PET-Exchange: A Privacy Enhanced Trading Exchange using Homomorphic Encryption

David Hasselquist*†, Jacob Wahlman*‡, Niklas Carlsson*
*Linköping University, Sweden
†Sectra Communications, Sweden
‡Nasdaq, Inc.

***Abstract*—The underlying trading mechanisms of electronic securities exchanges have mostly stayed the same over the years with some additions and improvements. However, over the recent decade, high-frequency traders using algorithmic trading have shifted the field using practices that many consider unfair or unethical. In addition, insider trading continues to cause trust issues on certain trading platforms. In this paper, we present PET-Exchange, a privacy-preserving framework for trading securities on an electronic stock exchange. By using homomorphic encryption, PET-Exchange prevents information disclosures and unfair advantages in the trading processes. By matching and trading encrypted orders, we study the performance under various volumes and timing constraints, and compare this to the unencrypted counterparts. Our analysis of PET-Exchange using market trade data shows the privacy and cryptographic tradeoffs, demonstrating it to be suitable for small-scale trading and privacy-preserving auctions. Finally, we discuss the potential impact on transparency, fairness, and opportunities for financial crime in an electronic securities exchange. The insights we provide take us one step closer to a privacy-aware and fair public securities exchange.**

I. Introduction

Since the introduction of electronic securities exchanges, the trading process has been made more easily available for users. Today, electronic securities exchanges handle millions of trades every market day with billions of shares changing owners. On these exchanges a user can place either buy orders (*bid*) or sell orders (*ask*) for a given price and volume of a financial asset. Trading securities on a public securities exchange such as the New York Stock Exchange or Nasdaq implies a certain transparency of the exchange's processes, where the transparency aims to ensure that no participant is offered an unfair advantage. However, the transparent process also entails risks contributing to unfair practices where malicious traders or high-frequency traders can gain an unfair advantage using tactics such as *front running* [1], *penny jumping* [2], or variants of insider trading [3].

By placing orders, users must trust that the exchange preserves the transaction privacy. When matching ask and bid orders, the exchange must first decrypt each order, creating a potential breach of the privacy chain. Somebody may therefore ask why a trader should trust that a broker or an exchange cannot and will not use the decrypted order information to gain an unfair advantage?

A method that larger broker firms use to avoid issues stemming from transparency is to perform trades in an off-exchange market. These alternative trading systems are commonly referred to as dark pools [4]. Here, users or brokers can submit pre-trade bids and wait to disclose the trades publicly until they have been executed. However, in recent years, the United States Securities and Exchange Commission (SEC) and its commissioners have voiced concerns about the lack of regulation surrounding dark pools and the unfair advantage it might provide its participants [4]. This has raised significant interest in alternative solutions to dark pools that enable both transparency and privacy.

Various efforts have been made to prevent unfair practices by utilizing cryptography. Today, encryption is used on public securities exchanges to encrypt orders and order books at rest and in transit. However, no practical solution has been implemented in public securities exchanges that handle encrypted orders, causing possible information leaks.

Homomorphic encryption is one potential solution to the above trust dilemma associated with public exchanges, allowing operations to be performed directly on ciphertexts without first decrypting the data [5]. The result of these operations is a new ciphertext encrypted under the original secret. When decrypted, the results correspond to the same value as if the operation would have been applied to the plaintext. Since homomorphic encryption allows operations to be performed directly on ciphertexts, a broker or exchange (in our context) do not need to access the plaintext to perform matching of ask and bid orders. If such functionality were implemented in a way that the exchange could still offer the same service, the use of homomorphic encryption could potentially help address the above trust dilemma. However, implementing the desirable functionalities is not trivial, and the solutions often come with additional performance overheads. For example, homomorphic operations can be highly resource intensive, and these solutions typically increase storage allocation requirements significantly compared to traditional cryptographic methods.

Fortunately, recent innovations have enabled new, improved homomorphic encryption schemes, making them applicable for increasingly more performance-intensive tasks. Based on our experience implementing a securities exchange using homomorphic encryption, we argue that this is the right time to apply it to financial applications such as securities exchanges.

In this paper, we present PET-Exchange, a privacy-preserving trading framework for trading securities where limit orders can be partially matched between multiple buyers and sellers. By carefully applying homomorphic encryption, our exchange model is capable of inserting into order books, matching, and executing limit orders despite them being encrypted. First, we present a theoretical model that outlines the design of each functionality. This includes the design of methods for using homomorphic encryption to preserve the privacy of the orders and for non-revealing continuous double auction matching of prices and volumes. We assume an honest-but-curious threat model where actors can observe all internal operations and attempt to extract sensitive information. By limiting the information that each entity can extract, giving them access only to the information needed to complete their task, we prevent information disclosure and limit the trust placed on each component of the system. Second, we validate our proof-of-concept implementation of PET-Exchange by simulating trades under various conditions. Third, we present a performance evaluation that provides insights into the most attractive tradeoffs associated with using two fundamentally different ways of comparing orders, different configurations (with different overhead-accuracy tradeoffs), and overheads associated with specific aspects of our implementation (e.g., additional "dummy" orders for validating the third-party evaluator). Our evaluation also provides insights into current performance bottlenecks and validates that we can achieve high accuracy with many decimals of precision (important as exchanges require floating point operations).

Overall, our results demonstrate that PET-Exchange can achieve the benefits of homomorphic encryption, effectively addressing the targeted trust dilemma without sacrificing accuracy. Our proof-of-concept implementation highlights that the framework design and variations thereof are possible today, opening the door to better prevent unfair trading practices in electronic securities exchanges. Finally, studying the most attractive configuration tradeoffs, performance insights are made applicable to a wide range of exchanges and use cases.

To put the work in the context of related work (Section VII), we first acknowledge that homomorphic encryption has been studied to improve user privacy in the securities market context. However, unique to this work is that no previous study addresses the challenge of matching limit orders, where orders are continuously private throughout the trading process until execution. Furthermore, no study has presented a privacy-preserving solution allowing partially-filled limit orders. While previous works have studied privacy-preserving auctions, these often fail to consider cases with multiple winners and fail to show a method for delaying the disclosure of auction winners.

Outline: Section II presents an overview of securities exchanges. Section III presents our PET-Exchange model, its components, and the role of each party. Section IV explains the design of our order handling. Our performance evaluation is presented in Section V. Here, we focus on relative differences and provide system insights into the performance, overheads, security, and accuracy associated with different implementation options. Section VI discusses various confidentiality, security, and ethical considerations. Finally, Section VII presents related works and Section VIII our conclusions.

II. Securities Exchange

Securities exchanges are trading platforms where securities are traded. Since the 1971 introduction of the Nasdaq electronic trading platform, the classical physical trading has shifted to electronic securities exchanges [6]. With this shift, several new trading methods have evolved. One such method is algorithmic trading with the goal of lowering latency. The reduced latency allowed brokers to react even quicker to movements in securities prices than any human could [2].

Securities are one type of instrument, a contract for a financial asset that can be transferred or held [7]. A public securities exchange allows users to trade outstanding securities and various instruments in publicly traded companies listed on the exchange. Exchanges also provide clearing functionality that acts as a neutral intermediate between the buyer and the seller to handle the payment.

Traders can typically use different order types on securities exchanges. Common order types include market orders, stop orders, limit orders, and many variations that usually introduce further constraints or conditions for execution [6]. Market orders are executed at the best available price, determined at the time of execution [6]. Stop orders are categorized into buy and sell stop orders, defining a limit where the orders turn into market orders [6]. Limit orders are executed at a specific limit price or better, where a bid order is executed at a limit price or below, while ask orders are executed at the limit price or above [6]. In this paper, we focus on privacy-preservation of limit order entry, matching, and trading, but note that our approach can also be generalized for other order types.

A. *Unfair Practices in Trading*

Unfair practices in securities exchange trading can lead to advantages for a group of traders at the cost of other traders. These practices influence the market and price of securities, often using non-public information. For example, some malicious traders can gain an unfair advantage by analyzing active and incoming orders before they are published or executed. This includes high-frequency traders using algorithmic trading with powerful computers to place orders on the securities exchange and consume information at a pace that is not possible for smaller traders [2]. Penny jumping [2] is a trading strategy not considered illegal but, in many cases, unfair. Here, traders place orders slightly above or below existing ones to ensure their bid or ask orders get executed first, with minimal price difference [2].

An example of illegal trading is front running, which is an action of capitalizing on a trade with knowledge about an upcoming trade that has not been publicly disclosed [1]. These trades are illegal as the trader uses non-public information to manipulate the price before the trade has been disclosed.

B. *Dark pools*

A reaction to the imposed transparency by public securities markets is the alternative trading systems, commonly referred to as dark pools [4]. A dark pool is a more privacy preserving off-exchange market and an alternative to the public exchange markets. Here, traders can hide their trading intents from other malicious traders and algotraders [4]. Dark pools allow participants to trade securities with a limited number of participants. In some cases, dark pool operators hide other active orders, so a client might not be aware that a counterpart order exists until they have submitted their order [4]. Compared to the public securities exchanges, where quotes are published on securities before the trade to enable transparency for order information pre-trade and post-trade, dark pools only publish post-trade data after the trade has been executed. Therefore, the dark pool trade is considered to be executed outside the exchange [4].

The need for an enhanced privacy trading market exchange has increased over the years, making dark pool trading more popular. The dark pools generally see fewer orders as there are fewer applicable traders, but the volume in each order is often larger. Recently, shares for popular securities have seen increased trading in dark pools. For example, the AMC stock has seen 64% of its trading volume being on dark pools [8], clearly showing the desire for privacy enhanced trading.

However, dark pool operators have been investigated for numerous violations of unfair practices in trading, including penny jumping [2] and front-running traders in the pool [4]. These unfair practices have raised concerns of dark pool trading among the regulators of financial markets [4]. Dark pools are also often criticized for their privacy and unavailable marketplace for smaller traders [4]. Therefore, alternatives for privacy-preservation exchange such as those presented in this paper are of interest. In PET-Exchange, order books are instead public and encrypted, enabling more transparency while keeping traders' privacy.

III. PET-EXCHANGE MODEL

The trading system of an electronic securities exchange consists primarily of an exchange interacting with brokers, matching engines, and order books. Similarly, we implement these in PET-Exchange. In addition, our PET-Exchange model consists of an evaluator responsible for handling decryption keys and solving challenges. All homomorphic encryption operations use the CKKS [9] scheme as the exchange model must operate on floating point values. Figure 1 shows an architectural overview of PET-Exchange. At a high level, we have three component groups (colored parts), one for the broker, one containing the exchange, and one containing the evaluator. We next describe each component.

Brokers: Brokers send orders for a given instrument to the electronic securities exchange where instruments are traded. Brokers can either place orders for listed instruments directly or act as intermediaries for their clients. Examples of brokers include E*TRADE and Charles Schwab.

Exchange: An electronic exchange allows brokers to place orders on securities for the listed instruments, allowing order entry and forwarding, with the central function being matching and executing two orders [6], [7]. The two largest public securities exchanges are New York Stock Exchange and Nasdaq.

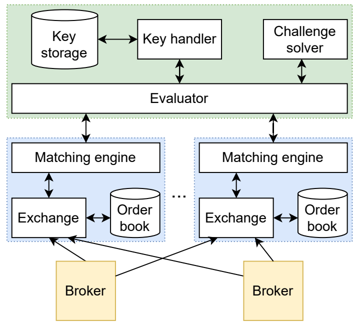


Fig. 1. Overview of PET-Exchange.

Order book: The electronic securities exchanges receive orders from brokers, which are inserted into order books connected to the order type and the instrument. Order books are a sorted list of buy and sell orders for a specific financial instrument. Typically, the order book is sorted based on the price point for the given instrument. However, additional requirements depending on the order type might alter the sorting of the order book [6]. In the electronic exchange, multiple order books can be used to keep track of performed orders and for active orders for traded instruments [6]. In a simple representation, an order book for active orders can be described as two columns B_i and A_i representing the active bid and ask orders. The exchange can then use the order book to determine the price at which a security is traded [6].

In PET-Exchange, an encrypted variant of the order book is used as the order price and volume are encrypted when inserted into the order book. The difference between a regular order book and the encrypted order book is that the properties identifying the order are no longer available to the exchange. Therefore, the trading price cannot be directly obtained from the order book. Instead, the matching engine uses the encrypted order book as a state holding the ordered queues of active bid and ask orders.

Matching engine: For each electronic securities exchange, there is at least one matching engine. The matching engines use different algorithms to match buy and sell orders in a securities exchange [6]. Orders are typically picked from the order book referring to a specific instrument. The engine matches orders that meet specific requirements. These requirements include limits on the price, volume, and any additional constraints that depend on the order type. The matching engine performs the matching logic between orders to clear or execute them. The orders are matched between an order to buy (*bid*) and an order to sell (*ask*), and the matching depends on the price and volume [6]. The exact fields included in order matching depend on the order types. Some types may exclude the price limit and instead define an order with a volume that should be cleared at the given market value of the instrument. In PET-Exchange, the matching engine supports the matching of encrypted orders, executing them entirely or partially, and does this without learning the order price or volume.

Evaluator: A shared evaluator is connected directly to the matching engines and also indirectly to the exchanges. The evaluator is a neutral party in the system and handles the key

TABLE I
SUMMARY OF NOTATION AND SUPPORTED OPERATIONS.

Notation/Operation	**Description**
I	Exchange instrument
P_k and S_k	Public and secret keys
θ	Encrypted order
θ^A and θ^B	Encrypted ask and bid order
θ_P and θ_V	Order price and volume
$\gamma(\theta)$	Padded order with padding γ
$C(\alpha,\ \beta)_i$	Challenge index i where α and β are two encrypted values
$P_k^I,\ S_k^I \leftarrow KeyGen(I)$	Generate public key P_k^I and secret key S_k^I for instrument I
$c \leftarrow EncField(p)$	Encryption of plaintext p
$p \leftarrow DecField(c)$	Decryption of ciphertext c
$c \leftarrow EncAdd(c_1,\ c_2)$	Addition of two ciphertexts c_1 and c_2
$c \leftarrow EncSub(c_1,\ c_2)$	Subtraction of two ciphertexts c_1 and c_2
$\lambda \leftarrow RandValue(i, j)$	Random value between i and j if specified
$z \leftarrow Compare(\alpha, \beta)$	Comparison of encrypted values α and β using local or remote comparison. λ = True if plaintext $\alpha < \beta$, else False
$\gamma(\theta) \leftarrow PadOrder(\theta,\ \gamma)$	Padding γ applied to an encrypted order θ
$([C(\alpha,\ \beta)_i;\ n],$ $[Boolean;\ n])$ $\leftarrow GenCha(C(\alpha,\ \beta),\ n)$	Generation of n challenges

pair of the public key P_k and secret key S_k. The evaluator (and its subcomponents) is the only component with access to the secret key. The evaluator is responsible for distributing the public keys for any instrument traded on the exchanges and can also determine the homomorphic operation results used when matching orders. While the evaluator has access to the secret keys, the component does not have to be fully trusted as the exchange only shares comparison values (and not order information) with the evaluator. In PET-Exchange, the evaluator interacts with two sub-components, the key handler, and the challenge solver.

Key handler & Key storage: The key handler component handles key generation and interacts with the key storage. For each traded instrument I, a key-pair with a public key P_k^I and a secret key S_k^I is stored and mapped on the key storage. Clients or brokers interacting with the exchange use the public key P_k^I for the instrument I to encrypt their orders before sending them to the exchange. Before encrypting, clients or brokers must first obtain P_k^I by querying the exchange. The exchange obtains the public key through the evaluator.

Challenger solver: A challenge solver is a logic unit part of the evaluator component and determines the result of a comparison operation repeated on a given set of challenges. An exchange creates challenges for two purposes: hiding the comparison of interest among a set of challenges and verifying the evaluator's behavior and execution. A challenge $C(\alpha,\ \beta)$ contains two encrypted values.

IV. ENCRYPTED ORDER HANDLING

We next describe the flow of order handling, starting with a client placing a trade order up to the order being matched and executed. At a high level, after an order is placed, the order entry (Section IV-A) is inserted into a (sorted) order book, leveraging a local or remote comparison function (Section IV-B), before orders are matched and executed (Section IV-C). For completeness and reproducibility, detailed algorithms are provided in Appendices A-D, and their implementations are part of the shared code.[1] A summary of notations and operations supported by PET-Exchange and used throughout this paper is given in Table I. Prior to the trading, we assume an evaluator to have generated the public key P_k^I for each instrument I, and that the exchange has distributed these to all clients and brokers encrypting and placing orders.

[1]https://github.com/wahl-sec/pet-exchange

A. Order Entry

A client initiates the trading process by placing an order at the broker using the public key P_k. As the order is encrypted, the order details are hidden from the broker and the exchange, and can only be decrypted by the evaluator. At the exchange, orders must be inserted such that the order book is sorted in terms of the bid or ask price. Here, we use binary search on encrypted order entries to find the correct position. Appendix A details the order entry algorithm (Algorithm 1).

B. Order Comparison

The comparison function in the order entry (Algorithm 1) uses either a local or a remote comparison method (seen from the exchange point of view). For the local comparison, the exchange compares two ciphertexts and queries the evaluator to decrypt and verify the solution. For the remote comparison, the exchange creates challenges to be solved by the evaluator and its subcomponents.

Local Comparison: In securities exchanges, comparing two values is vital for the double auction matching algorithm. However, if encrypted, they cannot be directly compared. Multiple methods for evaluating comparisons have been proposed for different schemes. In this work, we use the comparison process for the CKKS scheme as presented by Cheon et al. [10], [11], where we compare and identify the minimum and maximum of two encrypted values by approximating:

$$\lim_{m\to\infty} \frac{\max(a,b)^m}{a^m+b^m} = 1 \quad \text{and} \quad \lim_{m\to\infty} \frac{\min(a,b)^m}{a^m+b^m} = 1.$$

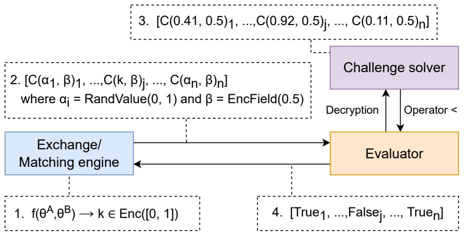
Because of the representation of CKKS encrypted values, the approximation is in the interval of $[0, 1]$ where a value near 0 implies that $a < b$, 0.5 implies that $a = b$, and 1 implies that $a > b$. Therefore, for the local comparison method, we compare two encrypted orders θ^A and θ^B by approximating:

$$k = f(\theta^A,\ \theta^B) \approx \begin{cases} 0, & \text{if } \theta^A < \theta^B \\ 0.5, & \text{if } \theta^A = \theta^B \\ 1, & \text{if } \theta^A > \theta^B, \end{cases}$$

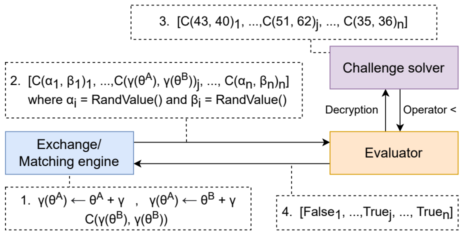
where the function is approximative and loosely follows the sigmoid function within the interval $[0, 1]$.

Since the process results in an encrypted approximation, the exchange must rely on the evaluator to decrypt the result using its secret key S_k. Therefore, the local comparison also includes an interactivity part between the exchange and the evaluator, querying the evaluator for the decrypted value.

However, to avoid revealing the comparison of interest to the evaluator, an exchange creates a challenge $C(k, 0.5)$ together with $n - 1$ additional random challenges $C(\text{RandValue}(0, 1), 0.5)$ encrypted using the same public key P_k. In addition to hiding the comparison of interest, these


3. [C(0.41, 0.5)1, ...,C(0.92, 0.5)j, ..., C(0.11, 0.5)n]
Challenge solver
2. [C(α1, β)1, ...,C(k, β)j, ..., C(αn, β)n]
where αi = RandValue(0, 1) and β = EncField(0.5)
Decryption
Operator <
Exchange/ Matching engine
Evaluator
1. f(θA,θB) → k ∈ Enc([0, 1])
4. [True1, ...,Falsej, ..., Truen]

(a) Local comparison


3. [C(43, 40)1, ...,C(51, 62)j, ..., C(35, 36)n]
Challenge solver
2. [C(α1, β1)1, ...,C(γ(θA), γ(θB))j, ..., C(αn, βn)n]
where αi = RandValue() and βi = RandValue()
Decryption
Operator <
Exchange/ Matching engine
Evaluator
1. γ(θA) ← θA + γ , γ(θA) ← θB + γ
C(γ(θB), γ(θB))
4. [False1, ...,Truej, ..., Truen]

(b) Remote comparison

Fig. 2. Overview of the challenge solving process.

additional challenges are later used for validation by the exchange. The challenge of interest is inserted at a random index j, where $0 \leq j \leq n$. The list of challenges $[C(\alpha,\beta)_1, ..., C(k,\beta)_j, ..., C(\alpha,\beta)_n]$, where α and β are encrypted values, is then sent to the evaluator, who is unable to learn the original order values θ^A and θ^B. As the challenges are sent without context, the information exposure is also limited, and the evaluator will only decrypt and compare approximation values between 0 and 1. Therefore, the approximation process effectively anonymizes the order information sent to the evaluator. This significantly limits the information available to the evaluator, who therefore does not have to be considered fully trusted. The challenge generation for the local comparison method is detailed in Appendix B (Algorithm 2a).

At the evaluator, the challenge solver decrypts each value, compares, and returns a list of comparison results (true and false values). Figure 2(a) summarizes the challenge solving process for the local comparison method. The challenge solver is detailed in Appendix C (Algorithm 3).

Using the comparison results, the exchange verifies the behavior of the evaluator by validating the response for the $n-1$ challenges. The validation uses the expected results from the challenge generation process and verifies that the results match. This is possible as the exchange has generated and has access to the unencrypted values. The challenge validation process is detailed in Appendix C (Algorithm 4). If all $n-1$ challenges are correctly validated, the returned response for the unknown challenge is also considered valid. If the response for any challenge does not match, the execution is halted as the evaluator cannot be trusted. In practice, different actions can be taken if the response is invalid. We note that the verifiability of the evaluator is based on the number of challenges. Since the evaluator does not know the challenge of interest, the same comparison logic must be applied to all challenges.

Remote Comparison: The remote comparison is similar to the local comparison, except that the approximate comparison is not utilized. Instead of sending comparison values (between 0 and 1) to the evaluator, the exchange sends encrypted order values to be compared remotely. There, the evaluator decrypts the values and returns which value is smallest. However, to avoid revealing the comparison of interest, the exchange uses additive homomorphic operations to apply a randomly generated padding γ on the encrypted ask order θ^A and bid order θ^B. The padded values $\gamma(\theta^A)$ and $\gamma(\theta^B)$ are then packed into a challenge $C(\gamma(\theta^A), \gamma(\theta^B))$. Similar to the local comparison method, the exchange generates $n-1$ other random challenges and inserts the challenge of interest at a random index. The list of n challenges is then sent to the evaluator.

For remote comparison, the challenge solving and validation follows the same process as the local comparison (Algorithms 3 and 4). Figure 2(b) summarizes the challenge solving process for the remote comparison method, and the challenge generation process is detailed in Appendix B (Algorithm 2b).

C. Order Matching and Execution

After inserting orders into order books, we are now ready to match and execute the orders. PET-Exchange matches and executes limit orders using the double auction algorithm, a general auction algorithm used extensively in electronic exchanges and other similar auctions where there are multiple buyers and multiple sellers.

We apply certain modifications to the original algorithm to enhance the privacy-preserving properties, including padding of order prices and volumes, challenge generation, and not disclosing the winning bid at the end of each iteration. The process of matching and executing orders in PET-Exchange is detailed in Appendix D (Algorithm 5). As the exchange and matching engine cannot access the price and volume of orders in cleartext, it uses the local or remote comparison functions to compare two values. By padding order prices and volumes using additive homomorphic operations $EncAdd(\theta, \gamma)$ for a randomly generated value γ, the matching engine can pass the encrypted value to the evaluator without revealing the actual order θ. The requirement on γ is that it should be large enough to hide a price of varying sizes inside the padding.

The matching algorithm uses the local or remote order comparison to determine the minimum volume $\theta_{\mathrm{V_{min}}}$ of two matched order volumes θ^A_V and θ^B_V. The minimum volume is then fully matched with either order θ^A_V or θ^B_V. For the order not being filled, the volume is deducted using homomorphic operation $EncSub(\theta_V, \theta_{\mathrm{V_{min}}})$, allowing the orders to be partially filled and continuously matched while preserving the privacy requirement on the volume.

V. Proof-of-Concept Evaluation

For our evaluation, we run the exchange, evaluator, and clients on different processes simultaneously on a single workstation using Ubuntu 20.04 with an Intel Core i7 9750H 2.6GHz CPU and 16GB RAM. For communication, we use the gRPC protocol implemented using the official gRPC framework [12]. Our implementation uses the Pyfhel [13] and SEAL library [14] for homomorphic encryption.

TABLE II
CKKS CONFIGURATION PARAMETERS AND THEIR OVERHEADS.

Name	Level	Poly. modulus degree	Security level	E[Order size]	E[Remote challenge size]	E[Local challenge size]
CKKS-11	1	2,048	128	66 KB	66 KB	N/A
CKKS-12	1	4,096	256	130 KB	131 KB	N/A
CKKS-14	6	16,384	128	3.15 MB	3.15 MB	2.10 MB
CKKS-15	6	32,768	256	6.29 MB	6.29 MB	4.19 MB
Plain	N/A	N/A	N/A	0.33 KB	0.18 KB	0.31 KB

A. Example Workload

To evaluate the proposed encrypted exchange and identify bottlenecks, we perform simulations of a trading session based on generated trading data. Here, we simulate short trading sessions that include the minimum needed to validate the order representation and glean insights into the system performance of an exchange that meets the minimum necessary functionality requirements for such exchange. We expect that most real exchanges would also include more conditions on orders, including fill-or-kill orders and other order types and conditions not currently supported by our example implementation.

As our default scenario, we evaluate the system, including order entries, using a trace of 200 randomly generated orders with a 50/50 split of buy and sell orders. For the tests, the order price is randomly generated between 10 and 90. This ensures that not all orders can be matched in the dataset and helps us validate the correctness of the matching. When padding, we use a similar distribution, effectively hiding the original order values and avoiding value overflow. Furthermore, the volume is randomly generated to better validate the ability to partially fill an order and to fit multiple buy orders in a larger sell order.

For the default performance tests, the orders are sent as quickly as possible; i.e., the client/broker sends the order and waits for an identifier for the order, and then the next order is sent from the list. Simultaneously as the clients send orders, the exchange accepts and matches orders, and the evaluator works on solving challenges. This scenario allows the entire system architecture to be evaluated simultaneously and provides a head-to-head comparison of the time spent by the different algorithms in a run-time scenario. We also run tests using a batch-based use case where all entries are inserted before being matched (Section V-F).

B. Ciphers and their Overheads

In our tests, we compare the performance of our implementations (with remote and local computation) when using both non-encrypted (plain) counterparts and challenges encrypted using homomorphic encryption. CKKS with different configurations (and multiplicative depth) are used for encrypted order handling. For the remote implementation, we are able to use CKKS-11, CKKS-12, CKKS-14, and CKKS-15, where the number represents the polynomial modulus degree of the ciphertext. For example, CKKS-11 is the CKKS configuration with a degree of 2^{11}. However, for the local comparison methods, we require a multiplicative depth (level) of 6 (due to the approximation algorithm described in Section IV-B), limiting us to CKKS-14 and CKKS-15. (We therefore do not report any results for CKKS-11 and CKKS-12 with the local computation implementation.) In our experiments, we use a CKKS scaling factor of 2^{45}, a first modulus prime of size 52 bits, and subsequent primes in the modulus chain of 45 bits.

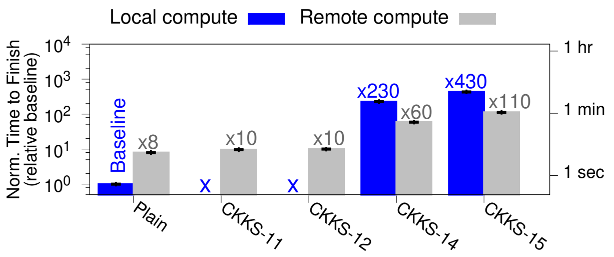


Fig. 3. Increase in time to finish (TTF) relative just doing the non-encrypted (i.e., plain) comparisons locally.

Table II summarizes the parameters used, their bit security, and the encrypted order (and challenge) sizes needed in different settings. With the order sizes being proportional in size to the degree, and challenges typically of a similar order of magnitude, using the larger degrees can therefore result in substantial overheads (proportional to their relative degree differences). While not shown explicitly in this table, it is also important to note that the use of additional challenges is multiplicative. For example, with one extra challenge (on average) per regular challenge, the communication overhead (measured in bytes) roughly doubles.

C. Performance Metrics

For our performance evaluation, we measure the time the trading system and the responsible components spend on different tasks. Here, we simulate a trading session with various configurations and report the following timing metrics.

Time-to-Finish (TTF): The TTF metric measures the total runtime for the trading session, from the time when all components are started until the end of the final matching round. This includes clients encrypting and sending orders, and the exchange receiving, inserting, and matching orders.

Time-to-Match (TTM): The TTM metric measures the time spent matching orders in the double auction matching algorithm (Algorithm 5). The metric captures the time for two orders to be compared by volume and price, and to execute the order if the two complementing orders can be matched.

Time-to-Insert (TTI): The TTI metric measures the time spent inserting an incoming order into the order book relevant to the instrument traded. This metric captures the performance of our binary search inserting algorithm (Algorithm 1), and the time to insert orders into the order book.

Time-to-Solve-Challenges (TTSC): The TTSC metric measures the time spent solving the challenges on the evaluator (Algorithm 3). The metric captures how long it takes to solve the challenges created by the exchange.

D. Local vs. Remote Computations

Figure 3 shows a performance comparison of the relative increase in time-to-finish (TTF) when using different CKKS configurations (with increasing degree, size, and multiplicative depth) for both the local and the remote computing approach. Here, all times are normalized (first y-axis) relative to using a non-encrypted (plain) implementation that does everything locally (which we give a normalized TTF of 1). In the figure, we include confidence intervals (very tight on log scale) and explicitly write out the relative increase in TTF for each case.

We note that the remote computation scheme is both faster and allows the use of smaller encryption configurations. For example, with CKKS-11 and CKKS-12, the increase in TTF (10 times) is very similar to using remote plain-text challenges (8-time increase). In contrast, the local-computing scheme requires the use of the larger CKKS configurations with bigger multiplicative depth (i.e., CKKS-14 and CKKS-15) and is therefore not able to achieve as fast TTFs. Using more homomorphic operations, the local computations are also more time consuming for these larger ciphers. For example, with CKKS-14, the TTF increases by 230 vs. 60 times, and with CKKS-15, the TTF increases by 430 vs. 110 times when using the local- vs. the remote-computation scheme.

The small increase in the TTF when comparing the smaller remote comparisons (with CKKS-11 and CKKS-12) with the plain remote comparison is encouraging as it shows that the added bandwidth (bigger challenges) and added cost due to their homomorphic operations are relatively small compared to other network delays and other order insertion costs.

The potential downside with the remote-computing scheme is that the client needs to rely on the evaluator to decrypt the challenges. While these challenges are padded (here using additions, but it is also possible to use multiplication and other operations that preserve which of the bids is greater), this gives the evaluator a bigger role and may leak somewhat more information (e.g., the absolute differences in the bids in the case that only addition is used). A broker may therefore be further encouraged to submit more additional "dummy" challenges when using the remote-computing scheme. Yet, the relatively small increases observed when using CKKS-11 and CKKS-12 (e.g., 10 times bigger than if just making the plain comparisons locally) suggest that this scheme may be feasible for small exchange scenarios with smaller order books.

When moving to the bigger configurations, additional compute power than used in our proof-of-concept experiments would be needed as the completion rates instead increase by a factor of 230 (430) or 60 (110) when using the local and remote computing scheme, respectively, with CKKS-14 (and CKKS-15). While the bigger ciphers also add network bandwidth overhead (e.g., by a factor of 16 going from CKKS-11 to CKKS-15), we expect the bandwidth to be more easily scaled than the processing hardware, and that the size of a CKKS-15 challenge (Table II) is of a similar size as a modern website (i.e., a few MB). As long as an exchange is willing to invest in compute power for local-computing, we therefore do not count out the use of the local-computing scheme.

E. Impact of Number of Challenges

As argued earlier, brokers may want to submit additional dummy challenges for the purposes of (1) hiding the comparison of interest, and (2) verifying the behavior and execution of the evaluator. Here, we expect that very few additional challenges would be needed to keep the third party honest. To better understand how much the use of additional challenges adds to the total completion times, Figure 4 shows the relative increase (as a factor) of the time to finish (TTF) when using 1, 5, or 10 additional challenges per "real" challenge. Here, all ratios are normalized compared to the base case when not submitting any additional challenges using the same scheme. We note that the increase for the local computations is very limited, whereas the TTFs using the remote computation scheme increases substantially, and in the cases of CKKS-14 and CKKS-15 even increase by a bigger factor than the number of additional challenges. This is due to the queue buildups contributing to a non-linear increase in the processing times.

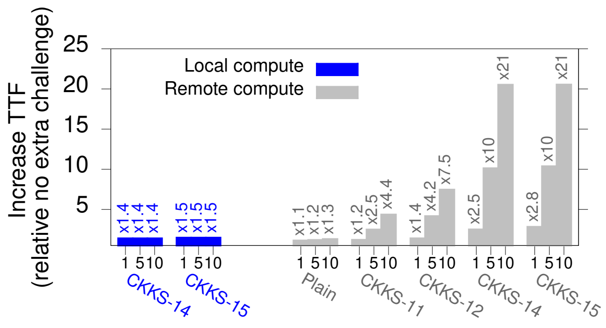


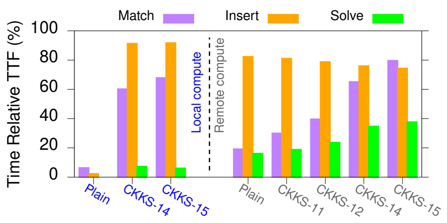
Fig. 4. Increase in time to finish (TTF) when adding N additional challenges to every challenge. Here, it should be noted that we expect that very few additional challenges would be needed to keep the third party honest.

F. Bottleneck Analysis and Batching

We next look closer at the different configuration bottlenecks. For each configuration, Figure 5 shows example results for the cases when the broker (a) submits no additional challenges and (b) five additional challenges per "real" challenge.

The main observation here is that the time to insert (TTI) (processing time of Algorithm 1) appears to be the main bottleneck. Regardless of the number of extra challenges submitted, the insertion process is consistently active for more than 74% of the time when using one of the CKKS configurations. Interesting future work therefore include the design of improved insertion algorithms that either pushes the time complexity below $\mathcal{O}(\log n)$, i.e., the complexity of our simple insertion algorithm, or the design of exchanges that incorporates the insertion delays into the design itself (e.g., through use of batching, periodic deadlines, or other mechanisms that help effectively maintain sorted order books).

Also, the time to match (TTM) (processing time of Algorithm 5) is substantial, especially for the larger ciphers. In our tests, both these components consistently (for all configurations) consumed more time than the challenge solver (Algorithm 3). Even in the case where the challenge solver consumed its biggest fraction (remote computing with CKKS-15), it consumed less than 38% when no additional challenges were submitted (Figure 5(a)) and less than 54% with five additional challenges (Figure 5(b)). As expected, we observed


(a) No extra challenges

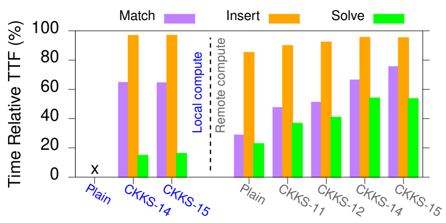

(b) Five extra challenges per challenge

Fig. 5. Relative time consumed on different tasks (executed in parallel) relative to the overall time to finish (TTF).

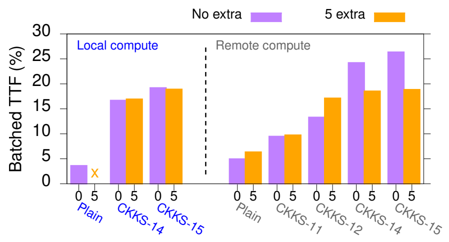


Fig. 6. Relative time to finish (TTF) for batched workload use case compared to the default use case.

TABLE III
AVERAGE DEVIATION FROM EXPECTED VALUE PER CONFIGURATION AND DECIMAL PRECISION.

Dec.	Plain	CKKS-11	CKKS-12	CKKS-14	CKKS-15
≤ 8	0	0	0	0	0
9	0	10^{-7}	10^{-5}	10^{-7}	0
10	0	$1.08 \cdot 10^{-6}$	$2.11 \cdot 10^{-8}$	$1.14 \cdot 10^{-8}$	0
11	0	$4.81 \cdot 10^{-8}$	$3.14 \cdot 10^{-8}$	$1.11 \cdot 10^{-8}$	0
12	0	$1.59 \cdot 10^{-6}$	$3.12 \cdot 10^{-8}$	$2.19 \cdot 10^{-9}$	0
13	0	$1.61 \cdot 10^{-8}$	$2.30 \cdot 10^{-8}$	$1.52 \cdot 10^{-9}$	0
14	10^{-11}	$1.11 \cdot 10^{-8}$	$2.83 \cdot 10^{-8}$	$4.44 \cdot 10^{-9}$	10^{-11}
15	10^{-11}	$1.46 \cdot 10^{-8}$	$1.97 \cdot 10^{-8}$	$6.51 \cdot 10^{-9}$	10^{-11}

a noticeable increase in the percentages when adding five additional "dummy" challenges per "real" challenge. We also note that the challenge solver overheads (as expected) consistently are smaller for the local vs. the remote-computation scheme.

Batch-based workload: To glean some insights into the benefits of removing the insertion bottleneck, we also run tests where we first insert all orders and the exchange only requires one iteration to match orders. This can be seen as an example where an exchange collects orders (e.g., over the night) and then process all orders at once. While this use case scenario removes the insertion bottleneck, it results in more orders needing to be handled simultaneously by the matching algorithm. Our results confirm a significant reduction in the time to finish (TTF) metric for these experiments. This is illustrated in Figure 6, which shows the relative TTF for these batched processing as the percentage of the TTF for the corresponding default scenario (i.e., same orders but clients submit them as fast as possible when the exchange opens). Substantial savings are observed, with all cases resulting in reductions of more than 74%. We also note that the biggest savings (i.e., smallest values shown here) are for the lighter ciphers (e.g., CKKS-11 with the remote computing), seeing a reduction of 91%.

G. *Precision and Accuracy*

To ensure correctness, operations must be performed with sufficient precision and accuracy. Through experiments, we find that the different configurations achieve full accuracy up to different precisions and that the accuracy may differ when operating with too many decimals precision. Table III presents example results from running the exchange with different decimals precision in the order price and volume. Here, we varied the maximum decimal precision allowed for the exchange from 0 to 15 and report the average deviation from the expected values. The results show that all CKKS configurations achieve perfect accuracy up to a precision of 8 decimal places. Furthermore, we observe increasing accuracy for increasing ciphertext sizes, with CKKS-15 achieving perfect accuracy when executing floating point values up-to-and-including 13 decimal places. In fact, with CKKS-15, we achieved the same accuracy as with the plain (non-encrypted).

These results show that the solution supports sufficient precision for the needs of modern markets. While certain markets might require additional precision, we deem the precision sufficient for the market type we have considered. These results also further highlight that there is a tradeoff between execution accuracy (with high precision) and the processing times (as the larger ciphers require more processing) and byte overhead (as the order and challenge sizes become bigger).

For the case that an exchange would select to operate in an approximate region (e.g., use CKKS-11 with 9 decimals of precision), we note that there may be some loss in the accuracy (e.g., by 10^{-7}). In this case, some (minor) incorrectness in prices and volumes may impact the index of a given security and the volumes being traded. While these smaller inaccuracies could likely be easily corrected with some delay (after executed orders are public), we expect that any exchange that would like to operate with more decimals of precision to also be willing to invest in the extra computing power that may be required when using, for example, CKKS-15.

VI. DISCUSSION

We note that the privacy-preserving properties of the proposed solution depend on the amount of encrypted order information. Our implementation and evaluation show the possibility of an exchange revealing only the instrument and order type, preserving the trader's privacy. However, the privacy-preserving properties rely heavily on the decryption on the evaluator. We next discuss further steps that can be taken to strengthen the system, and outline ethical considerations.

A. *Confidentiality and Evaluator Behavior*

The system design relies on the confidentiality of the encrypted exchange. Here, the evaluator plays a central role, being the only party with secret key access. Furthermore, both the local-computing scheme and the remote-computing scheme require the external evaluator to use these secret keys when comparing bids.

To avoid revealing the actual values to the evaluator, the remote-computation scheme relies on padding. Here, we use additive padding with the CKKS scheme (which still may allow the evaluator to learn something about the absolute difference between bids), and the values are sent to the evaluator without context (e.g., order identifiers, instrument, and order type), limiting the exposure of information to the evaluator decrypting the values. For further protection, it is possible to combine additive padding with multiplicative padding (which also preserves which bid is bigger), for example. This motivates the use of the CKKS scheme (capable of both addition and multiplication), rather than other additively homomorphic encryption schemes such as those proposed by Benaloh [15], Paillier [16], or Boneh et al. [17]. For the local-computing scheme, the exchange only shares the output of a comparison algorithm (a value between 0 and 1). Without knowledge about which bidders are compared (not shared), the evaluator cannot learn any valuable information from the decrypted ciphertext.

As noted earlier, additional "dummy" challenges can be used both to keep the evaluator honest (by continually evaluating the correct behavior) and to hide the true bid comparisons in the set of all challenges sent to the evaluator. Here, the exchange can use historic data, for example, as a way to better obscure the "true" challenges from the extra "dummy" challenges. In this paper, we do not consider such optimizations.

We also do not consider how to best protect against an adversary evaluator. In practice, we expect an evaluator to be a trusted authority (at least of similar trust level as certificate authorities in the certificate landscape or the major exchanges in today's financial landscape) and that their behavior would be closely monitored. We foresee a simple monitoring and trust system to be sufficient in most scenarios. However, if even greater transparency is desired, the system design could easily be extended to incorporate transparent order logs (similar to certificate transparency logs) and use signed challenges and responses. By logging signed challenges and responses, the exchanges would have proof of any wrongful answers.

B. Improved Trust in the Evaluator

Another potential way to improve the confidentiality would be to use homomorphic comparisons that do not require decryption at the evaluator. However, at the time of the writing, no efficient method for such operations currently exists. Yet, our exchange design provides an interesting use case for efficient homomorphic value comparisons. Assuming the eventual introduction of such solutions, the confidentiality could be improved by moving the evaluation comparison to the exchange level. The integrity of the evaluations can also be improved through the use of trusted execution environment (TEE). Combining homomorphic encryption with TEE was proposed by Drucker et al. [18] as a solution to localize the execution of homomorphically encrypted data and, as such, protect against unauthorized operations on the encrypted data. By only performing operations such as decryption and evaluation of the challenges in a TEE, the risk of malicious actors manipulating the evaluation mechanism is reduced.

Using TEEs allows for new design possibilities where the evaluator functionality can be moved to the exchanges while limiting secret key exposure to within the TEE. Such a design could help reduce communication latency and bandwidth usage between the exchange and evaluator. Further design improvements include hiding the trading instrument, or whether orders are buy or sell orders (e.g., as proposed by Gama et al. [19]). Such improvements can, for example, encrypt the order direction or allow traders to submit a pair of buy and sell orders, where one of the orders has zero volume. This further improves privacy at the cost of additional overheads.

C. Control of Evaluator

The encrypted exchange proposed here requires an evaluator component. While designed not to require any context about the data, the evaluator still requires some information to be disclosed (via decryption on the evaluator). While we do not provide any answer to who should control the evaluator due to the high regulatory environment of the securities market, we expect that regulatory institutions would be the most likely candidate. Another option is to form collaborations between multiple market operators. Furthermore, to reduce the risk of a single point of failure, we also foresee a solution where multiple evaluators work together and distribute the workload.

D. Reduced Trust in the Exchange

In our analysis, we have assumed a semi-trusted honest-but-curious exchange. However, we note that the exchange's trust can also be reduced. One potential solution is that brokers sign encrypted orders to provide data integrity. The evaluator can then verify the signatures prior to comparing decrypted order values. This can be implemented using homomorphic signatures [20] where the untrusted exchange can perform computations on encrypted and signed data, deriving a short signature certifying the correct computation output. The evaluator can then verify the correctness without retrieving the underlying data. To avoid revealing the signed comparison of interest, the exchange would be restricted to using past order challenges instead of self-generated orders. By systematically rotating "dummy" challenges, the exchange would be unable to learn the comparison of interest.

More concretely, θ_A and θ_B would be encrypted orders signed by brokers A and B. In the local comparison process, we calculate the encrypted value $k \in \{0, 1\}$, where k is short-signed by brokers A and B. This is packed into a challenge $C(k, Enc(0.5))$ and sent to the evaluator for comparison, who can check the request validity. Similarly, with remote comparison, adding padding γ to an encrypted and signed order θ results in a padded order that is short-signed.

While collusion would be possible between exchanges, brokers, or the evaluator, we do not expect this to be a major concern due to the signatures' non-repudiation property. As the entities are considered semi-trusted with authority to handle securities, any manipulation attempts can easily be tracked, keeping entities legally accountable. Further accountability could be achieved by introducing transparency logs or similar.

Here, we consider collusion to be out of scope, but note that the framework can easily be extended to include signatures.

E. Financial Crime and Ethical Considerations

The encrypted exchange proposed here limits the ability to view and act on information from non-published limit orders, reducing the risks of insider trading in financial institutes. Our solution allows an exchange to execute entire trading sessions without publishing order matchings until after the session, further limiting the information available to malicious actors to only incoming order counts for each instrument. In general, the use of encryption slows down the trading and prevents advanced knowledge of bids. This prevents insider trading attacks based on large-volume bids or penny jumping [2].

On the other hand, many existing solutions for preventing financial crime, including anti-money laundering and trade surveillance solutions like Nasdaq SMARTS [21], rely on real-time monitoring and pattern recognition. Such solutions are hampered or slowed down by the use of encryption. Furthermore, even without knowledge about the secret key, an attacker can still derive some knowledge from the available plaintext information. Overall, careful consideration must be given to potential regulatory requirements for trading surveillance. The desirable exchange choice and encryption level may therefore differ between exchanges and markets. Here, we present an example solution, evaluate a proof-of-concept implementation, and do not run the exchange with live data.

VII. Related Work

Cryptographic Exchange Models: Cryptographic solutions have previously been considered in exchanges and trading. Early work by Thorpe and Parkes [7] presents a protocol implementing cryptographic primitives to limit the information exposure in trading using homomorphic encryption and a partially transparent order book. In a follow-up study, Thorpe and Willis [22] present rule-based cryptographic trading, an alternative to dark pool trading, showing that malicious trading can be prevented by executing trades only if some conditions are fulfilled. Both works apply early versions of homomorphic encryption on an exchange and suffer from the performance and usability aspects. For the last decade, many improvements have been achieved in the homomorphic encryption field. In our work, we expand on the idea of an encrypted exchange and present an updated view. In contrast to prior work, we focus on limit orders and are able to partially match encrypted orders.

Privacy Enhanced Auctions: Prior works have focused on executing orders on auctions while limiting or not revealing the information about bids until a trade is executed. Cachin [23] includes a third party oblivious to the auction state and use homomorphic encryption to design a secure bid auction. Baudron et al. [24] study the use of a semi-trusted third party where clients encrypt their bids using all participants' public keys and send the resulting ciphertext to the auctioneer to determine the winning bid using a homomorphic encryption scheme. Balch et al. [25] use homomorphic encryption to perform inventory matching of buyers and sellers, skipping the steps of the continuous double auction in a public exchange where both prices and volumes are matched. Ngo et al. [26] present a new cryptographic scheme, focusing on key agreement and privacy aspects in blockchain-based dark pools.

In this paper, the proposed privacy enhanced trading framework is similar in terms of using a third party (evaluator) with secret key access. However, the exchange does not assume a fully trusted evaluator but instead shares only the numeric comparison of interest, padded using homomorphic encryption, and use additional challenges to verify the evaluator's behavior. By ensuring that the evaluator cannot access the original order information, it does not have to be fully trusted. We focus on non-revealing continuous double auction matching, addressing privacy issues on public exchanges.

Another approach for improving confidentiality and privacy in trading is to use secure multiparty computation (MPC) [27]–[30]. For example, Bogetoft et al. [29] present the first large-scale practical application of MPC and apply it in double auction. Asharov et al. [27] combine MPC with homomorphic encryption to improve the privacy of trading in dark pools.

Value Padding: Padding is the process of adding a value (often randomized), allowing a requester to pass on the padded value to an untrusted environment without revealing the initial value. In PET-Exchange, we use padding to hide the original value when comparing at the evaluator. Similar ideas of padding have been applied in previous works, covering a broad range of topics like the privacy of consumer IoT devices [31] and video streaming [32]. These example works highlight the tradeoff between the attack accuracy and the performance [31], and between the attack accuracy and the bandwidth overheads [32], respectively.

Performance Evaluation of Trading Networks: The Nasdaq electronic securities exchange handles tens of millions of trades every market day, totaling billions of shares in trading volume [33]. To keep up with the increasing volume, exchanges must operate and match trades quickly. Gama et al. [19] present and evaluate a new efficient volume matching protocol for the dark pools, demonstrating significant improvements in throughput. Homomorphic encryption has shown significance in multiple areas. However, using such cryptographic methods to enable confidentiality often results in performance overhead. Gao et al. [34] use the Paillier homomorphic scheme [16] to determine winners in auctions and present a performance analysis by simulating auctions with varying amounts of order entries, highlighting improvement possibilities using enhanced hardware.

VIII. Conclusions

We present PET-Exchange, the first privacy-preserving framework for trading securities that uses homomorphic encryption to effectively protect limit orders. Using CKKS to preserve order privacy, we implement a non-revealing continuous double auction matching of prices and volumes. In addition to standard components in electronic securities exchanges, we include an evaluator responsible for handling decryption keys and solving challenges. By splitting responsibilities and limiting the information visible to the evaluator and exchanges, we place limited trust on any one component.

Our proof-of-concept implementation is validated using simulations of generated order data, with performance results providing insights into the tradeoffs of approximate encrypted comparisons on the exchange or exact plaintext comparisons on the evaluator, as well as into which CKKS configurations provide the most attractive overhead-accuracy tradeoffs.

We also show that some proposed methods only result in a small increase in the total time to finish (TTF) when processing an example workload, compared to using local order matching on plaintext orders. For example, if willing to share padded orders with the evaluator (as with our remote-computing scheme), we can use smaller cipher configurations (e.g., CKKS-11, CKKS-12) that only increase the TTF by 10 times compared to a plaintext solution doing local comparisons at the exchange. If wanting to avoid sharing padded bid-pairs with the evaluator, but instead using the local-compute scheme, we must use the CKKS-14 and CKKS-15 ciphers (with greater multiplicative depth). For these, we observe another order-of-magnitude increase in the TTF (230 and 430 times, respectively, compared to the local plaintext baseline). Here, we expect much of this increase to be possible to offset by simply allocating a greater pool of computing resources.

Our experiments also show that the additional challenges used for (1) hiding the comparison of interest, and (2) verifying the evaluator's behavior come with significant overhead. This suggests that such "dummy" challenges should not be used more than needed. In practice, we do not expect that such "dummy" challenges would need to consume a significant portion of the challenges. In Section 6, we also discussed additional mechanisms to improve the trust in the evaluators, increase confidentiality, and otherwise improve the system.

Our evaluation indicates that the insertion algorithm is the current bottleneck, opening up for further research on how to best implement an insertion algorithm for encrypted bids or how to incorporate these delays into the exchange design. The results also confirm that our implementation using CKKS achieves high accuracy and precision.

Overall, PET-Exchange achieves the desired objectives without sacrificing accuracy, highlighting the potential of preventing unfair trading practices in electronic securities exchanges. Based on the results, we expect the most benefits to be seen by smaller markets with fewer orders (e.g., private auctions and smaller dark pools) and for which the local-computation scheme could be effectively used to ensure that the evaluator would have a very difficult time deriving insights about the originally compared values. Finally, by sharing the open-source tool, we hope to encourage further research applying homomorphic encryption to improve the privacy in auctions and other related financial settings.

Acknowledgment

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

[1] U.S. Securities and Exchange Commission, "Detecting personal trading abuses," https://www.sec.gov/rules/other/f4-433/mccann1.htm, 2000.

[2] S. Mahmoodzadeh and R. Gencay, "Human vs. high-frequency traders, penny jumping, and tick size," *Journal of Banking & Finance*, 2017.

[3] U.S. Securities and Exchange Commission, "Insider trading policy," https://www.sec.gov/Archives/edgar/data/1164964/000101968715004168/globalfuture_8k-ex9904.htm, 2015.

[4] ——, "Shedding light on dark pools," https://www.sec.gov/news/statement/shedding-light-dark-pools, 2015.

[5] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. ACM Symposium on Theory of Computing (STOC)*, 2009.

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

[7] C. Thorpe and D. C. Parkes, "Cryptographic securities exchanges," in *Proc. Financial Cryptography and Data Security (FC)*, 2007.

[8] B. Zambonin and D. Martins, "What dark pool trading volume says about AMC stock," https://www.thestreet.com/memestocks/amc/what-dark-pool-trading-volume-says-about-amc-stock, 2021.

[9] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proc. ASIACRYPT*, 2017.

[10] J. H. Cheon, D. Kim, and D. Kim, "Efficient homomorphic comparison methods with optimal complexity," in *Proc. ASIACRYPT*, 2020.

[11] J. H. Cheon, D. Kim, D. Kim *et al.*, "Numerical method for comparison on homomorphically encrypted numbers," in *Proc. ASIACRYPT*, 2019.

[12] Google, "gRPC," https://grpc.io/docs/what-is-grpc/introduction/, 2023.

[13] A. Ibarrondo and A. Viand, "Pyfhel: Python for homomorphic encryption libraries," in *Proc. ACM CCS WAHC*, 2021.

[14] Microsoft, "SEAL," https://github.com/Microsoft/SEAL, 2023.

[15] J. Benaloh, "Dense probabilistic encryption," in *Proc. SAC*, 1994.

[16] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. EUROCRYPT*, 1999.

[17] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-DNF formulas on ciphertexts," in *Proc. Theory of Cryptography (TCC)*, 2005.

[18] N. Drucker and S. Gueron, "Combining homomorphic encryption with trusted execution environment: A demonstration with Paillier encryption and SGX," in *Proc. ACM CCS MIST*, 2017.

[19] M. B. da Gama, J. Cartlidge, A. Polychroniadou, N. P. Smart, and Y. T. Alaoui, "Kicking-the-bucket: Fast privacy-preserving trading using buckets," in *Proc. FC*, 2022.

[20] S. Gorbunov, V. Vaikuntanathan, and D. Wichs, "Leveled fully homomorphic signatures from standard lattices," in *Proc. ACM STOC*, 2015.

[21] Nasdaq, "Nasdaq trade surveillance (SMARTS)," https://www.nasdaq.com/solutions/nasdaq-trade-surveillance, 2023.

[22] C. Thorpe and S. R. Willis, "Cryptographic rule-based trading," in *Proc. Financial Cryptography and Data Security (FC)*, 2012.

[23] C. Cachin, "Efficient private bidding and auctions with an oblivious third party," in *Proc. ACM CCS*, 1999.

[24] O. Baudron and J. Stern, "Non-interactive private auctions," in *Proc. Financial Cryptography (FC)*, 2001.

[25] T. Balch, B. E. Diamond, and A. Polychroniadou, "SecretMatch: Inventory matching from fully homomorphic encryption," in *Proc. AI in Finance (ICAIF)*, 2020.

[26] C. N. Ngo, F. Massacci, F. Kerschbaum, and J. Williams, "Practical witness-key-agreement for blockchain-based dark pools financial trading," in *Proc. Financial Cryptography and Data Security (FC)*, 2021.

[27] G. Asharov, T. Hybinette Balch, A. Polychroniadou, and M. Veloso, "Privacy-preserving dark pools," in *Proc. AAMAS*, 2020.

[28] D. Bogdanov, R. Talviste, and J. Willemson, "Deploying secure multi-party computation for financial data analysis," in *Proc. FC*, 2012.

[29] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler *et al.*, "Secure multiparty computation goes live," in *Proc. FC*, 2009.

[30] E. Makri, D. Rotaru, F. Vercauteren, and S. Wagh, "Rabbit: Efficient comparison for secure multi-party computation," in *Proc. FC*, 2021.

[31] A. J. Pinheiro, J. M. Bezerra, and D. R. Campelo, "Packet padding for improving privacy in consumer IoT," in *Proc. IEEE ISCC*, 2018.

[32] D. Hasselquist, C. Vestlund, N. Johansson, and N. Carlsson, "Twitch chat fingerprinting," in *Proc. IFIP TMA*, 2022.

[33] Nasdaq, "Daily market summary," https://www.nasdaqtrader.com/Trader.aspx?id=DailyMarketSummary, 2023.

[34] W. Gao, W. Yu, F. Liang, W. G. Hatcher, and C. Lu, "Privacy-preserving auction for big data trading using homomorphic encryption," *IEEE Trans. Network Science and Engineering (TNSE)*, 2018.

Appendix

A. Order Entry with Binary Search

Algorithm 1 shows the binary search for encrypted order entry. We note that the algorithm describes the process of sorting the orders in an ascending order applied to the queue of ask orders. For bid orders, we instead sort in descending order by reversing the comparison parameters. The *Length* and *Floor* functions are part of the standard library, returning the array length and rounding down a number, respectively.

B. Challenge Generation

Algorithms 2a and 2b show the challenge generation for the local and remote comparison process, respectively.

C. Challenge Solver & Validation

Algorithms 3 and 4 show the challenge solver and challenge validation process in PET-Exchange, respectively.

D. Matching Algorithm

Algorithm 5 shows the double auction matching algorithm used to continuously match and execute encrypted limit orders.

Algorithm 1 Order Entry with Binary Search

```
Require: $\theta$, $book$                ▷ Where $\theta$ is the encrypted order to insert
 1:                                         and $book$ is the sorted order book
 2: $lo \leftarrow 0$
 3: $hi \leftarrow Length(book)$
 4: while $lo < hi$ do
 5:     $mid \leftarrow Floor((lo + hi)/2)$
 6:     if $Compare(book[mid], \theta)$ then
 7:         $lo \leftarrow mid + 1$
 8:     else
 9:         $hi \leftarrow mid$
10:     end if
11: end while
12: return $lo$                          ▷ Return index to insert encrypted order
```

Algorithm 3 Challenge Solver

```
Require: $n \geq 0$, $challenges = [C(\alpha_1, \beta_1)_1, ..., C(\alpha_n, \beta_n)_n]$
 1: $results \leftarrow [false; n]$                 ▷ Result array of size $n$
 2: $i \leftarrow 1$
 3: while $i \leq n$ do
 4:     $a \leftarrow DecField(\alpha_i)$
 5:     $b \leftarrow DecField(\beta_i)$
 6:     $results[i] \leftarrow (a < b)$
 7:     $i \leftarrow i + 1$
 8: end while
 9: return $results$                  ▷ Result array for each comparison
```

Algorithm 4 Challenge Validation

```
Require: $results = [Boolean; n]$, $expected = [Boolean; n]$, $n \geq 0$
 1:                      ▷ Challenge results and expected results of size $n$
 2: $result \leftarrow NULL$
 3: $i \leftarrow 0$
 4: while $i < n$ do                         ▷ Verify all known challenges
 5:     if $expected[i] \neq NULL$ then
 6:         if $expected[i] \neq results[i]$ then
 7:             ...                           ▷ Handling of wrong result
 8:         end if
 9:     else
10:         $result \leftarrow results[i]$
11:     end if
12:     $i \leftarrow i + 1$
13: end while
14: return $result$
```

Algorithm 2a Challenge Generation (local comparison)

```
Require: $k, n \geq 0$              ▷ Where $k$ is result from approximation
 1:                                 function, $n$ is number of challenges
 2: $challenges \leftarrow [0; n]$          ▷ Challenge array of size $n$
 3: $expected \leftarrow [false; n]$     ▷ Expected challenge results array
 4: $j \leftarrow RandValue(0, n)$     ▷ Random index for unknown challenge
 5: $\beta \leftarrow EncField(0.5)$
 6: $challenges[j] \leftarrow C(EncField(k), \beta)$
 7: $expected[j] \leftarrow NULL$            ▷ Mark the unknown challenge
 8: $i \leftarrow 0$
 9: while $i < n$ do                ▷ Insert $n - 1$ other random challenges
10:     if $i \neq j$ then
11:         $a \leftarrow RandValue(0, 1)$
12:         $expected[i] \leftarrow (a < 0.5)$
13:         $challenges[i] \leftarrow C(EncField(a), \beta)$
14:     end if
15:     $i \leftarrow i + 1$
16: end while
17: return $challenges, expected$
```

Algorithm 2b Challenge Generation (remote comparison)

```
Require: $\theta^A, \theta^B, n \geq 0$
 1: $challenges \leftarrow [0; n]$          ▷ Challenge array of size $n$
 2: $expected \leftarrow [false; n]$     ▷ Expected challenge results array
 3: $\gamma \leftarrow RandValue()$                        ▷ Random padding
 4: $\gamma(\theta^A) \leftarrow PadOrder(\theta^A, \gamma)$
 5: $\gamma(\theta^B) \leftarrow PadOrder(\theta^B, \gamma)$
 6: $j \leftarrow RandValue(0, n)$     ▷ Random index for unknown challenge
 7: $challenges[j] \leftarrow C(\gamma(\theta^A), \gamma(\theta^B))$
 8: $expected[j] \leftarrow NULL$            ▷ Mark the unknown challenge
 9: $i \leftarrow 0$
10: while $i < n$ do                ▷ Insert $n - 1$ other random challenges
11:     if $i \neq j$ then
12:         $a \leftarrow RandValue()$
13:         $b \leftarrow RandValue()$
14:         $expected[i] \leftarrow (a < b)$
15:         $challenges[i] \leftarrow C(EncField(a), EncField(b))$
16:     end if
17:     $i \leftarrow i + 1$
18: end while
19: return $challenges, expected$
```

Algorithm 5 Matching Algorithm

```
Require: $\theta^A$, $\theta^B$
 1: $\theta_P^A, \theta_V^A \leftarrow \theta^A$
 2: $\theta_P^B, \theta_V^B \leftarrow \theta^B$
 3: $\gamma \leftarrow RandValue()$                        ▷ Random padding
 4: $\gamma(\theta_P^A), \leftarrow EncAdd(\theta_P^A, \gamma)$
 5: $\gamma(\theta_P^B) \leftarrow EncAdd(\theta_P^B, \gamma)$
 6: $result \leftarrow Compare(\gamma(\theta_P^A), \gamma(\theta_P^B))$   ▷ Compare price using local
 7:                                     or remote comparison method
 8: if not $result$ then
 9:     return              ▷ No more matches as the highest bid price
10:                                   is lower than lowest ask price
11: end if
12: $\gamma \leftarrow RandValue()$
13: $\gamma(\theta_V^A) \leftarrow EncAdd(\theta_V^A, \gamma)$
14: $\gamma(\theta_V^B)) \leftarrow EncAdd(\theta_V^B, \gamma)$
15: $result \leftarrow Compare(\gamma(\theta_V^A), \gamma(\theta_V^B))$   ▷ Compare volume using
16:                                     local or remote comparison
17: $\theta_V \leftarrow NULL$
18: if $result$ then
19:     $\theta_V^B \leftarrow EncSub(\theta_V^B, \theta_V^A)$          ▷ Partially filled order
20:     $\theta_V \leftarrow \theta_V^A$                                ▷ Traded volume
21:     $\theta_V^A \leftarrow EncField(0)$                  ▷ Marked order as filled
22: else
23:     $\theta_V^A \leftarrow EncSub(\theta_V^A, \theta_V^B)$
24:     $\theta_V \leftarrow \theta_V^B$
25:     $\theta_V^B \leftarrow EncField(0)$
26: end if
27: $PublishOrder(\theta_P^A, \theta_V)$
```