

Kicking-the-Bucket: Fast Privacy-Preserving Trading Using Buckets

Mariana Botelho da Gama¹ , John Cartlidge² , Antigoni Polychroniadou³,
Nigel P. Smart¹ , and Younes Talibi Alaoui¹ 

¹ imec-COSIC, KU Leuven, Leuven, Belgium.

² University of Bristol, Bristol, UK.

³ J.P. Morgan AI Research, New York, USA.

`mariana.botelhodagama@kuleuven.be`

`john.cartlidge@bristol.ac.uk`,

`antigoni.polychroniadou@jpmorgan.com`,

`nigel.smart@kuleuven.be`,

`younes.talibialaoui@kuleuven.be`

Abstract. We examine bucket-based and volume-based algorithms for privacy-preserving asset trading in a financial dark pool. Our bucket-based algorithm places orders in quantised buckets, whereas the volume-based algorithm allows any volume size but requires more complex validation mechanisms. In all cases, we conclude that these algorithms are highly efficient and offer a practical solution to the commercial problem of preserving privacy of order information in a dark pool trading venue.

1 Introduction

The majority of major stock exchanges are now electronic order-driven markets, where investors submit orders to buy or sell a quantity of stock at a particular price. Orders that are not immediately filled (i.e., those that do not immediately result in a trade) are publicly displayed in a limit order book (LOB), which presents a price-ordered view of the instantaneous demand and supply in the market. With each order in the book acting as an advertisement of an investor’s willingness to commit to a particular trade, the LOB is an efficient method for finding counterparties with whom to trade. However, sometimes it is beneficial for an investor to hide their trading intention. In particular, when attempting to trade in large volume (i.e., when wanting to buy or sell a large quantity of stock), exposing one’s intention will likely lead to adverse price movement as the information contained in the large order causes other investors to re-evaluate market price. This effect is known as *price impact*, or *market impact*, and it can be extremely costly to a large-volume investor. To reduce impact, an investor will often “salami slice” one large order into multiple smaller orders and drip feed these into the market slowly over time. So common is this approach that many exchanges offer an “iceberg” order type that automates a similar process. When an iceberg order is submitted, only a small proportion of the full order

volume (the “tip of the iceberg”) is displayed in the order book at any given time, while the bulk of the remaining order remains hidden (“submerged” out of view). However, while the use of icebergs to disguise order volumes can help limit the effects of market impact, icebergs are exposed to the risk that other investors will anticipate the hidden iceberg volume from information leaking from the visible tip.

To counter this, some trading venues hide all pre-trade order information. Commonly referred to as “dark pools” to contrast with the “lit” order books of an exchange, these trading venues ensure that all order information is non-displayed. As other investors have no access to the information in a dark pool, so market impact can be significantly reduced, or avoided entirely. Hidden away from viewing eyes, orders in a dark pool tend to take longer to fill than equivalent orders submitted to an exchange. However, in most cases, the potential savings available to large volume institutional investors will significantly outweigh the desire for trading urgency. That is, volume investors are usually prepared to wait as long as the final deal they make is fair. As a result, dark pool trading has risen in popularity, with more than 15% of all US equities, and more than 8% of all EU equities, trading on dark pools in 2017 [20]. Yet, dark pools persistently suffer from negative reputation as some operators have taken advantage of their privileged access to the non-displayed orders in their systems. Indeed, between 2011-2018, dark pool operators paid more than \$217 million to the SEC in penalty settlements for misusing customer order information or operating the dark pool in a way that disadvantaged their customers [8]. In the shadowy world of the dark pool, it is easier for a market manipulator to hide. As such, it is perhaps unsurprising that many investors have a fear of the dark.

There is now a strong commercial drive from financial institutions, such as JPMorgan [2,4], to offer investors a secure dark pool trading venue. To be commercially viable, such a platform would require guaranteed order privacy, the ability to handle imbalanced order-flows from around 1000 active investors or more, and periodic order matching at regular intervals, where execution price is determined by some reference value such as the mid-point of the National Best Bid and Offer (NBBO). To address this problem, we consider algorithms for implementing fast privacy-preserving trading protocols such that *nobody*, not even the system operator, can access (and therefore misuse) order information. These algorithms are designed to stop fraudulent behaviour but can also benefit honest dark pool operators as they offer customers a guarantee that does not rely solely on trust. Using multi-party computation (MPC) based protocols, the investors secret share their orders across several entities who emulate the dark pool operator. As long as these entities do not collude, nobody can access the system information. In [7], Cartlidge et al. used MPC to present a proof-of-concept implementation of three dark pool trading mechanisms, showing that “volume matching” can be viably executed in a privacy-preserving manner with order throughput similar to that required by a real world dark pool trading venue. Further, in [8], Cartlidge et al. demonstrated how to use MPC to run multiple auctions in parallel, offering simultaneous trading across thousands of

stocks such that the identity of the stock being traded is also hidden and secure. The throughput per MPC engine is however significantly lower than that of the volume matching from [7] due to the use of a more complex matching algorithm.

In this paper, we build upon the work from [7] and introduce two matching algorithms using MPC: (i) “bucket match”, and (ii) a “volume match” with a more efficient clearing phase. For both mechanisms, we trade one financial instrument (i.e., one stock) such that orders are matched according to volume only and price is determined by some external reference value. In *bucket match*, buy and sell orders placed in the same auction must have the same volume, which is determined by the bucket size. To hide the volume that each investor wishes to buy or sell (or the fact that the investor is even interested in trading a given stock), orders with zero volume may also be submitted. Multiple auctions with different sized buckets can be run in parallel, after which unfilled orders remaining in the different bucket lists may be matched against each other. In *volume match*, there is no bucketing and investors may submit orders of any volume they wish (including zero volume orders), similar to the volume trading algorithm presented in [7]. However, we extend the previous volume trading protocol by simplifying the clearing phase. Namely, all the orders in the direction with less total volume are opened simultaneously, instead of being checked one by one before opening. We also increase privacy by no longer revealing the direction of an order (i.e., it is not possible to tell whether the order is to buy or to sell). Both algorithms were implemented with the **Scale-Mamba** Framework [1] using Shamir Secret Sharing based MPC, which provides security with abort against active adversaries for an honest majority. We empirically evaluate the case where three MPC parties emulate the dark pool operator.

Related work: Work in secure privacy-preserving auction mechanisms can be roughly categorised into two broad categories: those involving a public bulletin board (e.g., a blockchain), for verifying auction correctness, or as a secure communication channel between parties; and those where MPC is used to implement an auction or dark pool using a set of operators. We briefly review these, below.

In 2021, Ngo et al. [18] introduced a framework for secure financial trading that uses a public bulletin board (e.g., a permissionless blockchain) hidden behind an anonymous network (e.g., Tor) for privacy-preserving communication between investors. The authors introduce witness-key-agreement (WKA), a cryptographic scheme that allows counterparties to securely agree on a secret using publicly committed information that meets some desired relation. Parties negotiate securely by publishing partial zk-SNARK proofs on the public bulletin board to reach a trade agreement. This process emulates a secure distributed over-the-counter (OTC) dark pool, such that trade price and volume is negotiated directly between counterparty pairs. Therefore, there is no need for an auctioneer (or dark pool operator) to match orders. The runtimes for each protocol step are below 15 seconds, the average block generation time in Ethereum.

Also in 2021, Galal and Youssef [14] introduced a publicly verifiable and secrecy preserving periodic auction protocol that makes use of a smart contract

deployed on the Ethereum blockchain. Investors first commit to their orders in the smart contract using Bulletproofs to generate an aggregate range proof. The auction (or dark pool) operator then privately receives orders from investors, each encrypted with the operator’s public key. The operator decrypts orders and calculates clearing price and volume for the auction, before publishing a proof of correctness to the smart contract. The smart contract serves as a secure bulletin board and enables public verification of the submitted zero-knowledge proofs. Constantinides and Cartledge [11] introduced a similar smart contract for validating the honesty of the operator. Again, orders are submitted in encrypted form to the smart contract, the operator matches orders off-chain in unencrypted form, and the result of the auction is published to the smart contract. This enables investors to verify whether their own orders were handled correctly, while preserving the privacy of all unexecuted orders. In addition, since the smart contract logic only handles order flow and is independent of the matching logic, the operator can use any double auction matching rules without altering the smart contract.

In 2019, Bag et al. [3] presented a protocol to perform a first-price sealed-bid auction without a central “auctioneer” entity. Decentralised bidders engage in the protocol to determine the winning bidder with the highest bid. The protocol consists of a committing phase, where every bidder sends an order commitment to a public bulletin board, then a second phase where bidders jointly compute the highest bid without leaking the other bids. This computation is performed using a modified version of the Anonymous Veto network protocol proposed in [15]. Following this, the winning bidder can come forward to prove they had the highest bid, and everyone else can verify their claim. The computation and communication have a linear complexity on the bit length of the bids throughout all phases; and the verification phase has linear complexity on the number of parties. While this protocol has efficient time complexity, it is not obvious how it could be extended to a double auction, where buyers and sellers are matched.

In 2006, Parkes et al. [19] proposed a secure protocol to perform a sealed-bid auction using homomorphic encryption, where only one auctioneer carries out the auction. The auctioneer publishes his/her public key, and the auction is performed by bidders committing to their bids and then sending the commitments to the auctioneer. Bidders then submit their bids to the auctioneer who verifies first if the bids are consistent with the commitments, before running the auction on clear bid data. Subsequently, the auctioneer posts the winner of the auction along with proofs that the computation was performed according to the specified protocol. One thing to note here is that, while the protocol prevents the auctioneer from cheating, the unmet orders are revealed to the public and so the trading intentions of these bidders are leaked. This work was extended in 2007 [22] to cope with continuous double auctions (where orders to buy and sell can be submitted and matched at any time), by checking whether orders can be matched with existing orders as soon as they are entered. In 2009 [23], protocols were further extended to enable trading in baskets of securities; and in 2012 [24], rule-based trading was introduced. The works of [2, 4] offer a pri-

vacy preserving double auction mechanism and a volume matching mechanism, respectively, without any leakage based on fully homomorphic encryption using a single operator.

In 2006, seminal work by Bogetoft et al. [6] introduced an MPC protocol to perform a one-shot double auction among a set of auctioneers, such that investors secret share their orders with the auctioneers and orders are obliviously addressed using Shamir Secret Sharing with passive security. This work was deployed in 2008 [5], to secure the Danish sugar beet auction between farmers and the company Danisco, the only sugar beet processor in Denmark. In this auction, farmers provide the amount of sugar beet they are willing to sell for every potential price. Similarly, buyers provide the amounts they are willing to buy for every potential price. The clearance price is then calculated as the point that supply equals demand. The auction was successfully run by three auctioneers, namely, Danisco; DKS, the sugar beet growers' association; and SIMAP, the research team. Since then, the auction has taken place every year.

In 2015, Jutla [16] introduced an MPC based protocol for periodic double auctions, with five entities playing the role of the auctioneers; four brokers and one regulating authority. Investors first submit orders during an open-auction period. Orders are then cleared at a single price and unmet orders remain in the auction for the following rounds. Making the assumption that the strategies of investors do not have to be kept secret, Jutla suggests that a passively secure protocol is sufficient, as long as the auctioneers wait a reasonable amount of time (e.g., one month) before releasing transcripts of the computations for audit. Jutla does not report an implementation of the protocol, but claims that the MPC technology at that time (in 2015) would be capable of executing the day's first auction in 30 minutes and subsequent auctions every 15 minutes; with additional 5 minute breaks between auctions, to allow bidders to digest results.

Cartlidge et al. [7] proposed an MPC based protocol for performing auctions in dark pools, where a set of $l = 2$ or $l = 3$ auctioneers can emulate the dark pool operator. Cartlidge et al. considered three common matching mechanisms: (i) a continuous double auction, where buyers and sellers can submit orders at any time and a limit order book is used for matching; (ii) a periodic double auction, where the clearance price is determined by maximising quantity matched; and (iii) a volume matching algorithm, which simply matches buy and sell volume and price is taken from some reference exchange. Investors submit orders by secret sharing them among the auctioneers, thus auctioneers learn nothing about the orders, except for the direction of the order (i.e., whether the order is to sell or to buy), as this information is sent to auctioneers on clear data. The protocols proposed are actively secure with abort and were implemented using the Scale-Mamba framework [1], with $l = 2$ using the SPDZ protocol [13], and $l = 3$ using Shamir Secret Sharing based MPC. The runtimes reported show that the volume matching is the fastest algorithm, capable of processing a throughput of around 1000 orders per second for the case where $l = 3$, and around 2000 orders per second for the case where $l = 2$. The throughput for the other two algorithms was found to be insufficient for real-world applicability. Namely, the

continuous double auction algorithm which can be commonly found in lit markets was considered unsuitable for evaluation in an MPC system for dark pools.

In 2020, Cartlidge et al. [8] introduced a follow-up work to secure a system inspired by the London Stock Exchange Group’s Turquoise Plato Uncross algorithm (TPU for short). The TPU manages dark pool trading across 4500 different instruments, thus Cartlidge et al. considered running the auction on multiple engines, where each engine addresses a sub-set of instruments, so as to cope with the amount of orders that TPU receives in real life. The challenge consisted of distributing instruments across engines without leaking the instruments that each engine is dealing with, as this would reveal information about the trading activity of each instrument. Cartlidge et al. [8] concluded that assigning 16 instruments to each engine (and thus 281 engines are needed)¹ would cope with the real world throughput that TPU needs to address. The worst case throughput for each of these engines is of around 8 orders per second for $l = 2$, and around 5 orders per second for $l = 3$. Note that, as mentioned before, this is indeed significantly lower than the throughput of the volume matching in [7] presented above.

2 Our Proposed Auction Algorithms

Both of the proposed algorithms follow the scheduled cross methodology, where the matching occurs at fixed points in time and is based on volume only. Trade price is determined by reference to an external lit market value, thus the orders for both algorithms do not contain price information. Each order contains the identity of the investor who submitted it, the direction of the order (i.e., whether it is a buy or a sell order), and, in the volume match case, the volume to be traded. A separate auction is run for each tradable instrument (i.e., each stock). The output of each auction consists of a list of all filled orders (although some orders might be partially filled, as will be explained at the end of this section).

A textual description of the bucket match and the volume match in the clear can be found below. Our goal is to adapt these algorithms to prevent revealing information besides what is absolutely essential for the trades to take place. As such, any hint about the volume or direction in which a given investor wishes to trade will be considered as an information leakage (as long as the orders are still waiting to be filled). In Section 3, we present secure versions of the algorithms and in Section 4 we analyse the corresponding leakage.

Bucket match: We consider an auction in which orders can only be executed in a given number y of bucket sizes. For each $j \in [1, \dots, y]$ we define the fixed bucket size as unit^j , and the algorithm maintains a list L^j of the orders with list L^j containing only buy and sell orders of size unit^j . Order i in list j is of the form $[\text{id}_i^j, \text{direction}_i^j]$, where id_i^j is the identity of the investor, and direction_i^j

¹ Plus one engine that serves as an entry gateway for orders; therefore a total of 282 engines required.

is the direction of the order, i.e., whether the order is a sell ($\text{direction}_i^j = 1$) or buy order ($\text{direction}_i^j = 0$). Therefore, if an investor wishes, for instance, to sell a volume v , the investor has to submit g^j *distinct* orders to list j , where $g^j \geq 0$, such that $v = \sum_{j=1}^{j=y} g^j \cdot \text{unit}^j$, with the direction of each of these orders indicating that they consist of sell orders, i.e., $\text{direction}_i^j = 1$ for all orders.

Orders are placed in their lists in order of arrival, and orders that arrived first will be matched first. The clearing of all orders is then run at periodic intervals. Unless the number of sell orders is identical to the number of buy orders in a given list, there will be leftover unmatched orders after this same list is cleared. After every list is cleared, we can check the direction of the leftover orders from each of them. If there are leftover orders with different directions (e.g., leftovers from L^1 are buy orders, and leftovers from L^2 are sell orders), then there will be another clearing period where the leftover orders of all lists are matched among each other. Recall that orders from different lists have different volume and hence we must now take into consideration their unit volume, in addition to their direction.

For ease of exposition, we will consider in our work only the cases of $y = 1$ and $y = 2$; i.e., we will either have one bucket size or two bucket sizes. As a shorthand, we will refer to these as *bucket-1* and *bucket-2*, respectively; *bucket-z* will refer to the general case of multiple lists, i.e., where $y > 1$.

Volume match: In this algorithm, the auction runs over one list L that contains orders of different sizes. Order i is thus of the form $[\text{id}_i, \text{direction}_i, \text{volume}_i]$, where id_i is the identity of the investor, direction_i is the direction of the order, and volume_i is the volume of the order. Note that, in this situation, if one wishes to trade a volume v , it is enough to submit a single order of volume v (though it is also possible to split the volume into multiple smaller orders). The procedure is then similar to the bucket match case, except that here we consider only one list and therefore the cross-list matching does not take place.

Table 1: Intuitive comparison of bucket match with 1 list, multiple lists, and volume match.

Algorithm	Total Orders	Additional Computation	Leakage Potential	Loss in Volume Submitted
Bucket-1	Most	-	Low	Low
Bucket-z	Medium	Cross-list matching	Cross-list match leakage	Low
Volume	Least	Input correctness check	Lowest	No loss

Intuitive Comparison: Bucket-1 will tend to receive more orders than bucket-z or volume match, as multiple orders must be submitted for trading large volumes. Therefore, as more orders need to be processed, runtimes for bucket-1 are

likely to be longer. Bucket-z solves this problem by introducing multiple bucket sizes, thus allowing orders of different volumes. However, it will usually require an additional cross-list matching period to find all possible matches between different bucket sizes. Moreover, we would like the volume of unmatched orders to remain secret, which might not be possible when matching orders of different volumes. If an order can only be partially matched, the leftover volume will become public. Therefore, bucket-z has potential for greater leakage than bucket-1. Regarding the total submitted volume, note that one cannot always submit the exact volume they wish, since all orders must fit the predefined bucket size(s). Thus, investors might need to submit a lower total volume than intended.

Volume match allows orders to be submitted with any volume, so there is additional uncertainty about the volume of unopened orders. There is also no need to implement an additional cross-list matching period, therefore preventing the leakage of leftover volume of partially matched orders. However, checking the correctness of input orders will be slower than in bucket-1 and bucket-z, with the runtime growing linearly with the number of input bits representing the volume.

3 Secure implementations of the algorithms

To ensure privacy of the orders we implement the above auction algorithms on top of a generic multi-party computation (MPC) system. For an overview of the requirements and some notation we will use see Appendix A.

3.1 Setup

The setup consists of a number of servers $S = \{S_1, \dots, S_l\}$ emulating the auctioneer, where the orders entering the auction will be secret shared among these servers.

3.2 Bucket match:

We aim to hide as much about the intention of the investors as possible, especially for unmet orders. Thus we allow investors to enter ‘dummy’ orders, i.e., orders which are neither buy or sell. We will discuss later the precise number of dummy orders which should be entered, and how this number affects the privacy and performance of the auction. Note that investors can submit dummy orders to stocks they do not wish to trade, thus hiding their trading activity in each stock.

For $i = 1, \dots, n$, each order i will be of the form $\text{ord}_i^j = [\langle \text{id}_i^j \rangle, \langle b_i^j \rangle, \langle s_i^j \rangle]$, where b_i^j and s_i^j are bits indicating the direction of the order, that is, a sell order will have $b_i^j = 0$, $s_i^j = 1$ and a buy order will have $b_i^j = 1$, $s_i^j = 0$. To allow dummy orders, orders can also contain $b_i^j = 0$ and $s_i^j = 0$. Every order for which $(b_i^j, s_i^j) \notin \{(0, 0), (0, 1), (1, 0)\}$ will be rejected. Each list j will contain n^j orders, among which m^j are dummy. For instance, if an investor j wants to sell a volume V , they need to enter the orders $\{\text{ord}_1^1, \dots, \text{ord}_{g^1}^1, \dots, \text{ord}_1^y, \dots, \text{ord}_{g^y}^y\}$ such that $V = \sum_{j=1}^y \sum_{i=1}^{n^j} (s_i^j - b_i^j) \cdot \text{unit}^j$.

To ensure that the conditional operation $\langle c \rangle > 0$ can be executed we need to ensure that $c \in [-2^{k-1}, \dots, 2^{k-1}]$. For the case of one list we simply need to ensure that the total number of orders n is less than 2^{k-1} . For the case of more than one list we need to ensure that $n \cdot \text{unit}^y < 2^{k-1}$.

Bucket-1 match: For ease of exposition we first examine the case when we have only one bucket size, i.e. $y = 1$. The formal description of the algorithm is given in Figure 3 in Appendix B. We distinguish 3 phases:

1. The input phase, where orders are entered into the auction and a check is run to discard invalid orders. In the input orders for this algorithm, the buy and sell entries b and s must be bits. Additionally, at least one of these two entries must be zero. To verify this, we draw three numbers $\alpha, \beta, \gamma \in \mathbb{F}_p$ at random and calculate

$$\langle t \rangle = \alpha \cdot (\langle b \rangle \cdot \langle b \rangle - \langle b \rangle) + \beta \cdot (\langle s \rangle \cdot \langle s \rangle - \langle s \rangle) + \gamma \cdot (\langle b \rangle \cdot \langle s \rangle).$$

Afterwards, we open $\langle t \rangle$ and check whether $t = 0$. The first two terms are zero only if b and s are bits, except with probability $1/p$. The last term is zero only if either $b = 0$ or $s = 0$, except with probability $1/p$. If more than one term is different from zero, their sum will be zero with probability $1/p$.

2. The clearing phase one, where we open the orders in the direction that will be completely cleared. First, we need to check which list has largest total volume. To do so, we first calculate

$$\langle c \rangle \leftarrow \sum_{i=1}^n \langle b_i \rangle - \langle s_i \rangle.$$

Then, we perform the comparison $\langle c \rangle > 0$ and open the output. If c is greater than zero, there are more buy orders than sell orders and so we open the $\langle s_i \rangle$ share of every order i . Otherwise, we open the $\langle b_i \rangle$ shares. The $\langle \text{id} \rangle$ of non-dummy orders is also opened. Opening $\langle s_i \rangle$ (or $\langle b_i \rangle$) will reveal whether order i is a sell order (or buy order, respectively). However, because of the existence of dummy orders, revealing that order i is not a sell order (or buy order, respectively) does not imply that it is an order in the opposite direction. We are then left with a mix of dummy and non-dummy orders, without knowing which are which.

3. The clearing phase two, where we open the orders in the direction that will be only partially cleared. The orders are opened one by one, and the $\langle \text{id} \rangle$ of non-dummy orders is also opened. For each opened order, we check whether the opposite direction has been completely cleared. When that is the case, we exit the algorithm.

Bucket-2 match: We now examine the case with two bucket sizes, i.e., $y = 2$. The size of the first bucket is unit^1 and the size of the second bucket is unit^2 . We present in Figure 4 (in Appendix B) the formal description of the algorithm. We distinguish the following phases of the algorithm:

1. The input phase, the clearing phase one and the clearing phase two are exactly as in the bucket match with one bucket size. Each of the two lists is cleared individually, and then we check whether the leftover orders from both lists have different directions. If so, we can proceed to matching orders from different lists. If all the orders have the same direction, we exit the algorithm. Note that, while we know the direction of the leftover orders, we do not know which of them might be dummy orders.
2. The clearing phase three, where we open the orders in the direction that will be completely cleared. First, we need to check which direction has largest total volume. To do so, we first calculate

$$\langle c \rangle \leftarrow \sum_{i=1}^{n^2} \langle \text{dir}_i^2 \rangle \cdot \text{unit}^2 - \sum_{i=1}^{n^1} \langle \text{dir}_i^1 \rangle \cdot \text{unit}^1,$$

where dir^j is b^j if the leftovers from list j are buy orders, or s^j if the leftovers from list j are sell orders. Then, we perform the comparison $\langle c \rangle > 0$ and open the output. If c is greater than zero, there is more volume in direction dir^2 and so we open all the $\langle \text{dir}^1 \rangle$ shares. Otherwise, we open the $\langle \text{dir}^2 \rangle$ shares. The $\langle \text{id} \rangle$ of non-dummy orders is also opened.

3. The clearing phase four, where we open the orders in the direction that will be only partially cleared. The orders are opened one by one, and the $\langle \text{id} \rangle$ of non-dummy orders is also opened. For each opened order, we check whether the opposite direction has been completely cleared. When that is the case, we exit the algorithm.

Note, the last opened order from the clearing phase four will not be necessarily completely matched. The unmatched volume from this last order will therefore be leaked. This source of leakage is further discussed in Section 4

3.3 Volume match:

Similarly to the bucket match, we will hide here the direction of orders and we will allow dummy orders. Each order i will be of the form $\text{ord}_i = [\langle \text{id}_i \rangle, \langle v_i \rangle, \langle \text{dir}_i^b \rangle, \langle \text{dir}_i^s \rangle]$, where v_i is the volume of the order, $\text{dir}_i^b = 0$ if ord_i is a sell order, $\text{dir}_i^s = 0$ if ord_i is a buy order, and $\text{dir}_i^b = \text{dir}_i^s = 0$ if ord_i is a dummy order. The list of orders from all the investors will contain n orders, m of which are dummy orders. If an investor wants to sell volume V , they need to enter orders $\text{ord}_1, \dots, \text{ord}_g$ such that $V = \sum_{i=1}^g (v_i \cdot (\text{dir}_i^s - \text{dir}_i^b))$

The formal description of this algorithm is presented (in Appendix B) in Figure 5. Again we distinguish 3 phases of the algorithm:

1. The input phase, where orders are entered into the auction and a check is run to discard invalid orders. To ensure investors enter values v_i that are valid non-negative numbers less than some bound B (which we assume is an exact power of two, i.e. $B = 2^\ell$), they enter the value as a sequence of ℓ bits, $v_{i,j}$, for $j = 0, \dots, \ell - 1$. Additionally, they enter two bits dir_i^b and

dir_i^s that indicate the direction of the order. All these values are checked to be bits, using the same check used in the bucket matching algorithm, and then the actual values of the volume in each direction are formed from $v_i^b = \text{dir}_i^b \cdot \sum_{j=0}^{\ell-1} v_{i,j} \cdot 2^j$ and $v_i^s = \text{dir}_i^s \cdot \sum_{j=0}^{\ell-1} v_{i,j} \cdot 2^j$. We still need to check that at least one of dir_i^b or dir_i^s is zero, so we calculate

$$\langle t_i \rangle = \langle \text{dir}_i^b \rangle \cdot \langle \text{dir}_i^s \rangle,$$

open $\langle t_i \rangle$ and check whether $t_i = 0$. Clearly, that happens if and only if either $\text{dir}_i^b = 0$ or $\text{dir}_i^s = 0$. To ensure the comparison $\langle c \rangle > 0$ can be evaluated correctly we simply need to pick parameters so that $n \cdot B < 2^{k-1}$.

2. The clearing phase one, where we open the orders in the direction that will be completely cleared. First, we need to check which list has largest total volume. To do so, we first calculate

$$\langle c \rangle \leftarrow \sum_{i=1}^n \langle v_i^b \rangle - \langle v_i^s \rangle.$$

Then, we perform the comparison $\langle c \rangle > 0$ and open the output. If c is greater than zero, the total buy volume is greater than the total sell volume and so we open the $\langle v_i^s \rangle$ share of every order i . Otherwise, we open the $\langle v_i^b \rangle$ shares. The $\langle \text{id} \rangle$ of non-dummy orders is also opened. We then calculate the total volume σ of the opened orders. Suppose the $\langle v_i^s \rangle$ shares were opened. For every $v_i^s = 0$, we calculate the cumulative buy volume of the first i orders, $\langle w_i \rangle = \sum_{h=1}^i \langle v_h^b \rangle$. If the $\langle v_i^b \rangle$ shares were opened, the cumulative sell volume is calculated instead. This cumulative volume will be used in the next clearing phase to avoid leaking the unmatched volume of the last opened order.

3. The clearing phase two, where we open the orders in the direction that will be only partially cleared. First, we run a binary search on the cumulative volume calculated previously to find the highest index u such that $\langle w_u \rangle < \sigma$. Then, the first u orders are opened, as well as the $\langle \text{id} \rangle$ of non-dummy orders. At this point, we still did not completely clear the orders opened during clearing phase one. However, if we open ord_{u+1} , part of its volume will remain unmatched and there will be an information leakage. To avoid this, we simply subtract the volume $\sigma - \langle w_u \rangle$ we still need from ord_{u+1} and open $\langle \text{id}_{u+1} \rangle$. This way, only the volume that will indeed be cleared is revealed, with the leftover volume of this last order remaining secret.

4 Leakage

There are two possible sources of information leakage in the described algorithms: (i) leakage from partially unmatched orders; and (ii) leakage from opening orders. Each of these sources is discussed below. All the analyses are equivalent when the buy orders have the largest total volume, thus we consider always the case when the total sell volume is more than the total buy volume.

Leakage from partially matched orders: This type of leakage can happen in both the volume match and the bucket-2 match, since in both of these algorithms there are orders with different volumes. In the bucket-1 match, every non-dummy order has exactly the same volume, so every opened order is completely matched and this type of leakage never happens.

In the volume match, orders from the direction with largest total volume are opened until the next order to be opened would finish clearing the other direction. We will then remove the volume we need to finish the clearing from this next order without opening its volume share. This means that the last order might still have some leftover volume, though it is also possible that all its volume was matched. Since it was at least partially matched, we need to reveal the investor who submitted the order so that the trade can be processed. We will therefore know that this investor might still have some volume left to trade and, if that is the case, we also know the direction of the order. The leftover volume in this last order and whether it is positive or not will however remain unknown.

In the bucket-2 match, the clearing phases one and two are the same as the bucket-1 match, and hence there is no leakage. As for clearing phases three and four, since the orders in each direction will have different volumes, the situation is similar to the volume match. Let unit^1 and unit^2 be the bucket sizes of the buy and the sell orders, respectively, in the clearing phases three and four. Considering $\text{unit}^1 = k \cdot \text{unit}^2$ for some $k \in \mathbb{N}$, if the sell orders have larger total volume, then there will be no leakage. If the buy orders have larger total volume, the unmatched volume will be $\text{leak} = h \cdot \text{unit}^2$, for $h \in \{0, \dots, k - 1\}$.

In case $\text{gcd}(\text{unit}^1, \text{unit}^2) = k$, for some $k \notin \{\text{unit}^1, \text{unit}^2\}$, then the unmatched volume will be either $\text{leak} \in \{0, k, 2k, \dots, \text{unit}^1 - k\}$, when the buy orders have largest total volume, or $\text{leak} \in \{0, k, 2k, \dots, \text{unit}^2 - k\}$, when the sell orders have largest total volume.

Note that for this algorithm the maximum leakage that can occur from unmatched orders is known, and the investors can plan how to divide their orders into the two lists according to this information.

Leakage from opening orders: Consider the bucket-1 match and suppose there are no dummy orders in a given auction. Let the sell orders be the ones with largest total volume, and hence the buy orders are the first ones to be opened. For each $\langle b_i \rangle$ that is revealed to be $b_i = 0$, we learn that this must be a sell order of unit volume. This means that as soon as we finish the clearing phase one, all the information about the orders' volume has been revealed.

Suppose now that the probability of having a dummy order is p_d , with the total number of dummy orders being $m = p_d \cdot n$. Let the buy orders be the first ones to be opened, and let the number of buy orders be $B = p_b \cdot (n - m)$ (note that here we must have $p_b \leq 1/2$ since there are less buy orders than sell orders). After clearing phase one, we will have $n - B$ orders which might be either dummies or sells, and the probability of finding a sell order is $\frac{n - B - m}{n - B}$. For each newly opened sell order, we learn whether an order is a sell or a dummy. Let i be the number of opened sell orders, and j the number of opened dummies, then

the probability of the next opened order being a sell is:

$$\Pr(\text{“order is sell”}) = \frac{n - B - m - i}{n - B - (i + j)}.$$

Assuming an even distribution of dummy orders within the buy orders.

By the end of clearing phase two, we should have opened a total of B sell orders plus m' dummy orders. At this moment, even if p_d is unknown, an adversary might use the information about previously opened orders and consider $p'_d = m'/(2B + m')$. The expected amount of leftover sell orders will then be $(n - 2B - m') \cdot (1 - p'_d)$. Note that, since we are in the bucketed case, knowing the amount of leftover sells implies knowing the total leftover sell volume.

In the bucket-2 match, the situation for the clearing phases one and two is identical to the bucket-1 match. For clearing phases three and four, we also know exactly the volume of each buy and sell order (even if this volume is different for buys and sells). However, note that these orders have a different format, i.e., they only contain the ID and either the sell or the buy volume, and so opening one of the directions does not leak information about the other. Therefore, the leakage associated with the opening of each of these lists will be the same as if we were continuing the clearing phase two openings.

The case for the volume match is similar, except that since each non-dummy order might have any positive volume, the uncertainty about the volume of unopened orders increases.

Summary The bucket-1 match has no leakage from partially matched orders, but there is some leakage from opening orders. In order to mitigate this effect, the investors must submit more dummy orders. The bucket-2 match does have leakage from partially matched orders (although this does not necessarily occur), in addition to the leakage from opening orders, which is similar to bucket-1 match. Once again, submitting dummy orders reduces this last type of leakage. Note also that when we have two lists, usually less non-dummy orders need to be submitted, so we can increase the proportion of dummy orders without getting worse runtimes than when using one list only.

Runtimes for different amounts of dummy orders are presented in Section 5. Note that for the chosen bucket sizes, bucket-2 match with 9 dummy orders per non-dummy order has faster runtimes than bucket-1 match with 5 dummy orders per non-dummy order. However, using bucket-2 match means we might get leakage from partially matched orders, depending on the balance between buy and sell orders in each list.

Volume match results in the least leakage. The leakage from partially matched orders corresponds only to the direction of a (possibly empty) order. The leakage from opening orders is minor when compared to bucket match, because of the uncertainty introduced by fact that orders can have any possible volume. This means that even if investors submit only 1 (or fewer) dummy order per non-dummy order, the leakage will remain low.

5 Runtimes

To provide runtimes of our algorithms, we model the situation where T investors participate in the auction, each of whom has one volume to submit drawn from the distribution $(\mathcal{N}(0, 1) + 5) \cdot 10^6$, and places the *same* order in three different auctions, each of which utilizes one of our three algorithms presented, namely volume match, bucket-1 match, and bucket-2 match. We varied T in $\{10, 100, 1000, 10000\}$, as well as the number of dummy orders submitted per non-dummy order (which we call d) in $\{0, 1, 5, 9\}$. Buy, sell, and dummy orders (when they exist) are evenly distributed in the lists of orders. We also assume that there is an *order imbalance* such that $2/5$ of the investors are buyers and $3/5$ are sellers.

This order imbalance was suggested through discussions with JPMorgan, a tier one US investment bank who operate in this space and have observed a tendency of investors to have a buy:sell imbalance in the ratio of 2:3. This conforms with evidence that informed investors tend to trade in the same direction (e.g., [25]). Here we model a sell imbalance ($3/5$ of investors are sellers), however buy imbalances (where $3/5$ of investors are buyers) also occur, depending on the mood of the market. For the protocols we have presented, results are symmetric such that a buy:sell imbalance of 2:3 has the same run time as a buy:sell imbalance of 3:2. If the imbalance is different or if there is no imbalance at all, the number of matched orders will be affected (assuming the submitted volume is drawn from the same distribution). This will influence the running time of the clearing phases, where we might need to reveal more or less id's. However, most of the total running time comes from the input phase and so a different order imbalance will not have a significant impact.

As a simplification, we present runtimes for the situation where there is only one auction trading one stock. However, a real world venue would allow trading in many stocks, so many auctions would be required. For instance, if the venue is trading 5000 different stocks then 5000 auctions are required. These auctions can be run sequentially, in which case the runtime for all auctions to complete is 5000 times the runtime of a single auction. Alternatively, multiple MPC engines can be used to run auctions in parallel. In the extreme case, where we have 5000 engines (i.e., one engine per stock), all auctions run in parallel and hence the total runtime for all auctions to complete is the same as the runtime presented for a single auction.

Setting: We used Scale-Mamba with Shamir secret sharing between $l = 3$ parties. All the parties run identical machines with an Intel i-9900 CPU and 128GB of RAM. The ping time between the machines is 1.003 ms.

Online phase of volume match: The average time for input phase depends on the bound B that is set for the volume of the orders. Recall that the orders' volumes are entered as a sequence of bits, and we must confirm that every one of them really is a bit. Therefore, the more bits we allow for the input volume,

the longer it will take to run this check. Here we assume that the volume of each order can have at most 32 bits, and we obtain an average time for the input phase of 0.00062 seconds (0.62 ms) per order, with a standard deviation of 0.00005 seconds (0.05 ms).

Runtimes are provided in Table 2 (see Appendix C, where we also provide a comparison of this version of volume matching to that described in [7]). One can notice that clearing phase 1 is faster than clearing phase 2. This is mainly due to the fact that the operation of opening directions can be vectorized for the case of clearing phase 1, as we are opening the direction of all orders, while for the case of clearing phase 2, this operation has to be sequential, as we do not know for how many orders we should open the direction.

Online phase of bucket-1 match: The average time of the input phase is 0.00013 seconds (0.13 ms) per order, with a standard deviation of 0.00001 seconds (0.01 ms). Note that the order format check is similar to the one used for the volume match, but here the volume of each order consists of a single bit, resulting in a faster input phase.

However, unlike what happens in the volume match, every order must now have the same fixed volume. This means that each investor must submit different non-dummy orders that sum up to the desired volume. When this volume is not a multiple of the chosen bucket size `unit`, we round the volume down to the closest multiple. Thus, we will generally have more orders than in the volume match, depending on the exact value of `unit`. If `unit` is small, more orders will be needed and the total submitted volume will be closer to the volume match case. If we choose `unit` to be large, we will not need as many orders, but the investors will submit significantly less volume than in the volume match case. The average number of orders and the average total submitted volume for different bucket sizes can be found in Table 4 (in Appendix C).

In our case, 99.7% of the investors will submit a volume between $2 \cdot 10^6$ and $8 \cdot 10^6$. If we choose e.g. `unit` = 10^6 , the volume submitted by each investor will be rounded down to the closest multiple of 10^6 . This will result in an average submitted volume of $4.49 \cdot 10^6$, as opposed to the average volume of $5 \cdot 10^6$ obtained in the volume match, where no rounding is needed. We will also have around 4.5 orders for each order in the volume match case.

We present in Table 3 (in Appendix C) the runtimes corresponding to the bucket match for one list with `unit` = 10^6 . One can make the same remark as the volume match for the runtimes. That is, clearing phase 1 is faster than clearing phase 2 due to the fact that we can vectorise computation for the case of clearing phase 1.

Online phase of bucket-2 match: Let `unitk` denote the bucket size associated with list L^k . We assume that `unit1` (the small bucket) is smaller than `unit2` (the big bucket).

Similar to bucket-1 match, the volume to be traded in bucket-2 match will be divided into multiple orders according to the bucket sizes. If the volume cannot

be fully obtained with a combination of the two buckets, we round it down to the closest possible combination. We assume that the investors will divide their volume such that they use as many big buckets as possible. The average number of orders in each list and the average total submitted volume for different bucket sizes can be found in Table 6.

Clearing phases 1 and 2 of step 1-II of Figure 4 (in Appendix B) are the same as bucket-1, except that now we run them on two lists. If we can run these two clearing phases in parallel (i.e., with two MPC engines) for the two lists, then it will be beneficial to have both lists with approximately the same number of orders. One thing to note is that clearing phases 3 and 4 were never performed. This is due to the fact that the order imbalance distribution we are considering always results in both lists having leftover orders of the same direction (i.e., sells), and clearing phases 3 and 4 take place only if the leftovers are from opposite directions.

Table 5 (in Appendix C) presents runtimes for bucket-2 match, with $\text{unit}^1 = 10^6$ and $\text{unit}^2 = 3 \cdot 10^6$. Note that the table contains only the time taken by clearing phases 1 and 2. One can notice that the runtimes of clearing phases 1 and 2 combined are similar for the two lists. This results from how we distributed orders among the lists. That is, as it was shown in Table 6 (in Appendix C), we would expect the same size on average for the two lists in the case of $\text{unit}^1 = 10^6$ and $\text{unit}^2 = 3 \cdot 10^6$. However, the slight differences in runtimes result from not only the size of the lists, but also how many orders were opened during the execution.

Summary: If clearing phases 3 and 4 of bucket-2 match are not executed then all three algorithms have roughly the same leakage, which in each case is extremely small and relies on estimating unmatched order volume by observing historical dummy ratios. In practice, this level of information leakage is negligible if investors use a randomised dummy order submission strategy. Assuming a 3:2 imbalance in orders to sell or buy, this implies that bucket-2 (or, more generally, bucket-z) is to be preferred as it has the quickest input phase. However, the precise trade off between the simple cost of input checking in bucket-z versus the more complicated cost of input checking in the volume matching algorithm depends on the exact distribution of dummy orders that investors submit in a real environment. Compared with volume match, bucket-z match is likely to incentivise the placement of more dummy orders to disguise the fact that each real order has a known volume equal to the bucket size. Once this number of additional dummy orders grows above some threshold, then volume match becomes more efficient than bucket-z match. For example, with $T = 1000$ investors, with a 9:1 ratio of dummy to real orders in bucket-2 match and a 1:1 ratio of dummy to real orders in volume match, volume match has an input phase of 1.24s and a clearing phase of 0.06s, whereas bucket-2 match has a longer input phase of 2.8s and a longer parallel clearing phase of 0.27s. However, in either scenario that bucket-z or volume match is quickest, the presented runtimes demonstrate that these algorithms can securely input and clear more than a thousand or-

ders per second, and are therefore clearly capable of handling the throughput requirements of a real world dark pool trading venue.

Acknowledgments

This work has been supported in part by ERC Advanced Grant ERC-2015-AdG-IMPACT, by the FWO under an Odysseus project GOH9718N, and by Cyber-Security Research Flanders with reference number VR20192203. Additionally, the first author is supported by the Flemish Government through FWO SBO project SNIPPET S007619N. The second author is sponsored by Refinitiv.

This paper was prepared in part for information purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates (“JPMorgan”), and is not a product of the Research Department of JPMorgan. JPMorgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful. 2021 JPMorgan Chase & Co. All rights reserved.

References

1. Aly, A., Cong, K., Cozzo, D., Keller, M., Orsini, E., Rotaru, D., Scherer, O., Scholl, P., Smart, N.P., Tanguy, T., Wood, T.: SCALE-MAMBA v1.12: Documentation (2021), <https://homes.esat.kuleuven.be/~nsmart/SCALE/Documentation.pdf>
2. Asharov, G., Balch, T.H., Polychroniadou, A., Veloso, M.: Privacy-preserving dark pools. In: Seghrouchni, A.E.F., Sukthankar, G., An, B., Yorke-Smith, N. (eds.) Proceedings of the 19th International Conference on Autonomous Agents and Multi-agent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020. pp. 1747–1749. International Foundation for Autonomous Agents and Multiagent Systems (2020)
3. Bag, S., Hao, F., Shahandashti, S.F., Ray, I.G.: SEAL: Sealed-bid auction without auctioneers. Cryptology ePrint Archive, Report 2019/1332 (2019), <https://eprint.iacr.org/2019/1332>
4. Balch, T., Diamond, B.E., Polychroniadou, A.: Secretmatch: inventory matching from fully homomorphic encryption. In: Balch, T. (ed.) ICAIF '20: The First ACM International Conference on AI in Finance, New York, NY, USA, October 15-16, 2020. pp. 15:1–15:7. ACM (2020), <https://doi.org/10.1145/3383455.3422569>
5. Bogetoft, P., Christensen, D.L., Damgaard, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J.D., Nielsen, J.B., Nielsen, K., Pagter, J., Schwartzbach, M., Toft, T.: Multiparty computation goes live. Cryptology ePrint Archive, Report 2008/068 (2008), <http://eprint.iacr.org/2008/068>

6. 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. LNCS, vol. 4107, pp. 142–147. Springer, Heidelberg (Feb / Mar 2006)
7. Cartlidge, J., Smart, N.P., Talibi Alaoui, Y.: MPC joins the dark side. In: Galbraith, S.D., Russello, G., Susilo, W., Gollmann, D., Kirde, E., Liang, Z. (eds.) ASIACCS 19. pp. 148–159. ACM Press (Jul 2019)
8. Cartlidge, J., Smart, N.P., Talibi Alaoui, Y.: Multi-party computation mechanism for anonymous equity block trading: A secure implementation of Turquoise Plato Uncross. Cryptology ePrint Archive, Report 2020/662 (2020), <https://eprint.iacr.org/2020/662>
9. Catrina, O., de Hoogh, S.: Improved primitives for secure multiparty integer computation. In: Garay, J.A., Prisco, R.D. (eds.) SCN 10. LNCS, vol. 6280, pp. 182–199. Springer, Heidelberg (Sep 2010)
10. Catrina, O., Saxena, A.: Secure computation with fixed-point numbers. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 35–50. Springer, Heidelberg (Jan 2010)
11. Constantinides, T., Cartlidge, J.: Block Auction: A general blockchain protocol for privacy-preserving and verifiable periodic double auctions. In: 2021 IEEE International Conference on Blockchain (Blockchain) (Dec 2021)
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. LNCS, vol. 3876, pp. 285–304. Springer, Heidelberg (Mar 2006)
13. Damgard, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority – or: Breaking the SPDZ limits. Cryptology ePrint Archive, Report 2012/642 (2012), <http://eprint.iacr.org/2012/642>
14. Galal, H., Youssef, A.: Publicly verifiable and secrecy preserving periodic auctions. In: Workshop on Trusted Smart Contracts (WTSC) (2021), <https://fc21.ifca.ai/wtsc/WTSC21paper2.pdf>
15. Hao, F., Zielinski, P.: A 2-round anonymous veto protocol. In: Christianson, B., Crispo, B., Malcolm, J.A., Roe, M. (eds.) Security Protocols. pp. 202–211. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
16. 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>
17. Keller, M., Rotaru, D., Smart, N.P., Wood, T.: Reducing communication channels in MPC. In: Catalano, D., De Prisco, R. (eds.) SCN 18. LNCS, vol. 11035, pp. 181–199. Springer, Heidelberg (Sep 2018)
18. Ngo, N., Massacci, F., Kerschbaum, F., Williams, J.: Practical witness-key-agreement for blockchain-based dark pools financial trading. In: Financial Cryptography and Data Security 2021 (2021), <https://fc21.ifca.ai/papers/113.pdf>
19. Parkes, D.C., Rabin, M.O., Shieber, S.M., Thorpe, C.A.: Practical Secrecy-Preserving, Verifiably Correct and Trustworthy Auctions, pp. 70–81. Association for Computing Machinery, New York, NY, USA (2006), <https://doi.org/10.1145/1151454.1151478>
20. 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>

21. Smart, N.P., Wood, T.: Error detection in monotone span programs with application to communication-efficient multi-party computation. In: Matsui, M. (ed.) CT-RSA 2019. LNCS, vol. 11405, pp. 210–229. Springer, Heidelberg (Mar 2019)
22. Thorpe, C., Parkes, D.C.: Cryptographic securities exchanges. In: Dietrich, S., Dhamija, R. (eds.) FC 2007. LNCS, vol. 4886, pp. 163–178. Springer, Heidelberg (Feb 2007)
23. Thorpe, C., Parkes, D.C.: Cryptographic combinatorial securities exchanges. In: Dingleline, R., Golle, P. (eds.) FC 2009. LNCS, vol. 5628, pp. 285–304. Springer, Heidelberg (Feb 2009)
24. Thorpe, C., Willis, S.R.: Cryptographic rule-based trading - (short paper). In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 65–72. Springer, Heidelberg (Feb / Mar 2012)
25. Zhu, H.: Do Dark Pools Harm Price Discovery? The Review of Financial Studies 27(3), 747–789 (12 2013)

A Background on Multi-Party Computation

A.1 Multiparty Computation

MPC allows a set of distrustful parties to perform computation on their inputs without having to reveal them. We distinguish two types of adversaries in MPC protocols: semi-honest adversaries, and malicious adversaries. Semi-honest adversaries are assumed to follow the exact description of the protocol, however, they may try to infer information from their view of the protocol about other players’ inputs. On the other hand, malicious adversaries may deviate from the protocol. Protocols that are secure against semi-honest adversaries and malicious adversaries are called passively secure protocols and actively secure protocols, respectively.

In this work, we will consider active security with abort. Roughly speaking, the adversary here can deviate from the protocol however this will be caught, with overwhelming probability, by honest parties who will abort the computation once this happens. We present in Figure 1 the formal description of the MPC functionality we are considering when processing an arithmetic circuit over the finite field \mathbb{F}_p .

A.2 Shamir Secret Sharing based MPC

We will consider Shamir Secret Sharing to realize the functionality of Figure 1, where a secret s is shared by giving player i the tuple $\{i, f_s(i)\}$, for a polynomial f_s of degree t with coefficients in a prime field \mathbb{F}_p , such that $f_s(0) = s$. A value s being shared as such will be denoted by $\langle s \rangle$ from now on. Clearly, if at least $t + 1$ players combine their shares, they can recover the secret s , and no set of at most t players can recover it. In our protocols we will require $t < n/2$, i.e. an honest majority, so as to enable efficient mechanism to perform multiplications.

MPC functionality $\mathcal{F}^{\mathcal{P}}[\text{MPC}]$

The functionality runs with $\mathcal{P} = \{P^1, \dots, P^l\}$ 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 $\text{varid} \in I$.

Initialize: On input (init, p) from all parties, where p is a prime, the functionality stores (domain, p) ,

Input: On input $(\text{input}, P^i, \text{varid}, x)$ from party P^i and $(\text{input}, P^i, \text{varid}, ?)$ from all other parties, with varid a fresh identifier and $x \in \mathbb{F}_p$, the functionality stores (varid, x) in memory.

Add: On command $(\text{add}, \text{varid}_1, \text{varid}_2, \text{varid}_3)$ from all parties, where $\text{varid}_1, \text{varid}_2$ are present in memory and varid_3 is not, the functionality retrieves (varid_1, x) , (varid_2, y) and stores $(\text{varid}_3, x + y)$ in memory.

Multiply: On input $(\text{multiply}, \text{varid}_1, \text{varid}_2, \text{varid}_3)$ from all parties, where $\text{varid}_1, \text{varid}_2$ are present in memory and varid_3 is not), the functionality retrieves (varid_1, x) , (varid_2, y) and stores $(\text{varid}_3, x \cdot y)$ in memory.

Output: On input $(\text{output}, \text{varid}, i)$ from all honest parties, where 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 **Deliver** then y is output to all parties if $i = 0$, or y is output to party i if $i \neq 0$. If the adversarial input is not equal to **Deliver** then \emptyset is output to all parties.

Figure 1: MPC functionality $\mathcal{F}^{\mathcal{P}}[\text{MPC}]$

Scale-Mamba: Scale-Mamba is a framework that implements various MPC protocols, including actively secure Shamir Secret Sharing based MPC with abort. Scale-Mamba adopts an offline-online methodology. That is, the computation can be split into two phases, a function independent offline phase where we pre-process data that will then be used in the function dependent online phase. This pre-processed data mainly consists of so-called Beaver triples. These are secret shared triples of the form $\{\langle a \rangle, \langle b \rangle, \langle c \rangle\}$, such that $c = a \cdot b$. The production of these Beaver triples is performed using the protocols of [17, 21], which reduces the total amount of communication needed per multiplication, compared to the traditional protocols. The protocols work since $t < n/2$.

Basic Arithmetic: Opening a secret shared value $\langle x \rangle$ can be performed by simply revealing the shares which make up $\langle x \rangle$ to all parties. The validity of this sharing can be checked, since $t < n/2$, using the error-detection properties of the Reed-Solomon code underlying the Shamir sharing. However, in [17, 21], a more efficient methodology for opening is given, which results in less communication. In particular, each player must only send just enough information to reconstruct the shared value. The checking of the values for correctness is performed by each player maintaining a running hash of the complete set of shares determining the sharing $\langle x \rangle$. These hashes are then compared with each other at the end of the

computation. As a shorthand, we denote such a global opening corresponding to the `Output` command of Figure 1 by $x \leftarrow \text{Open}(\langle x \rangle)$.

Addition comes for free with this secret sharing, that is, it consists of a local computation, as $f_x(i) + f_y(i) = (f_x + f_y)(i)$ for two secrets x and y . Indeed, any linear operation can be performed purely using local computation. We write such functions evaluations, for ease of expression, as $\langle z \rangle \leftarrow \alpha \cdot \langle x \rangle + \beta \cdot \langle y \rangle + \gamma$. To multiply two secrets, we consume one of the pre-processed triples from the offline phase. That is, to multiply $\langle x \rangle$ with $\langle y \rangle$, one can take a triple $\{\langle a \rangle, \langle b \rangle, \langle c \rangle\}$ and compute $\langle \epsilon \rangle \leftarrow \langle x - a \rangle$ and $\langle \gamma \rangle \leftarrow \langle y - b \rangle$, then opening them so as to get the product $\langle z \rangle \leftarrow \langle x \rangle \cdot \langle y \rangle$ by having the players compute $\langle z \rangle \leftarrow \langle c \rangle + \epsilon \cdot \langle b \rangle + \gamma \cdot \langle a \rangle + \epsilon \cdot \gamma$ which is a local computation. For shorthand we denote multiplication by $\langle z \rangle \leftarrow \langle x \rangle \cdot \langle y \rangle$.

Comparison: A key operation in our work will be to take two shared values $\langle x \rangle$ and $\langle y \rangle$ for $x, y \in \mathbb{F}_p$ and ‘think’ of the values x and y as representing integers in the range $(-p/2, \dots, p/2)$. In particular we think of x and y in the range $[-2^{k-1}, \dots, 2^{k-1}]$ for some k such that $2^k \ll p$. We will then wish to obtain a sharing $\langle b \rangle$ of the bit which represents the comparison of x and y , an operation that we will denote by $\langle b \rangle \leftarrow \langle x \rangle < \langle y \rangle$. To perform such comparisons, Scale-Mamba uses the protocols of [9, 10, 12]. At a high level, these protocols require that we additively mask a value x in $[-2^{k-1}, \dots, 2^{k-1}]$ by a random number r , then open the result $y \leftarrow \text{Open}(\langle x \rangle + \langle r \rangle)$. The number y will leak information about x if r is not chosen to be big enough. The statistical distance between the distribution from which y is drawn and the uniform distribution is $2^{-\text{sec}}$, if r is chosen from the interval $[-2^{\text{sec}+k-1}, \dots, 2^{\text{sec}+k-1}]$. Thus, `sec` has to be big enough to ensure an overwhelmingly small statistical distance. For our experiments, we chose k to be 64, and `sec` to be 40. Moreover, in order to guarantee that no overflows occur during computation, we require `sec` + $k < \log_2(p)$. For our experiments, we choose p of size 128.

A.3 The $\mathcal{F}_{\text{Rand}}()$ Functionality

We also require a functionality $\mathcal{F}_{\text{Rand}}()$ which allows the parties to agree on random values in \mathbb{F}_p . In practice this can be implemented by all parties committing to a seed, then the parties open the seeds. The seeds are then XOR’d together to produce a single shared seed, which is passed as the key to a PRF to produce the shared random value. We present this as an ideal functionality in Figure 2.

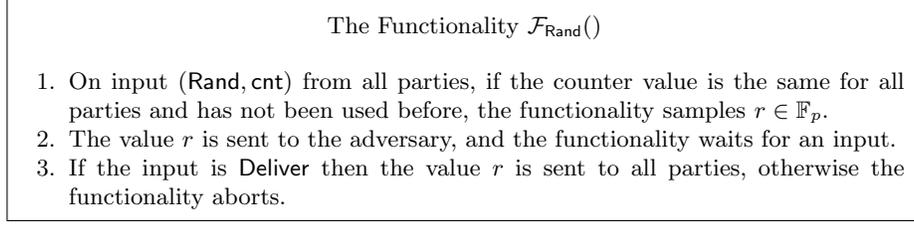


Figure 2: The Functionality $\mathcal{F}_{\text{Rand}}()$

B Secure Variants of the Auction Algorithms

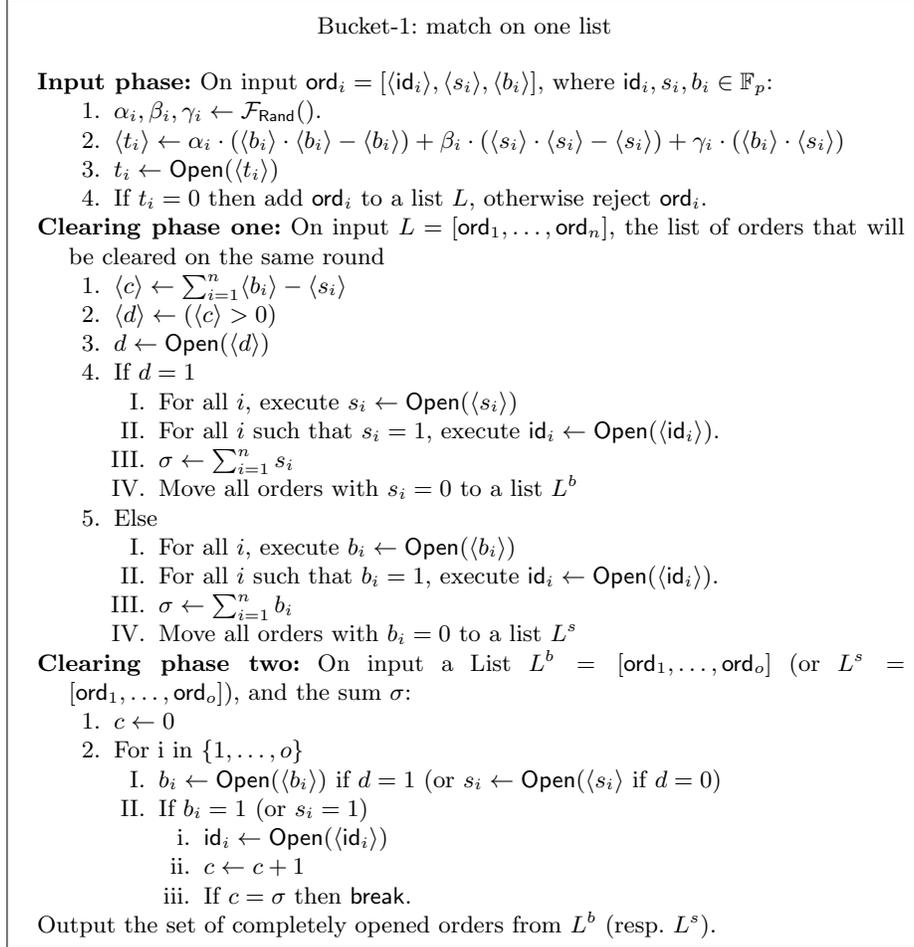


Figure 3: Bucket-1: match on one list

Bucket-2: match on two lists

Given two different buckets of sizes unit^1 and unit^2 , we take as inputs the orders $\text{ord}_i^1 = [\langle \text{id}_i \rangle, \langle s_i \rangle, \langle b_i \rangle]$ of volume unit^1 , and the orders $\text{ord}_i^2 = [\langle \text{id}_i \rangle, \langle s_i \rangle, \langle b_i \rangle]$ of volume unit^2 . For both bucket sizes, $\text{id}_i, s_i, b_i \in \mathbb{F}_p$.

Run Single Bucket Match Algorithm: First an instance of the bucket match algorithm on one list (Figure 3) is called for each bucket size:

1. Run the input phase of Figure 3 for each bucket size and put the valid orders in the lists L^1 and L^2 resp.
2. Run the clearing phases one and two from Figure 3 on L^1 and L^2 . After clearing phase one, we should have for each list j that either $\langle b^j \rangle$ or $\langle s^j \rangle$ have been opened for every order. Thus, after clearing phase two, we know that the remaining unopened orders are either orders in the unopened direction (if the unopened direction bit is 1) or dummy orders (if the unopened direction bit is 0).
3. If the leftover orders from lists L^1 and L^2 have different directions, put in R^1 and R^2 resp. the orders that were not matched from lists L^1 and L^2 . List R^j is of size n^j and its orders are of the form $[\langle \text{id}^j \rangle, \langle \text{dir}^j \rangle]$, where dir^j is s^j if the unopened direction from L^j is sell (or dir^j is b^j if the unopened direction from L^j is buy).
4. If they have the same direction then exit the algorithm.

Clearing phase three: On input $R^1 = [\text{ord}_1^1, \dots, \text{ord}_{n^1}^1]$ and $R^2 = [\text{ord}_1^2, \dots, \text{ord}_{n^2}^2]$

1. $\langle c \rangle \leftarrow \sum_{i=1}^{n^2} \langle \text{dir}_i^2 \rangle \cdot \text{unit}^2 - \sum_{i=1}^{n^1} \langle \text{dir}_i^1 \rangle \cdot \text{unit}^1$
2. $\langle d \rangle \leftarrow (\langle c \rangle > 0)$
3. $d \leftarrow \text{Open}(\langle d \rangle)$
4. If $d = 1$
 - I. For all i , execute $\text{dir}_i^1 \leftarrow \text{Open}(\langle \text{dir}_i^1 \rangle)$
 - II. $\sigma \leftarrow \sum_{i=1}^{n^1} \text{dir}_i^1 \cdot \text{unit}^1$
 - III. For all i such that $\text{dir}_i^1 = 1$, execute $\text{id}_i \leftarrow \text{Open}(\langle \text{id}_i \rangle)$
 - IV. $\sigma \leftarrow \sigma / \text{unit}^2$
5. Else
 - I. For all i , execute $\text{dir}_i^2 \leftarrow \text{Open}(\langle \text{dir}_i^2 \rangle)$
 - II. $\sigma \leftarrow \sum_{i=1}^{n^2} \text{dir}_i^2 \cdot \text{unit}^2$
 - III. For all i such that $\text{dir}_i^2 = 1$, execute $\text{id}_i \leftarrow \text{Open}(\langle \text{id}_i \rangle)$
 - IV. $\sigma \leftarrow \sigma / \text{unit}^1$

Clearing phase four: On input a List $R^2 = [\text{ord}_1^2, \dots, \text{ord}_{n^2}^2]$ (or $R^1 = [\text{ord}_1^1, \dots, \text{ord}_{n^1}^1]$), and the sum σ :

1. $c \leftarrow 0$
2. For i in $\{1, \dots, n^2\}$ (or i in $\{1, \dots, n^1\}$)
 - I. $\text{dir}_i^2 \leftarrow \text{Open}(\langle \text{dir}_i^2 \rangle)$ if $d = 1$ (or $\text{dir}_i^1 \leftarrow \text{Open}(\langle \text{dir}_i^1 \rangle)$ if $d = 0$)
 - II. If $\text{dir}_i^2 = 1$ (or $\text{dir}_i^1 = 1$)
 - i. $\text{id}_i^2 \leftarrow \text{Open}(\langle \text{id}_i^2 \rangle)$ (or $\text{id}_i^1 \leftarrow \text{Open}(\langle \text{id}_i^1 \rangle)$)
 - ii. $c \leftarrow c + 1$
 - iii. If $c = \sigma$ then break.

Figure 4: Bucket-2: match on two lists

Volume match

Input phase: On input $\text{ord}'_i = [\langle \text{id}_i \rangle, \langle v_{i,j} \rangle, \langle \text{dir}_i^b \rangle, \langle \text{dir}_i^s \rangle]$, where $\text{id}, v_{i,j}, \text{dir}_i^b, \text{dir}_i^s \in \mathbb{F}_p$:

1. $\langle v_i^s \rangle \leftarrow \text{dir}_i^s \cdot \sum_{j=0}^{\ell-1} \langle v_{i,j} \rangle \cdot 2^j$.
2. $\langle v_i^b \rangle \leftarrow \text{dir}_i^b \cdot \sum_{j=0}^{\ell-1} \langle v_{i,j} \rangle \cdot 2^j$.
3. $\text{ord}_i = [\langle \text{id}_i \rangle, \langle v_i^s \rangle, \langle v_i^b \rangle]$
4. $\alpha_{i,j} \leftarrow \mathcal{F}_{\text{Rand}}()$ for $j = 0, \dots, \ell - 1$.
5. $\beta_{i,1}, \beta_{i,2} \leftarrow \mathcal{F}_{\text{Rand}}()$.
6. $\gamma_i \leftarrow \mathcal{F}_{\text{Rand}}()$.
7. $\langle t_i \rangle \leftarrow \sum_{j=0}^{\ell-1} (\alpha_{i,j} \cdot (\langle v_{i,j}^b \rangle \cdot \langle v_{i,j}^s \rangle - \langle v_{i,j}^b \rangle)) + \beta_{i,1} \cdot (\langle \text{dir}_i^b \rangle \cdot \langle \text{dir}_i^b \rangle - \langle \text{dir}_i^b \rangle) + \beta_{i,2} \cdot (\langle \text{dir}_i^s \rangle \cdot \langle \text{dir}_i^s \rangle - \langle \text{dir}_i^s \rangle) + \gamma_i \cdot (\langle \text{dir}_i^b \rangle \cdot \langle \text{dir}_i^s \rangle)$
8. $t_i \leftarrow \text{Open}(\langle t_i \rangle)$
9. If $t_i = 0$ then add ord_i to a list L , otherwise reject ord'_i .

Clearing phase one: On input $L = [\text{ord}_1, \dots, \text{ord}_n]$, the list of orders that will be cleared on the same round

1. $\langle c \rangle \leftarrow \sum_{i=1}^n \langle v_i^b \rangle - \langle v_i^s \rangle$
2. $\langle d \rangle \leftarrow (\langle c \rangle > 0)$
3. $d \leftarrow \text{Open}(\langle d \rangle)$
4. If $d = 1$
 - I. For all i , execute $v_i^s \leftarrow \text{Open}(\langle v_i^s \rangle)$
 - II. For all i such that $v_i^s > 0$, execute $\text{id}_i \leftarrow \text{Open}(\langle \text{id}_i \rangle)$
 - III. $\sigma \leftarrow \sum_{i=1}^n v_i^s$
 - IV. For all i such that $v_i^s = 0$, execute $\langle w_i \rangle \leftarrow \sum_{h=1}^i \langle v_h^b \rangle$ and move $\langle w_i \rangle$ to a list W .
 - V. Move all orders with $v_i^s = 0$ to a list L^b
5. Else
 - I. For all i , execute $v_i^b \leftarrow \text{Open}(\langle v_i^b \rangle)$
 - II. For all i such that $v_i^b > 0$, execute $\text{id}_i \leftarrow \text{Open}(\langle \text{id}_i \rangle)$
 - III. $\sigma \leftarrow \sum_{i=1}^n v_i^b$
 - IV. For all i such that $v_i^b = 0$, execute $\langle w_i \rangle \leftarrow \sum_{h=1}^i \langle v_h^s \rangle$ and move $\langle w_i \rangle$ to a list W .
 - V. Move all orders with $v_i^b = 0$ to a list L^s

Clearing phase two:

On input a list $L^b = [\text{ord}_1, \dots, \text{ord}_o]$ (or $L^s = [\text{ord}_1, \dots, \text{ord}_o]$), a list $W = [\langle w_1 \rangle, \dots, \langle w_o \rangle]$ and the sum σ :

1. Run a binary search on list W to find the highest u satisfying $\langle w_u \rangle < \sigma$.
2. For i in $\{1, \dots, u\}$
 - I. $v_i^b \leftarrow \text{Open}(\langle v_i^b \rangle)$ if $d = 1$ (or $v_i^s \leftarrow \text{Open}(\langle v_i^s \rangle)$ if $d = 0$)
 - II. If $v_i^b > 0$ (or $v_i^s > 0$), execute $\text{id}_i \leftarrow \text{Open}(\langle \text{id}_i \rangle)$
3. $\langle v_{u+1}^b \rangle \leftarrow \langle v_{u+1}^b \rangle - \langle w_u \rangle + \sigma$ (or $\langle v_{u+1}^s \rangle \leftarrow \langle v_{u+1}^s \rangle - \langle w_u \rangle + \sigma$)
4. $\text{id}_{u+1} \leftarrow \text{Open}(\langle \text{id}_{u+1} \rangle)$

Figure 5: Volume match

C Experimental Runtimes

Table 2: Volume match runtimes in seconds. Each investor submits a single non-dummy order with volume drawn from the distribution $(\mathcal{N}(0, 1) + 5) \cdot 10^6$. d is the number of dummy orders submitted per non-dummy order and n is the total number of orders submitted by all the investors.

Investors	d	n	Clearing 1	Clearing 2	Traded Volume
10	0	10	0.0008	0.0025	2.25e+07
	1	20	0.0008	0.0033	
	5	60	0.0009	0.0053	
	9	100	0.001	0.0062	
100	0	100	0.0021	0.0064	1.98e+08
	1	200	0.0023	0.0096	
	5	600	0.0032	0.019	
	9	1000	0.0041	0.0276	
1000	0	1000	0.0143	0.03	1.99e+09
	1	2000	0.0157	0.0496	
	5	6000	0.0258	0.1325	
	9	10000	0.0344	0.2079	
10000	0	10000	0.1344	0.2409	1.99e+10
	1	20000	0.1537	0.4416	
	5	60000	0.2493	1.2372	
	9	100000	0.3382	2.038	

Comparison with the volume matching from [7]: We compare here between our algorithm for the volume match, and the one from [7], with respect to the functionalities provided, the leakage induced, and the online time required.

- **Functionalities:** Both algorithms take a set of orders and match them. However, we also implemented a step where we check the correctness of the orders, and we did not omit opening the id's of the orders when they are matched. Besides, we provided to investors the possibility of inserting dummy orders. We will see the impact of this in the leakage comparison.
- **Leakage:** To properly compare the leakage induced in both algorithms, we will split the analysis into 3 parts: Leakage prior to the clearing phase (i.e., clearing phases 1 and 2 in our case); Leakage during the clearing phase; and Leakage after the clearing phase.

Prior to the clearing phase, in our algorithm, orders are entered without leaking their direction, and these orders will be placed in the same list,

while in the volume match of [7], investors need to specify whether their orders are sell or buy orders, so as to put them in the corresponding lists. Besides, as we allow dummy orders, an adversary will not be able to identify non-dummy orders from the ones inserted, while in [7] dummy orders were not considered. During the clearing phase, [7] does not leak anything, while in our case, we leak either the sell or the buy orders will be fully matched. However, we argue that tolerating this minor leakage, was one of the reasons for which we obtained better runtimes for the clearing phases as we will see in online time comparison. After the clearing phase, as dummy orders were not considered in [7], an adversary will know how many orders remain, as well as the corresponding directions, while in our case, the leakage we induce is strictly less than in [7] thanks to the usage of dummy orders, as we have shown in the leakage analysis in section 4.

- **Online time:** We have implemented our volume match under the setting of 3 players using Shamir Secret Shamir based MPC. This setting was also considered in [7]. We have used better machines to implement our algorithm, but we argue that this does not contribute much to the difference of the runtimes between the two implementations.

To compare the runtimes between the two algorithms, we can take for instance the case where $n = 1000$ orders with no dummies, which is equivalent to having two lists (a sell list and a buy list) in [7] that sum up to 1000 orders. For this setting, we can execute the clearing phase with all the orders being matched (which results in the slowest runtime we can obtain) in 0.044 seconds, while in [7], the runtime will not depend on the number of cleared orders, and will be 0.9 seconds.

However, taking advantage of our reduced leakage compared to [7], requires inserting dummy orders, thus one would expect a bigger list for our case. As it is shown in our experiments, if we consider for instance the investors submitting 9 dummy orders for each non-dummy order, for $T = 1000$ we obtain 0.355 seconds for the case where all orders are matched, which is about three times faster than the case of [7].

Finally, the input phase in our case will require 6.2 seconds for $n = 10000$. This step was omitted in [7], where one would expect the usage of a similar trick as ours to input orders, that is, the investors will input the bits corresponding to the volume to be traded, which will be summed up and checked for correctness. We estimate the runtime of the input phase of [7] for one order to be 0.44 ms (with a standard deviation of 0.08 ms), and therefore 0.44 seconds for $n = 1000$.

Thus, the overall summary is that we provided more functionalities, with less leakage, and faster runtimes for the clearing phase. This was feasible due to avoiding the heavy use of comparisons in our volume match, which are expensive with MPC. However, our input phase will penalise our runtimes more than the ones of [7], as we have more checks to perform due to dummy inputs as well as the form of our orders. That is, our input phase is 40 % slower than the one of [7]. Besides, assuming that we would expect more

orders than the case of [7], this will slow down our input phase by a factor of $d + 1$.

Table 3: Bucket-1 match runtimes in seconds. Each investor submits a total volume drawn from the distribution $(\mathcal{N}(0,1) + 5) \cdot 10^6$, rounded down to the closest multiple of the bucket size $\text{unit} = 10^6$. d is the number of dummy orders submitted per non-dummy order and n is the total number of orders submitted by all the investors.

Investors	d	resulting n	Clearing 1	Clearing 2	Traded Volume
10	0	47	0.0014	0.0013	2.10e+07
	1	94	0.0013	0.0023	
	5	282	0.0016	0.0067	
	9	470	0.002	0.011	
100	0	439	0.0063	0.0102	1.76e+08
	1	878	0.007	0.0186	
	5	2634	0.01	0.0538	
	9	4390	0.0133	0.0906	
1000	0	4473	0.0586	0.1046	1.79e+09
	1	8946	0.0675	0.2002	
	5	26838	0.0946	0.5522	
	9	44730	0.129	0.8971	
10000	0	45022	0.5856	1.0515	1.79e+10
	1	90044	0.6554	1.9466	
	5	270132	0.9379	5.5058	
	9	450220	1.2819	9.0969	

Table 4: Average number of submitted orders and volume according to the bucket size unit.

Bucket Size	# Orders	Average Submitted Volume
Volume match	n	$(5 \cdot 10^6) \cdot n$
$1 \cdot 10^5$	$49.4n$	$(4.94 \cdot 10^6) \cdot n$
$5 \cdot 10^5$	$9.5n$	$(4.74 \cdot 10^6) \cdot n$
$1 \cdot 10^6$	$4.5n$	$(4.49 \cdot 10^6) \cdot n$
$2 \cdot 10^6$	$2n$	$(3.99 \cdot 10^6) \cdot n$

Table 5: Runtimes for bucket-2 match in seconds. Each investor submits a total volume drawn from the distribution $(\mathcal{N}(0, 1) + 5) \cdot 10^6$, and the volume is divided such that as many big buckets as possible are used. The bucket sizes are $\text{unit}^1 = 10^6$ and $\text{unit}^2 = 3 \cdot 10^6$. d is the number of dummy orders submitted per non-dummy order and n is the total number of orders submitted by all the investors.

Investors	d	n for list 1	Clearing phases 1 + 2	n for list 2	Clearing phases 1 + 2	Traded Volume
10	0	8	0.0009	13	0.0012	2.10e+07
	1	16	0.001	26	0.0015	
	5	48	0.0015	78	0.0027	
	9	80	0.0021	130	0.004	
100	0	112	0.0044	109	0.0048	1.76e+08
	1	224	0.0066	218	0.0073	
	5	672	0.0156	654	0.0176	
	9	1120	0.0238	1090	0.0271	
1000	0	1101	0.0403	1124	0.0428	1.79e+09
	1	2202	0.0643	2248	0.0687	
	5	6606	0.1585	6744	0.1699	
	9	11010	0.2506	11240	0.2705	
10000	0	10918	0.4001	11368	0.4145	1.79e+10
	1	21836	0.671	22736	0.6929	
	5	65508	1.6081	68208	1.6681	
	9	109180	2.5474	113680	2.6297	

Table 6: Average number of submitted orders and volume according to the bucket sizes unit^1 and unit^2 .

Bucket Size 1	Bucket Size 2	# Orders 1	# Orders 2	Average Submitted Volume
Volume match		n (single list)		$(5 \cdot 10^6) \cdot n$
$1 \cdot 10^5$	$1 \cdot 10^6$	$4.5n$	$4.5n$	$(4.94 \cdot 10^6) \cdot n$
$5 \cdot 10^5$	$1 \cdot 10^6$	$0.5n$	$4.5n$	$(4.74 \cdot 10^6) \cdot n$
$5 \cdot 10^5$	$2 \cdot 10^6$	$1.5n$	$2n$	$(4.74 \cdot 10^6) \cdot n$
$1 \cdot 10^6$	$2 \cdot 10^6$	$0.5n$	$2n$	$(4.49 \cdot 10^6) \cdot n$
$1 \cdot 10^6$	$3 \cdot 10^6$	$1.1n$	$1.1n$	$(4.49 \cdot 10^6) \cdot n$

C.1 Online runtimes comparison

Regarding the input phase, the volume match is about four times slower than the bucket match. This results from the fact that an order for the volume match has the same form as an order for the bucket match, in addition to $l = 32$ bits specifying volume, which all have to be checked for correctness. For both algorithms, the input phase takes significantly more time than the clearings, so the fourfold time increase in the volume match will have a big impact on the total runtime. However, if the investors for the auction in question tend not to trade big volumes, one can dedicate fewer bits, l , for the size of the volumes to be traded, which will decrease the time needed for the input phase of volume match.

If we consider an equal number of orders, dummies and matches, clearing phase one of bucket-1 match is identical to clearing phase one of bucket-2 match. However, this will be executed twice for the latter as we have two lists. As for phase one of the volume match, it has an extra calculation that consists of computing the cumulative volumes of the orders that will be cleared in clearing phase 2. This, however, does not have a significant impact on the runtimes.

Clearing phase two is identical for bucket-1 match and bucket-2 match. However, it will be executed twice for bucket-2. Clearing phase two for volume match is also similar, except that we execute at the beginning a binary search that performs comparisons on the volumes of orders. This induces a slowdown in the runtimes compared to clearing phase 2 of the bucket match, however, the extra time produced is not significant as we only perform $\log(n)$ comparisons, which is negligible to the amount of computation performed afterwards.

Clearing phases three and four exist only in bucket-2 match, when the leftover orders from each list are in opposite directions. The runtime of these phases depends of course on the way buy and sell orders are distributed in the two initial lists. Although this adds an extra component to the total runtime, the impact is small when compared to the input phase runtime.

Note, however, that if we consider an equal number of orders, dummies and matches, the volume traded in each algorithm will generally not be the same. Hence, to provide an accurate analysis, we consider a fixed number of investors that generate their orders according to the distribution we considered for the experiments, namely $(\mathcal{N}(0, 1) + 5) \cdot 10^6$. As before, investors also have an imbalanced ratio of 3 sellers to 2 buyers.

In the volume match, since one can submit orders of any volume, trading large volumes becomes easier. Besides, investors will be capable of placing the exact volumes they wish. The number of dummies should also decrease because the uncertainty about the volume of each unopened order is higher than the bucket case.

For the bucket-1 match, the chosen bucket size should not be very large, so that both small and large volumes can be submitted by each investor. For trading large volumes, several orders must be submitted, and hence there will be typically more orders than in the bucket-2 match or in the volume match. Additionally, more dummies must also be submitted to decrease the leakage

from opening orders. However, note that investors will not be able to place the exact volume they wish, as volumes are bound to the unit `unit` considered for the auction. Thus, each investor will submit an order with lower volume than they wish to trade. This difference is minimised by reducing the size of `unit`. However, as `unit` decrease, the number of orders increases and the slower the auction will run, as shown in Table 4.

In bucket-2 match, clearings one and two are similar to the volume match, in the sense that large volumes can be traded without the need to submit a large number of orders. On the other hand, since there are only two possible volumes for each non-dummy order, the number of dummies should be greater than in the volume match case, and one would expect the same as in bucket-1 match. Besides, similarly to bucket-1 match, there will be some trade volume lost. This volume lost will be more significant than bucket-1 match if `unit < unit1` and vice versa. In general, assuming there is an order imbalance and as a consequence clearing phases 3 and 4 are not executed, bucket match with $z + 1$ lists should be faster than bucket match with z lists, if we dedicate for each list an MPC engine. This should also hold if we dedicate only one engine for all lists, as adding one list will decrease the total number of orders and therefore the input phase will be faster, but also the clearing phases should become faster in general, as they will be dealing with lists of smaller sizes. For instance, in our experiments, for the case of $T = 1000$ and $d = 9$, the input phase for bucket-1 match requires $44730 \cdot 0.00013 = 5.81$ seconds, while this required $(11010 + 11240) \cdot 0.00013 = 2.89$ seconds for the two lists of bucket-2 match. Besides, the clearing required 1.03 seconds for bucket-1 match, and $0.2506 + 0.2705 = 0.52$ seconds for the two lists of bucket-2 match.

As to the bucket match with z lists compared to the volume match, as stated earlier, the input phase for the volume match is about five times the one of the bucket match (precisely 4.76 times). However, one would expect fewer orders submitted to the auction for the volume match than the bucket match, because we will have fewer dummy orders for the volume match, and also because for each order submitted in the volume match, there will be at least one order submitted in one of the lists among the z lists, and most likely at least two orders among the z lists for a rational choice of the units. Thus, if we consider for instance that investors tend to place 9 times more dummies in the bucket match than the volume match, the input phase for the bucket match will be at least $\frac{2 \cdot 10}{4.76 \cdot 2} = 2.1$ times the one for the volume match. Thus, the volume match will be faster than the bucket match with z lists as the runtimes are dominated by the input phase. For instance, for the case of $T = 1000$, with $d = 9$ for the bucket match and $d = 1$ for the volume match, where we had on average $4.5 \cdot n$ orders for bucket-1 match, and $1.1 \cdot n$ orders in each list of the bucket-2 match, the input phase for volume match, bucket-1 match, and bucket-2 match were, respectively, $2000 \cdot 0.00062 = 1.24s$, $44730 \cdot 0.00013 = 5.81s$, and $(11010 + 11240) \cdot 0.00013 = 2.8s$; and for clearing phases, they are respectively $0.0157 + 0.0496 = 0.06s$, $0.129 + 0.8971 = 1.03s$, and $0.2506 + 0.2705 = 0.52s$ if run sequentially or $0.27s$ in a parallel execution.