaks no information as no secret shared values are opened.

Given $L_r^b = [(\langle \mathsf{name}_j^b \rangle, \langle b_j \rangle, \langle v_j \rangle)]_{j=1}^{\mathsf{size}(L_r^b)}$ and $L_r^s = [(\langle \mathsf{name}_k^s \rangle, \langle c_k \rangle, \langle w_k \rangle)]_{k=1}^{\mathsf{size}(L_r^s)}$ of every instrument r in engine E_l, we process them to open any matched orders.

(1) For every instrument r in engine E_l.
  (I) For $j = 1, \ldots, \mathsf{size}(L_r^b)$
    (A) For $k = 1, \ldots, \mathsf{size}(L_r^s)$
      (i) $\langle f \rangle \leftarrow (\langle v_j \rangle \geq \langle c_k \rangle) \cdot (\langle w_k \rangle \geq \langle b_j \rangle)$.
      (ii) Open $\langle f \rangle$.
      (iii) If $f = 1$ then
        (a) $\langle t \rangle \leftarrow \langle v_j \rangle \geq \langle w_k \rangle$.
        (b) $\langle u \rangle \leftarrow \langle v_j \rangle + \langle t \rangle \cdot (\langle w_k \rangle - \langle v_j \rangle)$.
        (c) Print Trade and open $(\langle \mathsf{name}_j^b \rangle, \langle \mathsf{name}_k^s \rangle, \langle u \rangle)$.
        (d) $\langle v_j \rangle \leftarrow \langle v_j \rangle - u$.
        (e) $\langle w_k \rangle \leftarrow \langle w_k \rangle - u$.
          [Updating volumes.]
        (f) $\langle g \rangle \leftarrow (\langle b_j \rangle > \langle v_j \rangle)$.
        (g) $\langle b_j \rangle \leftarrow \langle b_j \rangle + \langle g \rangle \cdot (\mathsf{MAX} - \langle b_j \rangle)$.
        (h) $\langle g \rangle \leftarrow (\langle c_k \rangle > \langle w_k \rangle)$.
        (i) $\langle c_k \rangle \leftarrow \langle c_k \rangle + \langle g \rangle \cdot (\mathsf{MAX} - \langle c_k \rangle)$.
          [i.e. if volume is smaller than MES, we set MES to be MAX.]
  (II) $L_r^b \leftarrow \emptyset$, $L_r^s \leftarrow \emptyset$.
    [Delete all orders from the lists]

Figure 10. The Π_{unc} sub-protocol used for uncrossing the order book.

4.4 Order book uncrossing: the Π_{unc} sub-protocol

In the uncrossing phase, parties in every engine in $\{\mathsf{E}_1, \ldots, \mathsf{E}_L\}$ execute the sub-protocol in Figure 10 for all the corresponding instruments. Again this protocol was given and analyzed in [7] in the case of $R = 1$, the extension to general R is immediate. Note that, if a buy and sell order are matched, one of them will be completely matched, and the other one may be only partially matched. In our implementation, we do not remove completely matched orders during the auction, we instead replace them in an oblivious manner with dummy orders of the form $(\langle 0 \rangle, \langle \mathsf{MAX} \rangle, \langle 0 \rangle)$, therefore, we do not leak which order is only partially matched.

Similarly, if for some order the volume became smaller than the MES, we obliviously replace this order by a dummy order. This will prevent the leakage of orders that were partially matched and the remaining volume is still bigger than the MES.

We refer to [7] for the formal security analysis. In terms of our leakage model step 1.I.A.iii in Π_{unc}, if $f = 0$, the adversary only obtains the information that a buy and a sell orders do not match, this is allowed by event 2 from Figure 5 . If $f = 1$, one order will be completely matched and the other order will be partially matched. The name, MES, instrument, and the matched volume of both orders will be opened at the end of this period. This is captured by event 3 in Figure 5. At the end, all unmatched orders will be discarded. This will allow the adversary to bound the volumes of these orders from opened orders. This is captured by event 4 from Figure 5.

5 Runtimes for Turquoise Plato Uncross

The main question we need to ask is whether our protocol can deal with the specific requirements of a real life Dark Market operation such as Plato. Namely for the given number of instruments and expected throughput, can we evaluate the given algorithms within the given five second time intervals. Recall in time interval t we assume that the traders enter trades into the market, then the trades for time interval t are

uncrossed during time interval $t+1$; whilst the trades for time interval $t+1$ are enterred. As remarked earlier we assume all non-completed trades are wiped at the end of a time interval.

The main the parameter one one needs to determine is the size R, namely the number of instruments which are assigned to each engine. Recall a small R implies a more efficient implementation per engine, but we will require more engines. Whereas a large R implies less information is leaked during each time period, however the computational requirements may then mean the protocol is impractical.

To answer these questions we conducted experiments on computers running Ubuntu that have an identical configuration : namely i7-7700K CPUs and 32 GB of RAM. The ping time between those machines was .47 milliseconds and they were connected by a 10 GBit switch. We considered two settings for the number of Organizations n. Namely, $n = 2$ where we will use the SPDZ-based MPC protocol, and $n = 3$ where we will use the Shamir-based MPC protocol.

We examine each of the phases in turn. We start by noting that Π_{prep} is pre-processing and so can happen before each time period, perhaps during the night before trading begins.

Protocol Π_{inp}: The protocol is executed by the engine E_0, and the main cost is the cost of a single Paillier distributed decryption per order. For both of our access structures the Paillier decryption took $\mathsf{dd} = 0.118$ seconds, using a value of N of 2048 bits and an e value of one. This is the time to execute on a single core of each machine in the engine E_0.

As discussed earlier we expect, due to the need to clear the order book at the end of every time interval, that the traders will input at worst 2000 orders in every five second period. Assuming in a real installation E_0 is implemented using machines with 64-cores, processing the Paillier decryptions should take around $2000 \cdot 0.118/64 = 3.687$ seconds. However, this ideal latency might scale back due to network bottlenecks within engine E_0. This can easily be remedied in a real installation by having multiple versions of E_0. Thus in practice we do not see any problem in executing Π_{inp} in an real-life scenario.

Protocol Π_{ins}: The insertion phase is run by each engine E_i. Each engine is dealing (obliviously) with R different instruments. The engine takes each incoming order and inserts it into the buy or sell list for this instrument; without knowing which instrument it is. We measured the *online* time needed to insert incoming orders in the buy order book, for various values of R, and found the graphs given in Figure 11[10]. Note, the time needed to insert a new incoming order increases as the number of already processed orders increases.

In any given time interval, assuming a uniform distribution of completed orders over the day and given the numbers previously from the Plato auction, we expect $66307/6120 \approx 10$ orders to complete in every five second interval. In the worse case these are assigned to the same engine E_i in this time interval; we call this engine the 'hot' engine. Thus E_i has at least ten buy orders and ten sell orders coming in which will match.

Of the 2000 other orders coming in during the five second interval we expect them not to match. On average $2000/L$ of these to pass to a specific hot engine E_i. As they are unmatched we can assume that they all come in on the buy side; which will give us the worst performance given Figure 11. Thus we can assume the hot engine E_i has $M = 10 + 2000/L$ buy orders, and $N = 10$ sell orders coming in.

Selecting, for example, $R = 16$ we find that $L = U/R = 4500/16 \approx 281$. Which gives us $M = 17$ buy orders and $N = 10$ sell orders on the 'hot' engine. This will translate to a total run time of $0.8109 + 0.2915 = 1.1$ seconds to execute the protocol Π_{ins} on the buy and sell side together when $n = 2$, and $1.4229 + 0.5115 = 1.9$ seconds when $n = 3$. Note, that this protocol can be run in parallel with protocol Π_{inp}, i.e. as engine E_i receives orders when running Π_{inp} it then starts executing Π_{ins} to insert them.

Using a smaller value of R would result in a faster protocol, but more leakage of information. However, using a larger value of R such as $R = 32$, would result in run times which stretch our requirement guarantee to complete all phases in under five seconds.

[10] The *offline* time for MPC protocols can be much more, but this calculation can be performed during the previous night. For the SPDZ protocol for two parties the offline time is a factor 150-350 times slower than the online time; whereas for the three party Shamir based protocol it is only a factor of 3-8 times slower. This cost of the offline phase could impact the choice of $n = 2$ or $n = 3$ in practice. The offline phase is more expensive when $n = 2$, but the online phase is faster when $n = 2$. This disparity is why we give both implementations in this paper.

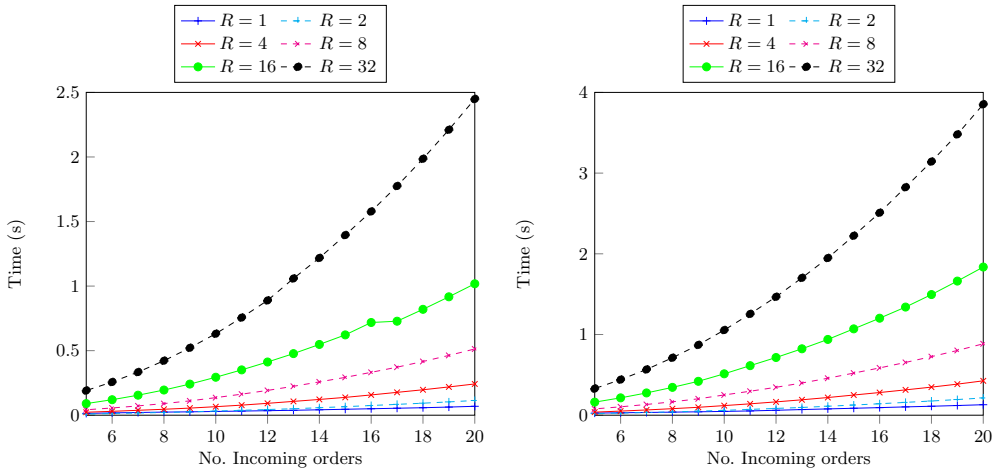

Fig. 11. Run times for Π_{ins} for buy (or sell) orders for two players (left) and three players (right)

Protocol Π_{unc}: The uncross phase, for the orders placed in time period t, occurs in time period $t+1$. However, we want to feed back the results relatively fast to enable traders to place new bids in the time period $t+1$, for example if their order was not matched. If we let $T(M,N)$ denote the time to perform the uncross phase for M buy and N sell orders, then we see that

$$T(M,N) \leq T(V,V) \text{ when } V = (M+N)/2.$$

The reason for this is that the algorithm for uncross essentially works as follows: For all $i \in \{1,\ldots,N\}$ and all $j \in \{1,\ldots,M\}$ compute and open a value. If the value is one (i.e. a trade is executed) you then do some operations which do not depend on M or N. Thus for a fixed number of trades the most expensive case is when $M \cdot N$ is maximal; which implies the above equation.

Thus we first investigate how $T(M,N)$ behaves when $M = N$ and we vary the number of completed orders v. For the case of $M = N = 10$ we obtain the graphs in Figure 12. The run time itself (theoretically) depends on the number of trades that are actually completed. The number of trades is bounded by $M+N-1$, as whenever a trade happens at least one order is completely matched (when $M, N \neq 0$). However, the affect of the number of trades on the run time is very minor, as the graphs show. The reason for this can be seen in Figure 10: Line 1.I.A.i needs to be executed $N \cdot M$ times, where as the steps in lines 1.I.A.iii only are executed when a trade is executed. The most expensive step is the comparison. So we have a 'base' number of $2 \cdot N \cdot M$ comparisons which happen irrespective of the number of orders executed, and then an extra three comparisons executed per order executed. Thus the run time is roughly $2 \cdot N \cdot M + 3 \cdot v$. As we have $N = M = 10$ and $v \leq 20$ in our experiment in Figure 12, we see that the number of trades makes a small different to the overall runtime. Note, the graphs contain a little variation due to small experimental errors.

We then looked at the growth of $T(M,N)$ when $M = N$ for a fixed number of trades, in particular ten trades. These are given in Figure 13. From this graph we see that our 'hot' engine, with $R = 16$, should require for the two party case $T(17,10) \leq T(14,14) \approx 2.5$ seconds to complete the uncrossing phase, and $T(17,10) \leq T(14,14) \approx 3.6$ seconds for the three party case.

Implication of choosing $R = 16$: Having selected $R = 16$ to obtain a reasonable level of leakage within a given time to perform the auction; we can now see how this would affect the resources needed to implement the Plato auction in this way. Recall from earlier that the Plato system offers 4500 different instruments, and hence we had $L = 281$ MPC engines. Adding one engine for the gateway, we have 282 engines in total. Thus each organization will need to provide 282 machines to implement the auction system.

Note, these figures could considerably be improved if more powerful machines were used to implement the engines $\mathsf{E}_1,\ldots,\mathsf{E}_L$. However, as with all MPC systems pushing more CPU power into the application may not give the expected performance benefit as one also needs to worry about bandwidth constraints.

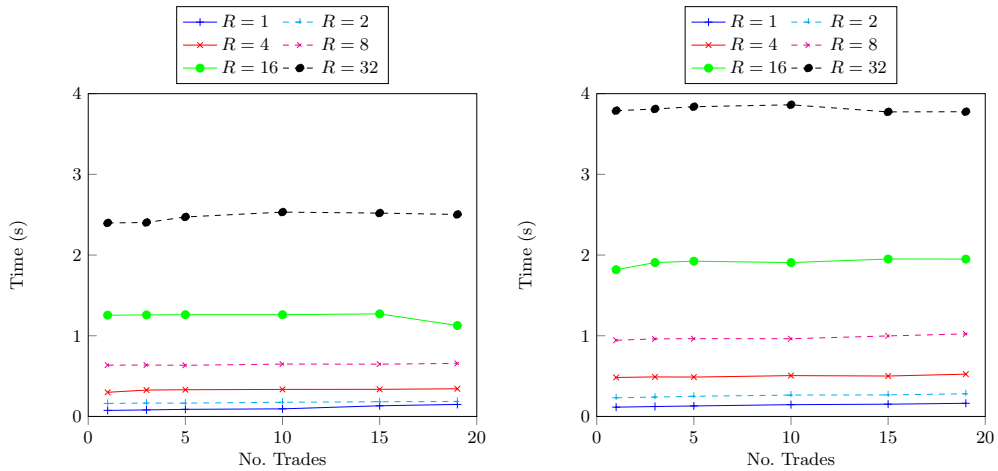

Fig. 12. Run times for Π_{unc} for two players (left) and three players (right) when $M = N = 10$,

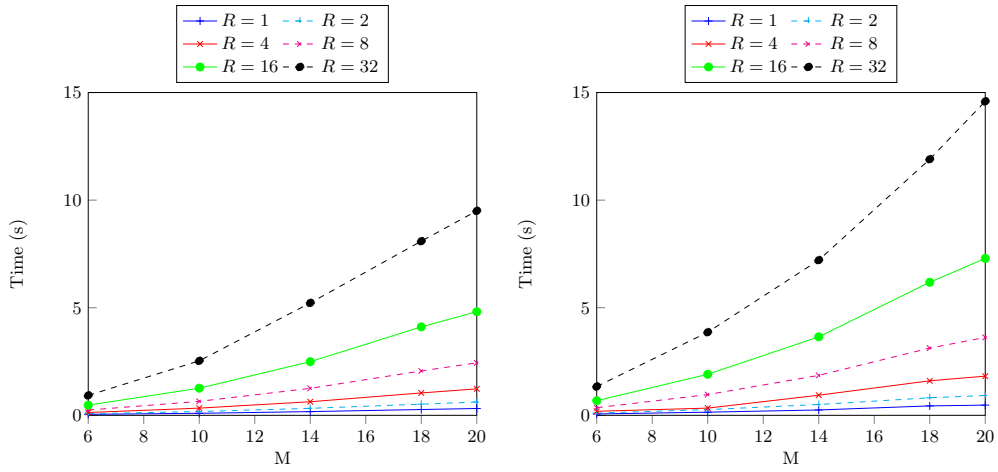

Fig. 13. Run times for Π_{unc} for two players (left) and three players (right) with respect to $M = N$, for 10 trades.

Thus one would also need to improve the throughput of the underlying network connecting the machines within each engine.

Acknowledgements

This work has been supported in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT, by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contract No. N66001-15-C-4070 and FA8750-19-C-0502, by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA) via Contract No. 2019-1902070006, by the FWO under an Odysseus project GOH9718N, and by CyberSecurity Research Flanders with reference number VR20192203.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any of the funders. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

1. Aly, A., Keller, M., Orsini, E., Rotaru, D., Scholl, P., Smart, N.P., Wood, T.: SCALE and MAMBA documentation (2018), https://homes.esat.kuleuven.be/~nsmart/SCALE/Documentation.pdf
2. Barnes, R.: Turquoise trading. The Parliamentary Review (Finance): Highlighting Best Practice, pp. 18–20 (2018), https://www.theparliamentaryreview.co.uk/organisations/turquoise-trading
3. Bogetoft, P., Christensen, D.L., Damgård, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J.D., Nielsen, J.B., Nielsen, K., Pagter, J., Schwartzbach, M.I., Toft, T.: Secure multiparty computation goes live. In: Dingledine, R., Golle, P. (eds.) FC 2009: 13th International Conference on Financial Cryptography and Data Security. Lecture Notes in Computer Science, vol. 5628, pp. 325–343. Springer, Heidelberg, Germany, Accra Beach, Barbados (Feb 23–26, 2009)
4. Bogetoft, P., Damgård, I., Jakobsen, T., Nielsen, K., Pagter, J., Toft, T.: A practical implementation of secure auctions based on multiparty integer computation. In: Di Crescenzo, G., Rubin, A. (eds.) FC 2006: 10th International Conference on Financial Cryptography and Data Security. Lecture Notes in Computer Science, vol. 4107, pp. 142–147. Springer, Heidelberg, Germany, Anguilla, British West Indies (Feb 27 – Mar 2, 2006)
5. Bouchaud, J.P., Bonart, J., Donier, J., Gould, M.: Trades, Quotes and Prices: Financial Markets Under the Microscope. Cambridge University Press, Cambridge, UK (2018), https://doi.org/10.1017/9781316659335
6. Bouchaud, J.P., Farmer, J.D., Lillo., F.: How markets slowly digest changes in supply and demand. In: Hens, T., Schenk-Hoppe, K. (eds.) Handbook of Financial Markets: Dynamics and Evolution, pp. 57–160. Elsevier: Academic Press, Amsterdam, NL (2009), https://doi.org/10.1016/B978-012374258-2.50006-3
7. Cartlidge, J., Smart, N.P., Talibi Alaoui, Y.: MPC joins the dark side. In: Galbraith, S.D., Russello, G., Susilo, W., Gollmann, D., Kirda, E., Liang, Z. (eds.) ASIACCS 19: 14th ACM Symposium on Information, Computer and Communications Security. pp. 148–159. ACM Press, Auckland, New Zealand (Jul 9–12, 2019)
8. Catrina, O., de Hoogh, S.: Improved primitives for secure multiparty integer computation. In: Garay, J.A., Prisco, R.D. (eds.) SCN 10: 7th International Conference on Security in Communication Networks. Lecture Notes in Computer Science, vol. 6280, pp. 182–199. Springer, Heidelberg, Germany, Amalfi, Italy (Sep 13–15, 2010)
9. Catrina, O., Saxena, A.: Secure computation with fixed-point numbers. In: Sion, R. (ed.) FC 2010: 14th International Conference on Financial Cryptography and Data Security. Lecture Notes in Computer Science, vol. 6052, pp. 35–50. Springer, Heidelberg, Germany, Tenerife, Canary Islands, Spain (Jan 25–28, 2010)
10. CME Globex: Marketing brochure. https://www.cmegroup.com/globex/files/globexbrochure.pdf (2018)
11. Comerton-Forde, C.: Shedding light on dark trading in Europe. Keynote Speech: CEPR-Imperial-Plato Inaugural Market Innovator (MI3) Conference (Jun 2017), https://cepr.org/sites/default/files/Comerton-Forde%2C%20Carole%20paper.pdf
12. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: Halevi, S., Rabin, T. (eds.) TCC 2006: 3rd Theory of Cryptography Conference. Lecture Notes in Computer Science, vol. 3876, pp. 285–304. Springer, Heidelberg, Germany, New York, NY, USA (Mar 4–7, 2006)

13. Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In: Kim, K. (ed.) PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography. Lecture Notes in Computer Science, vol. 1992, pp. 119–136. Springer, Heidelberg, Germany, Cheju Island, South Korea (Feb 13–15, 2001)
14. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. Lecture Notes in Computer Science, vol. 7417, pp. 643–662. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2012)
15. Farmer, J.D., Gerig, A., Lillo, F., Waelbroeck, H.: How efficiency shapes market impact. Quantitative Finance 13(11), 1743–1758 (2013), https://doi.org/10.1080/14697688.2013.848464
16. Harkavy, M., Tygar, J.D., Kikuchi, H.: Electronic auctions with private bids. In: 3rd USENIX Workshop on Electronic Commerce. pp. 61–73 (1998)
17. Hazay, C., Mikkelsen, G.L., Rabin, T., Toft, T.: Efficient RSA key generation and threshold Paillier in the two-party setting. In: Dunkelman, O. (ed.) Topics in Cryptology – CT-RSA 2012. Lecture Notes in Computer Science, vol. 7178, pp. 313–331. Springer, Heidelberg, Germany, San Francisco, CA, USA (Feb 27 – Mar 2, 2012)
18. Jutla, C.S.: Upending stock market structure using secure multi-party computation. Cryptology ePrint Archive, Report 2015/550 (2015), http://eprint.iacr.org/2015/550
19. Keller, M., Rotaru, D., Smart, N.P., Wood, T.: Reducing communication channels in MPC. In: Catalano, D., De Prisco, R. (eds.) SCN 18: 11th International Conference on Security in Communication Networks. Lecture Notes in Computer Science, vol. 11035, pp. 181–199. Springer, Heidelberg, Germany, Amalfi, Italy (Sep 5–7, 2018)
20. Lipmaa, H., Asokan, N., Niemi, V.: Secure Vickrey auctions without threshold trust. In: Blaze, M. (ed.) FC 2002: 6th International Conference on Financial Cryptography. Lecture Notes in Computer Science, vol. 2357, pp. 87–101. Springer, Heidelberg, Germany, Southampton, Bermuda (Mar 11–14, 2003)
21. LiquidMetrix: Guide to European Dark Pools - February 2017. Intelligent Financial Systems Limited. https://www.liquidmetrix.com (Feb 2017)
22. London Stock Exchange Group: LSEG Electronic Order Book Trading, Monthly Market Report, Feb. (Feb 2017), https://www.londonstockexchange.com/statistics/monthly-market-report/feb-17.pdf
23. London Stock Exchange Group: LSEG Electronic Order Book Trading, Monthly Market Report, Feb. (Feb 2020), https://www.londonstockexchange.com/statistics/monthly-market-report/lseg-monthly-market-report-february-2020.pdf
24. Massacci, F., Ngo, C.N., Nie, J., Venturi, D., Williams, J.: FuturesMEX: Secure, distributed futures market exchange. In: IEEE Symposium on Security and Privacy (SP). pp. 335–353. IEEE Computer Society, San Francisco, CA (May 2018), https://doi.org/10.1109/SP.2018.00028
25. Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: EC '99: Proceedings of the 1st ACM conference on Electronic Commerce. pp. 129–139 (Nov 1999), https://doi.org/10.1145/336992.337028
26. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) Advances in Cryptology – EUROCRYPT'99. Lecture Notes in Computer Science, vol. 1592, pp. 223–238. Springer, Heidelberg, Germany, Prague, Czech Republic (May 2–6, 1999)
27. Parkes, D.C., Rabin, M.O., Shieber, S.M., Thorpe, C.: Practical secrecy-preserving, verifiably correct and trustworthy auctions. Electronic Commerce Research and Applications 7(3), 294–312 (2008)
28. Parkes, D.C., Thorpe, C., Li, W.: Achieving trust without disclosure: Dark pools and a role for secrecy-preserving verification. In: Third Conference on Auctions, Market Mechanisms and Their Applications (AMMA'15). pp. 38–48. Chicago, IL (Aug 2015), http://nrs.harvard.edu/urn-3:HUL.InstRepos:32785051
29. Partisia: Secure order matching. Webpage (2018), https://partisia.com/order-matching
30. Petrescu, M., Wedow, M.: Dark pools in European equity markets: emergence, competition and implications. European Central Bank: Occasional Paper Series, No. 193 (Jul 2017), https://www.ecb.europa.eu/pub/pdf/scpops/ecb.op193.en.pdf
31. Rabin, M.O., Mansour, Y., Muthukrishnan, S., Yung, M.: Strictly-black-box zero-knowledge and efficient validation of financial transactions. In: Czumaj, A., Mehlhorn, K., Pitts, A.M., Wattenhofer, R. (eds.) ICALP 2012: 39th International Colloquium on Automata, Languages and Programming, Part I. Lecture Notes in Computer Science, vol. 7391, pp. 738–749. Springer, Heidelberg, Germany, Warwick, UK (Jul 9–13, 2012)
32. Rabin, M.O., Servedio, R.A., Thorpe, C.: Highly efficient secrecy-preserving proofs of correctness of computations and applications. In: 22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007). pp. 63–76. Wroclaw, Poland (Jul 2007), https://doi.org/10.1109/LICS.2007.24
33. Thorpe, C., Parkes, D.C.: Cryptographic securities exchanges. In: Dietrich, S., Dhamija, R. (eds.) FC 2007: 11th International Conference on Financial Cryptography and Data Security. Lecture Notes in Computer Science, vol. 4886, pp. 163–178. Springer, Heidelberg, Germany, Scarborough, Trinidad and Tobago (Feb 12–16, 2007)

34. Thorpe, C., Parkes, D.C.: Cryptographic combinatorial securities exchanges. In: Dingledine, R., Golle, P. (eds.) FC 2009: 13th International Conference on Financial Cryptography and Data Security. Lecture Notes in Computer Science, vol. 5628, pp. 285–304. Springer, Heidelberg, Germany, Accra Beach, Barbados (Feb 23–26, 2009)
35. Thorpe, C., Willis, S.R.: Cryptographic rule-based trading - (short paper). In: Keromytis, A.D. (ed.) FC 2012: 16th International Conference on Financial Cryptography and Data Security. Lecture Notes in Computer Science, vol. 7397, pp. 65–72. Springer, Heidelberg, Germany, Kralendijk, Bonaire (Feb 27 – Mar 2, 2012)
36. United States of America before the Securities and Exchange Commission: In the Matter of Pipeline Trading Systems LLC, et al., Securities Exchange Act of 1934 Release No. 65609. https://www.sec.gov/litigation/admin/2011/33-9271.pdf (24 Oct 2011)
37. United States of America before the Securities and Exchange Commission: In the Matter of eBX, LLC Securities Exchange Act of 1934 Release No. 67979. https://www.sec.gov/litigation/admin/2012/34-67969.pdf (3 Oct 2012)
38. United States of America before the Securities and Exchange Commission: In the Matter of LavaFlow, Inc. Securities Exchange Act of 1934 Release No. 72673. https://www.sec.gov/litigation/admin/2014/34-72673.pdf (25 Jul 2014)
39. United States of America before the Securities and Exchange Commission: In the Matter of Liquidnet, Inc., Securities Exchange Act of 1934 Release No. 72339. https://www.sec.gov/litigation/admin/2014/33-9596.pdf (6 Jun 2014)
40. United States of America before the Securities and Exchange Commission: In the Matter of ITG Inc. and Alternet Securities, Inc., Securities Exchange Act of 1934 Release No. 75672. https://www.sec.gov/litigation/admin/2015/33-9887.pdf (12 Aug 2015)
41. United States of America before the Securities and Exchange Commission: In the Matter of UBS Securities LLC, Securities Exchange Act of 1934 Release No. 74060. https://www.sec.gov/litigation/admin/2015/33-9697.pdf (15 Jan 2015)
42. United States of America before the Securities and Exchange Commission: In the Matter of Barclays Capital Inc., Securities Exchange Act of 1934 Release No. 77001. https://www.sec.gov/litigation/admin/2016/33-10010.pdf (31 Jan 2016)
43. United States of America before the Securities and Exchange Commission: In the Matter of Credit Suisse Securities (USA) LLC, Securities Exchange Act of 1934 Release No. 77002. https://www.sec.gov/litigation/admin/2016/33-10013.pdf (31 Jan 2016)
44. United States of America before the Securities and Exchange Commission: In the Matter of Credit Suisse Securities (USA) LLC, Securities Exchange Act of 1934 Release No. 77003. https://www.sec.gov/litigation/admin/2016/33-10014.pdf (31 Jan 2016)
45. United States of America before the Securities and Exchange Commission: In the matter of Deutsche Bank Securities Inc. Securities Exchange Act of 1934 Release No. 79576. https://www.sec.gov/litigation/admin/2016/33-10272.pdf (16 Dec 2016)
46. United States of America before the Securities and Exchange Commission: In the Matter of Citigroup Global Markets, Inc. and Citi Order Routing and Execution, LLC Securities Exchange Act of 1934 Release No. 84124. https://www.sec.gov/litigation/admin/2018/33-10545.pdf (14 Sep 2018)
47. United States of America before the Securities and Exchange Commission: In the Matter of ITG Inc. and Alternet Securities, Inc., Securities Exchange Act of 1934 Release No. 84548. https://www.sec.gov/litigation/admin/2018/33-10572.pdf (7 Nov 2018)
48. United States of America before the Securities and Exchange Commission: In the Matter of Merrill Lynch, Pierce, Fenner & Smith Incorporated Securities Exchange Act of 1934 Release No. 83462. https://www.sec.gov/litigation/admin/2018/33-10507.pdf (19 Jun 2018)
49. United States Securities and Exchange Commission: SEC institutes enforcement action against 20 former New York Stock Exchange specialists alleging pervasive course of fraudulent trading. Press Release, https://www.sec.gov/news/press/2005-54.htm (12 April 2005)
50. Varian, H.R.: Economic mechanism design for computerized agents. In: 1st USENIX Workshop on Electronic Commerce. p. 9 (Jul 1995)
51. Zheng, W., Popa, R.A., Gonzalez, J.E., Stoica, I.: Helen: Maliciously secure coopetitive learning for linear models (2019)