

FilterMatch: An Efficient Privacy-Preserving Inventory Matching Framework

Zheng Qu^{a,b}, Guopeng Lin^{a,b}, Peiran Wang^{a,b}, Ruotao Deng^{a,b}, Wen Zeng^{a,b} and Weili Han^{a,b,*}

^aCollege of Computer Science and Artificial Intelligence, Fudan University, Shanghai, 200438, P. R. China

^bLaboratory for Data Security and Governance, Fudan University, Shanghai, 200438, P. R. China

ARTICLE INFO

Keywords:

Multi-party computation
Privacy-preserving
Inventory matching
Secret sharing

ABSTRACT

Inventory matching is a widely used financial mechanism that market makers offer to clients to protect their trading intentions from the public market. A fundamental problem with traditional inventory matching is that it reveals clients' trading information to the market maker. To solve this problem, J.P. Morgan AI Research proposed PrimeMatch, a privacy-preserving inventory matching framework. Nonetheless, PrimeMatch suffers from two inefficiencies: (1) its communication overhead scales rapidly with the size of the stock universe; (2) its communication overhead is amplified due to the repeated comparison computation in its secure minimum protocol. In this paper, we propose FilterMatch, an efficient privacy-preserving inventory matching framework via communication optimization. Specifically, (1) we propose a two-stage matching mechanism to reduce the overhead caused by unmatched stocks, making it insensitive to stock universe size; (2) we design a novel secure three-way comparison protocol to reduce the overhead caused by comparison, avoiding the repeated computation. Experimental results demonstrate that FilterMatch achieves a $20.22\times \sim 108.17\times$ reduction in communication size and a $3.32\times \sim 8.63\times$ speedup in runtime compared to PrimeMatch.

1. Introduction

Inventory matching is a widely used financial mechanism that market makers offer to clients to protect their trading intentions from the public market [1]. Specifically, each client first sends to the market maker (e.g., banks or brokerages) a trading list containing their trading intentions. Then, the market maker locates the matched ones and executes the trades internally. This process allows clients to avoid adverse price impact triggered by publicly exposed trading intentions.

Although inventory matching presents a private alternative to public market matching, it inevitably exposes clients' trading information to the market maker. The market maker could infer clients' investment strategies from their submitted trading lists and exploit this knowledge for profit. Moreover, concentrating all trading lists at a market maker makes it vulnerable to single-point-of-failure events. Any insider abuse, infrastructure breakdown, or security breach could lead to large-scale disclosure of clients' trading records.

To solve this problem, J.P. Morgan AI Research proposes PrimeMatch [1], the first framework deployed in the financial production environment [1]. In PrimeMatch, each paired client sends their trading lists to the market maker in encrypted form. Then, through the joint execution of a secure comparison protocol, only the minimum of the intended trading quantities for each stock of mutual interest is revealed to the market maker.

PrimeMatch, however, suffers from two inefficiencies as follows: (1) Its communication overhead scales rapidly with the size of the stock universe. In the case of a stock universe with 10,000 stocks, it takes about 16 minutes for two clients to finish a complete matching process. Such latency could be unacceptable. Specifically, PrimeMatch requires clients to submit an intended trading quantity for each stock in the universe (using 0 for the uninterested stocks). Determining the executed trading quantity requires running the secure minimum comparison protocol for each stock, resulting in a huge communication overhead. (2) Its communication overhead is amplified due to the repeated comparison computation in its secure minimum protocol. Typically, a single comparison direction that only computes 'greater-than' (or 'less-than') is insufficient for ensuring symmetric disclosure for both paired clients. It is required to distinguish three cases (greater than, less than, equal to) to inform each client whether any of its intended quantity remains after trades. To achieve this, PrimeMatch directly computes two comparison outputs in opposite directions within its secure minimum protocol.

In this paper, we propose FilterMatch, an efficient privacy-preserving inventory matching framework via communication optimization. We overcome the two aforementioned shortcomings through the following two optimizations: (1) We propose a two-stage matching mechanism. In the first stage, we verify the zero equality of the intended trading quantity for each stock. If it holds, we can skip that stock and save an invocation of the secure comparison protocol. In the second stage, we proceed to compute the trading quantities for the remaining stocks, which are of mutual interest of both clients. Since the mutually interested stocks are relatively sparse compared to the entire stock universe, the first-stage filtering reduces the communication overhead by an order of

*Corresponding author

✉ 24110240070@fudan.edu.cn (Z. Qu); 17302010022@fudan.edu.cn (G. Lin); prwang@fudan.edu.cn (P. Wang); 23212010008@fudan.edu.cn (R. Deng); 23212010047@fudan.edu.cn (W. Zeng); wlhan@fudan.edu.cn (W.H.)
ORCID(s): 0000-0002-4861-8470 (Z. Qu); 0000-0001-8825-1987 (G. Lin); 0009-0001-3038-1946 (P. Wang); 0009-0007-0679-7283 (R. Deng); 0009-0003-2408-4480 (W. Zeng); 0000-0001-8663-436X (W.H.)

magnitude. (2) We design a new secure three-way comparison protocol. Instead of computing two comparison outputs, the new secure three-way comparison protocol determines all three cases (greater than, less than, equal to) by replacing one secure comparison with a lightweight equality test. Since the secure equality test protocol is much less costly than the secure comparison protocol, the communication overhead is reduced by nearly half.

The main contributions of FilterMatch are summarized as follows:

- We propose a two-stage matching mechanism to make communication overhead insensitive to the stock universe size.
- We design a new secure three-way comparison protocol to avoid the repeated comparison computation.

Compared to PrimeMatch, our proposed framework FilterMatch reduces the communication overhead by an order of magnitude. Specifically, for a complete inventory matching process, our proposed framework FilterMatch reduces the communication complexity from $O(u)$ to $O(m)$, where u is the size of stock universe and m is the number of matched stocks. For a single round of protocol execution for each client pair, our proposed framework FilterMatch reduces the communication complexity from $O(n^2)$ to $O(n)$, where n is the bit length of the input value. The experimental results demonstrate that FilterMatch achieves a $20.22\times \sim 108.17\times$ reduction in communication size and a $3.32\times \sim 8.63\times$ speedup in runtime¹. These results show that FilterMatch is much more efficient and practical for real-world deployment.

Organization: The rest of this paper is organized as follows. Section 2 investigates related works. Section 3 presents the technical preliminaries. Section 4 presents the design of the proposed framework FilterMatch. Section 6 presents the experimental evaluation. Section 7 presents a discussion about future work. Finally, Section 8 concludes the paper.

2. Related Works

2.1. Inventory Matching Frameworks

Inventory matching can be regarded as a special case of dark pool matching. Dark pool matching refers to matching buy and sell orders without exposing them to the open market [2]. The primary difference is that orders in dark pools contain an acceptable price range, whereas inventory matching operates at a fixed price. Privacy-preserving inventory matching frameworks [3, 4, 5, 6, 7, 1, 8, 9] are primarily realized through Homomorphic Encryption (HE), Differential Privacy (DP) or Secret Sharing (SS) techniques. Balch et al. [4] propose SecretMatch based on HE. Clients encrypt their orders locally and submit them to the market maker. The market maker directly executes order matching

over the ciphertexts. Polychroniadou et al. [8] propose Atlas-X based on DP. It adds noise to the market maker's inventory list before publishing it to clients. Polychroniadou et al. [9] propose indifferential privacy based on DP. Clients submit real orders together with fake orders. The true orders are revealed only when there is a match. Compared with HE and DP, SS techniques are more commonly adopted. Cartledge et al. [5] propose a multi-server matching architecture. Clients split their orders into shares and send each share to one server, so that no single server can reconstruct a complete order. Da Gama et al. [7] propose a bucket-based matching strategy. It groups clients' orders by volume and performs matching within each bucket. Polychroniadou et al. [1] propose PrimeMatch. It enables clients to find potential matches while keeping unmatched trading interests hidden.

Limitations: In general, existing privacy-preserving inventory matching frameworks face challenges in simultaneously achieving strict privacy and high efficiency. HE-based methods incur high computation overhead. DP-based methods only provide relaxed privacy guarantees. SS-based methods introduce scalability limitations.

2.2. Secure Comparison Protocols

Secure comparison is the core building block of privacy-preserving inventory matching, which requires determining the minimum trading quantity. Existing secure comparison protocols are primarily realized through SS [10, 11, 12, 13, 14], Garbled Circuits (GC) [15, 16, 17, 18, 19, 20], HE [21, 22, 23], and Function Secret Sharing (FSS) [24, 25, 26, 27]. Mohassel et al. [12] propose ABY3 based on SS. It converts input values from the form of arithmetic shares to Boolean shares. Then, a Boolean comparison circuit is built and evaluated to obtain the comparison result. Wagh et al. [14] propose SecureNN, which is also built on SS. It adds random masks to input values and obtains intermediate values. The comparison result is then computed from these intermediate values and the masks. Mohassel et al. [16] propose SecureML based on GC. It encodes each bit of the input value as garbled labels. The labels are exchanged and decoded to recover the comparison result. Hu et al. [28] propose OPERA based on the HE technique. It batches the input values for multiple comparisons before encryption. Then, the ciphertexts are evaluated in parallel to obtain the comparison results. Xing et al. [27] propose a comparison protocol based on FSS. It generates function keys in an offline phase. In the online phase, these keys are evaluated on the input values and return shares of the comparison result.

Limitations: Existing secure comparison protocols are not tailored for bidirectional comparison in inventory matching. Directly applying them introduces additional protocol invocations and increases the overall overhead.

3. Preliminaries

3.1. Notation

The notation used throughout this paper is summarized in Table 1.

¹The source code is publicly available on <https://github.com/ZhengQu7777/PrimeMatch.git>.

Table 1
Summary of Notation

Notation	Description
PPT	probabilistic polynomial time
λ	the security parameter
u	the size of the stock universe
n	the bit length of the input value
m	the number of interested stocks
$[n]$	$0, \dots, n-1$
P_i	the client i
P^*	the market maker
p	the field size
$\langle x \rangle_i^A$	the additive share held by P_i
$\langle x \rangle_i^B$	the boolean share held by P_i

3.2. Secret sharing

In this paper, we use two different secret sharing schemes designed for the two-party setting: the arithmetic sharing scheme [29] and the boolean sharing scheme [30]. In the arithmetic sharing scheme, a value $x \in \mathbb{Z}_p$ is divided into two shares $\langle x \rangle_0^A, \langle x \rangle_1^A \in \mathbb{Z}_p$, such that $\langle x \rangle_0^A + \langle x \rangle_1^A \equiv x \pmod{p}$. This scheme consists of two algorithms as follows:

1. **Sharing.** $\text{Shr}^A(x)$: Given an input x , the algorithm samples $\langle x \rangle_0^A \leftarrow_R \mathbb{Z}_p$ and computes $\langle x \rangle_1^A = x - \langle x \rangle_0^A \pmod{p}$. Then, it outputs $\langle x \rangle_0^A$ and $\langle x \rangle_1^A$.
2. **Reconstructing.** $\text{Rec}^A(\langle x \rangle_0^A, \langle x \rangle_1^A)$: Given input shares $\langle x \rangle_0^A$ and $\langle x \rangle_1^A$, the algorithm computes $x = \langle x \rangle_0^A + \langle x \rangle_1^A \pmod{p}$ and outputs x .

In the boolean sharing scheme, a value $x \in \mathbb{Z}_{2^n}$ is divided into two shares $\langle x \rangle_0^B, \langle x \rangle_1^B \in \mathbb{Z}_{2^n}$, such that $\langle x \rangle_0^B \oplus \langle x \rangle_1^B \equiv x$. It consists of the following two algorithms:

1. **Sharing.** $\text{Shr}^B(x)$: Given an input x , the algorithm samples $\langle x \rangle_0^B \leftarrow_R \mathbb{Z}_{2^n}$ and computes $\langle x \rangle_1^B = x \oplus \langle x \rangle_0^B$. Then, it outputs $\langle x \rangle_0^B$ and $\langle x \rangle_1^B$.
2. **Reconstructing.** $\text{Rec}^B(x)$: Given input shares $\langle x \rangle_0^B$ and $\langle x \rangle_1^B$, the algorithm computes $x = \langle x \rangle_0^B \oplus \langle x \rangle_1^B$ and outputs x .

Both secret sharing schemes have the linearity property. Linear operations (addition/Exclusive OR, multiplication/AND by a constant) performed on the shares can directly carry over to the reconstructed secret.

3.3. Commitment scheme

A commitment scheme [31] allows one to make a commitment to a message while concealing its content from others. A commitment scheme contains two probabilistic algorithms: $\text{Gen}(\cdot)$ and $\text{Com}(\cdot, \cdot; \cdot)$. Given the security parameter λ , $\text{Gen}(1^\lambda)$ outputs the public parameter pps . Given the message m , a random r and pps , $\text{Com}(pps, m; r)$ outputs the commitment com of m . A commitment scheme consists of the following three phases:

1. **Setup phase:** A trusted third party runs $\text{Gen}(1^\lambda)$ and sends the public parameter pps to the committer and verifier.
2. **Commit phase:** The committer chooses a random r and the message m to be committed. Then the committer runs $\text{Com}(pps, m; r)$ and sends the commitment com to the verifier.
3. **Open phase:** The committer sends the random r and the message m to the verifier. The verifier checks whether the equation $com = \text{Com}(pps, m; r)$ holds.

A commitment scheme has two basic properties: the hiding property and the binding property. The hiding property means every PPT adversary cannot acquire any information about the message m from the commitment com . The binding property means every PPT adversary cannot find a pair of (m_0, r_0) and (m_1, r_1) , such that $\text{Com}(pps, m_0; r_0) = \text{Com}(pps, m_1; r_1)$. Besides, a commitment scheme can be homomorphic, which means $\text{Com}(pps, m_0 + m_1; r_0 + r_1) = \text{Com}(pps, m_0; r_0) * \text{Com}(pps, m_1; r_1)$.

3.4. Coin Tossing Scheme

A coin-tossing (or coin-flipping) scheme [32] allows two parties to jointly generate a random number. The generated randomness can be expanded by a pseudorandom generator. In this paper, the coin-tossing scheme is denoted as Π_{coin} . It is used by the clients to generate a common random seed.

4. Design of FilterMatch

4.1. Overview

As shown in Figure 1, FilterMatch involves two entities: clients and the market maker. Each client holds a trading list containing the stock identity id , the trading direction dir and the intended trading quantity num . For every possible stock, the trading list contains two values, corresponding to the buy and sell directions. A value of 0 in the num column indicates the client has no trading interest in that stock and direction. Before matching, each client submits its trading list in encrypted form to the market maker, which acts as a central server. After matching, the market maker obtains the final quantity to be executed for each matched stock without accessing the plaintext trading lists.

4.2. Two-Stage Matching Mechanism

The two-stage matching mechanism is formally described in Protocol 1. Before the two stages, the market maker pairs clients in a random order. Each paired client establishes a secure channel, while the market maker only relays messages between them. In Stage 1, the market maker filters out unmatched stocks for which at least one client has no trading interest. Specifically, each client computes one interest bit for each stock, indicating whether it has a nonzero intended trading quantity. Then a secure Boolean multiplication protocol is run on the two interest bits, and a result of 1 means that both clients have trading interest

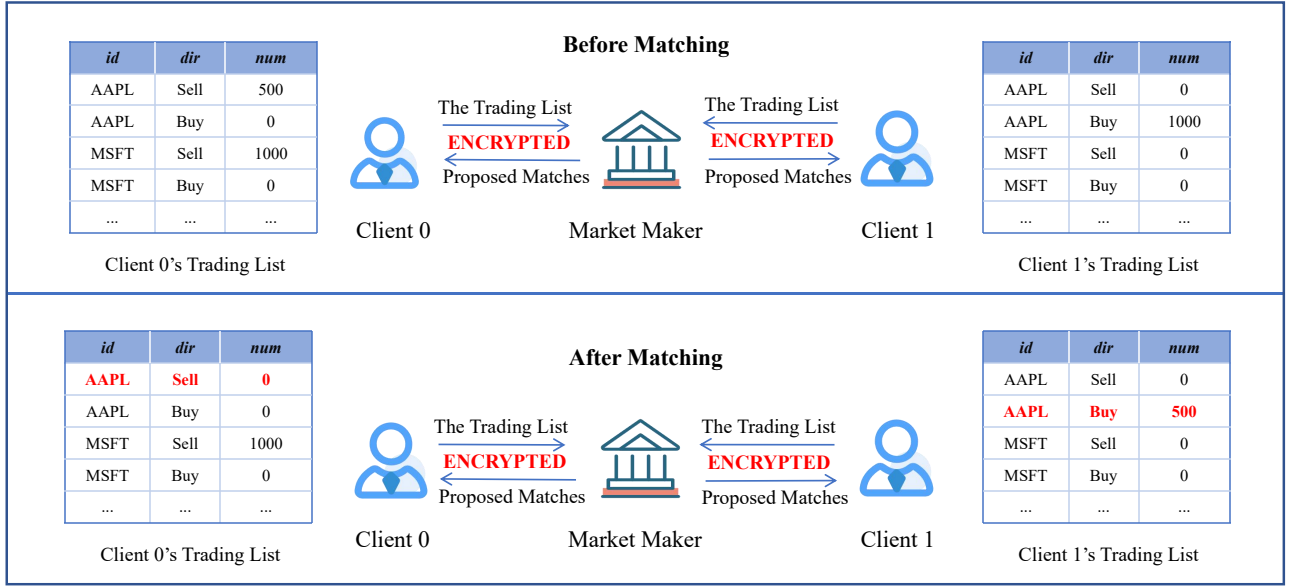


Figure 1: Overview of FilterMatch

in this stock. In Stage 2, for the remaining matched stocks, the market maker and clients run Protocol 2 to determine the executed trading quantities. The market maker then executes the corresponding trades. During this process, only the executed trading quantity is exposed to the market maker. Upon completion of the trades for the current pair, the market maker proceeds to the next pair and repeats the process.

Protocol 1: Two-Stage Mechanism

Input: The client $P_i (i \in \{0, 1\})$ with the trading list $\mathcal{L}_i = \langle (id_k, dir_k, num_k) \rangle_{k \in [u]}$, the market maker P^* without input.

Output: None

- 1: P^* sets the security parameter λ , and P_0 and P_1 establish a secure channel using λ .
- 2: P_0 and P_1 run the coin-tossing scheme Π_{coin} together, and obtain a common random seed s .
- 3: **Stage 1:** For each $k \in [u]$, each P_i locally computes an interest bit $h_{i,k}$, where $h_{i,k} = 1$ if $num_{i,k} \neq 0$ and $h_{i,k} = 0$ otherwise.
- 4: For each $k \in [u]$, P^* , P_0 , P_1 run secure Boolean multiplication protocol on $h_{0,k}$ and $h_{1,k}$. Then P^* obtains $a_k = h_{0,k} \cdot h_{1,k}$.
- 5: **Stage 2:** For each k where $a_k = 1$, P^* , P_0 , P_1 run Protocol 2 on $num_{0,k}$ and $num_{1,k}$. Then P^* obtains $c_{k,0}$ and $c_{k,1}$.
- 6: For each k where $a_k = 0$, P^* directly sets $c_{k,0}, c_{k,1} = 0$.
- 7: For each $k \in [u]$, P^* sends $c_{k,0}$ to P_0 , and $c_{k,1}$ to P_1 .
- 8: Receiving $c_{k,i} = 1$, each P_i sends its $num_{i,k}$ to P^* .

4.3. Secure Three-Way Comparison Protocol

As illustrated in Protocol 2, the secure three-way comparison protocol takes two unsigned integer values v_0 and v_1 as input and outputs two boolean values c_0 and c_1 indicating whether $v_0 \leq v_1$ and $v_1 \leq v_0$. This protocol can be divided into three parts: the equality test (step 1-5), the one-sided comparison (step 6-11) and the final decision (step 12).

Equality test. This part is designed to verify the equality of two unsigned integers. For two unequal unsigned integer values, subtracting one from the other would yield a non-zero result. We can verify the inequality by checking for the presence of a '1' for each bit position of the result. In the three-party computation environment, the clients P_0, P_1 first perform Boolean secret sharing on their respective values v_0 and v_1 . Then each client locally XORs the two Boolean shares it holds for v_0 and v_1 . Finally, P_0 and P_1 permute the bit sequence using the pre-shared random seed and send the permuted value to the market maker P^* . The random permutation step hides the differing bit positions of v_0 and v_1 from P^* . P^* XORs the two received values and checks if any bit equals 1; if so, the two inputs are not equal. Compared to the original protocol, the communication and computation complexity of three parties are lowered from $O(n^2)$ to $O(n)$, where n is the bit length of the input values.

Comparison. This part is designed to check whether the first unsigned value is strictly less than the second. For two unsigned integer values v_0 and v_1 , we search for the first differing bit downwards from the most significant bit to the least significant bit. Obviously, the value whose first differing bit is set to '1' is the larger one. In the three-party computation environment, clients P_0 and P_1 first perform additive sharing on each bit of their input values in $\mathbb{Z}_p$. Then each client locally computes $w_{i,j} = \langle v_0[j] \rangle - \langle v_1[j] \rangle$ and $f_{i,j} = 1 + w_{i,j} + w_{i,j}^{accum}$. The constant term 1 forces $f_{i,j}$ to

Protocol 2: Secure Three-way Comparison

Input: The market maker P^* without input, the client $P_i (i \in \{0, 1\})$ with unsigned integer value $v_i \in [2^n - 1]$ and a common random seed s .

Output: P^* obtains $c_0 = (v_0 \leq v_1)$ and $c_1 = (v_1 \leq v_0)$. P_0 and P_1 obtain c_0 and c_1 respectively.

- 1: **(Equality test)** Each client P_i computes a pair of Boolean shares $(\langle v_i \rangle_0^B, \langle v_i \rangle_1^B)$ of its input v_i and sends $\langle v_i \rangle_{1-i}^B$ to P_{1-i} .
- 2: Upon receiving $\langle v_{1-i} \rangle_i^B$ from P_{1-i} , each P_i computes $c_i = \langle v_i \rangle_i^B \oplus \langle v_{1-i} \rangle_i^B$.
- 3: Each P_i generates the same random permutation π through the random seed s .
- 4: Each P_i computes d_i such that $d_i[j] = c_i[\pi(j)]$ for $j \in [n]$, and sends d_i to P^* .
- 5: P^* computes $e = d_0 \oplus d_1$ and sets $a = 1$ if all $e[j] = 0$ for $j \in [n]$ (indicating $v_0 = v_1$), and $a = 0$ otherwise.
- 6: **(Comparison)** Each P_i computes $v_i[j] \in \{0, 1\}$ for each bit position $j \in [n]$, such that $v_i = \sum_{j=0}^{n-1} v_i[j] \cdot 2^{n-1-j}$.
- 7: Each P_i computes a set of additive share pairs $\{(\langle v_i[j] \rangle_i^p, \langle v_i[j] \rangle_{1-i}^p)\}_{j \in [n]}$ and sends the set $\{\langle v_i[j] \rangle_{1-i}^p\}_{j \in [n]}$ to P_{1-i} .
- 8: Upon receiving the set $\{\langle v_{1-i}[j] \rangle_i^p\}_{j \in [n]}$ from P_{1-i} , each P_i initializes $w_{i,0}^{accum} = 0$ and computes $w_{i,j} = \langle v_0[j] \rangle_i - \langle v_1[j] \rangle_i$, $f_{i,j} = 1 + w_{i,j} + w_{i,j}^{accum}$, $w_{i,j}^{accum} += w_{i,j}^{accum} + 2^{2+j} \cdot w_{i,j}$ by iterating from $j = 0$ to $n - 1$.
- 9: Each P_i generates a random permutation π and a set of random scalars $\{r_j \in \mathbb{Z}_p\}_{j \in [n]}$ through the random seed s .
- 10: Each P_i computes $g_{i,j} = f_{i,\pi(j)} \cdot r_j$ and sends the sequence $(g_{i,0}, \dots, g_{i,n-1})$ to P^* .
- 11: P^* computes $g_j = g_{0,j} + g_{1,j} \pmod{p}$ for $j \in [n]$, and sets $b = 1$ if $g_j \neq 0$ for any $j \in [n]$ (indicating $v_0 < v_1$), and $b = 0$ otherwise.
- 12: **(Decision)** P^* computes $c_0 = (v_0 \leq v_1) = a \vee b$ and $c_1 = (v_1 \leq v_0) = a \vee (1 - b)$, and outputs (c_0, c_1) .

be zero at positions where $v_0[j] < v_1[j]$, i.e., when $v_0[j] = 0$ and $v_1[j] = 1$. The term $w_{i,j}^{accum}$ is to ensure $f_{i,j}$ equals zero only at the first position where $v_0[j] < v_1[j]$. Finally, P_0 and P_1 multiply each $f_{i,j}$ by a random scalar and permute the sequence using the pre-shared random seed. This step masks the first differing position between v_0 and v_1 . It is worth noting that a natural idea is to set $w_{i,j}^{accum}$ as $w_{i,j}^2$. However, it would break the linearity of the computation. Besides, setting $w_{i,j}^{accum}$ as $2^{2+j} \cdot w_{i,j}$ can also avoid the problem of wrap-around and false positives. For the server P^* , it adds up the $f_{i,j}$ from P_0 and P_1 and checks whether $f_{i,j} = 0$ for each bit j . If so, P^* concludes $v_0 < v_1$.

Decision. This part is designed to compute the final results for two clients. The market maker P^* can obtain $v_0 \leq v_1$ and $v_1 \leq v_0$ directly from $v_0 = v_1$ and $v_0 < v_1$. Then P^* sends the results to P_0 and P_1 respectively.

Theorem 1. Let v_0 and v_1 be two unsigned integers in $\{0, \dots, 2^n - 1\}$. Assume that the field size p satisfies

$$p > 2 + 4 \cdot (2^n - 1).$$

Then Protocol 2 correctly determines whether $v_0 \leq v_1$ and $v_1 \leq v_0$.

The proof follows the same argument as Theorem 4.4 in PrimeMatch [1] and is omitted here. The above condition prevents wrap-around in $\mathbb{Z}_p$ and therefore avoids false positives during comparison.

5. Security Analysis

5.1. Security Model

We work in the standard semi-honest security model under an honest-majority assumption. In this setting, all parties faithfully follow the prescribed protocol while only attempting to infer additional information from the messages they observe. Under the honest-majority assumption, the adversary is further restricted to corrupting fewer than half of the parties. Specifically, we focus on a three-party scenario comprising two clients and a single market maker. In this context, the adversary controls at most one party, and no collusion is possible.

We analyze security following the standard simulation paradigm [33, 34]. The core idea is that for any adversary observing the real protocol execution, one can construct a simulator in the ideal world that reproduces an indistinguishable view using only the input and output permitted by the ideal functionality. If such a simulator exists, then the adversary cannot learn anything beyond what the ideal functionality reveals. Hence, real and ideal executions become indistinguishable to any environment, establishing simulation-based security under the semi-honest model.

Theorem 2. Protocol 2 securely implements the ideal three-way comparison functionality $\mathcal{F}_{\text{TWC}}$ in the presence of a static, semi-honest adversary corrupting at most one party.

Proof. We first describe the ideal functionality that the protocol aims to securely realize.

Ideal Functionality $\mathcal{F}_{\text{TWC}}$

Input. Client P_0 holds $v_0 \in [2^n - 1]$, client P_1 holds $v_1 \in [2^n - 1]$, and the market maker P^* has no input.

Output. The functionality computes the comparison bits

$$c_0 = (v_0 \leq v_1), \quad c_1 = (v_1 \leq v_0),$$

and outputs c_0 to P_0 , c_1 to P_1 , and (c_0, c_1) to P^* .

Let $\text{VIEW}_i^\Pi(v_0, v_1)$ denote the view of party P_i during an execution of Protocol 2, including its input, local randomness, and all messages received. Let $\text{OUT}^\Pi(v_0, v_1)$ denote the outputs of the parties. We consider two corruption cases.

Case 1: A corrupted client P_i . Assume without loss of generality that P_0 is corrupted. Its view consists of:

- its input v_0 and randomness used for Boolean sharing and additive sharing,
- the Boolean share $\langle v_1 \rangle_0^B$ received during the equality-test phase,
- the permuted vector d_1 received via P^* ,
- the additive shares $\langle v_1[j] \rangle_0^p$ received during the comparison phase,
- and the intermediate values $w_{0,j}, f_{0,j}, g_{0,j}$ computed locally.

We construct a simulator Sim_0 having access only to v_0 and the bit $c_0 = (v_0 \leq v_1)$ supplied by the ideal world.

1. It samples random Boolean shares $\langle v_1 \rangle_0^B$ and generates random d_1 using a random permutation π , matching the distribution of the equality-test phase.
2. It computes the bit-decomposition of v_0 and samples random additive shares $\langle v_1[j] \rangle_0^p$ uniformly for all j .
3. Using these random values and fresh randomness for π and $\{r_j\}$, it locally computes $w_{0,j}, f_{0,j}, g_{0,j}$ exactly following the protocol.
4. It outputs the simulated view

$$\widetilde{\text{VIEW}}_0 = (v_0, \text{randomness}, \langle v_1 \rangle_0^B, d_1, (\langle v_1[j] \rangle_0^p)_j, (w_{0,j})_j, (f_{0,j})_j, (g_{0,j})_j).$$

Since Boolean shares and additive shares are information-theoretically independent of v_1 , and all other values are deterministic functions of locally simulated randomness and v_0 , we have

$$\widetilde{\text{VIEW}}_0 \equiv \text{VIEW}_0^\Pi(v_0, v_1),$$

establishing perfect simulation.

Case 2: The corrupted market maker P^* . In the real protocol, P^* receives:

- the vectors d_0 and d_1 from the equality-test phase,
- their XOR value $e = d_0 \oplus d_1$ and the bit $a = (v_0 = v_1)$,
- the masked values $(g_{0,j})_j$ and $(g_{1,j})_j$ from the comparison phase,
- the combined values $g_j = g_{0,j} + g_{1,j} \bmod p$ and the bit $b = (v_0 < v_1)$.

From the protocol structure:

- d_0 and d_1 are uniformly random except that e is all-zero iff $v_0 = v_1$,
- each g_j is uniformly random in $\mathbb{Z}_p$ except for the zero/non-zero pattern determined by b .

The simulator Sim_* receives (c_0, c_1) from the ideal functionality and recovers $a = (c_0 \wedge c_1)$ and $b = c_0 \wedge (1 - a)$.

It then simulates:

1. random d_0, d_1 (and thus e) consistent with the bit a ,
2. random $g_j \in \mathbb{Z}_p$ with zero/non-zero pattern consistent with b ,
3. random additive decompositions $g_j = g_{0,j} + g_{1,j} \bmod p$.

The simulated view $\widetilde{\text{VIEW}}_*$ is identically distributed to the real-world view $\text{VIEW}_*^\Pi(v_0, v_1)$.

Combining the above two cases, we obtain that for any semi-honest adversary corrupting at most one party, there exists a polynomial-time simulator that reproduces the adversary's view using only the input/output of the ideal functionality. Therefore, Protocol 2 securely realizes $\mathcal{F}_{\text{TWC}}$ in the semi-honest model. $\square$

6. Evaluation

6.1. Experiment Setting

Implementation: We implement FilterMatch and the PrimeMatch baseline in C++. Both frameworks are conducted in a three-party setting with two clients and one server. Each party is simulated by a separate process. The computation of additive shares is performed over the prime field $\mathbb{Z}_p$. In our implementation, the trading quantity is represented as a 32-bit unsigned integer, i.e., $n = 32$. As formalized in Theorem 1, the field size should satisfy $p > 2 + 4 \cdot (2^{32} - 1) = 17,179,869,182$. Therefore, we choose the prime field size $p = 17,179,869,209$ and use `uint64` to represent field elements. Our implementation utilizes a thread pool for each party to parallelize the computation across different stocks. Our implementation is intended as a proof-of-concept to focus on the efficiency of the core protocols. Therefore, practical deployment considerations, such as identity authentication, are considered out of scope and ignored here.

Table 2

Total sent bytes (KB), total received bytes (KB), runtime (ms) in WAN and runtime (ms) in LAN of FilterMatch vs. PrimeMatch for varying stock universe size with fixed number of mutually interested stocks.

	Framework	Stock Universe Size									
		2000		4000		6000		8000		10000	
		Server	Client	Server	Client	Server	Client	Server	Client	Server	Client
Total Sent Bytes	FilterMatch	4.02 (0.97×)	46.10 (33.22×)	7.93 (0.98×)	61.73 (49.61×)	11.84 (0.99×)	77.35 (59.39×)	15.74 (0.99×)	92.98 (65.87×)	19.65 (0.99×)	108.60 (70.50×)
	PrimeMatch	3.91	1531.25	7.81	3062.50	11.72	4593.75	15.62	6125.00	19.53	7656.25
Total Received Bytes	FilterMatch	46.57 (44.29×)	24.82 (20.22×)	62.20 (66.32×)	34.59 (29.02×)	77.82 (79.51×)	44.36 (33.95×)	93.45 (88.28×)	54.12 (37.10×)	109.07 (94.55×)	63.89 (39.28×)
	PrimeMatch	2062.51	501.95	4125.01	1003.91	6187.51	1505.86	8250.01	2007.81	10312.51	2509.77
Runtime in WAN	FilterMatch	387.79 (3.58×)	347.57 (3.89×)	478.53 (4.58×)	403.96 (5.24×)	517.99 (5.73×)	409.93 (6.99×)	590.84 (6.28×)	450.61 (7.92×)	660.13 (6.63×)	486.86 (8.63×)
	PrimeMatch	1388.82	1350.66	2189.44	2117.01	2970.53	2864.28	3708.84	3567.17	4373.64	4200.67
Runtime in LAN	FilterMatch	88.01 (3.32×)	53.34 (4.81×)	163.00 (3.54×)	93.06 (5.43×)	228.90 (3.74×)	128.03 (5.86×)	309.86 (3.65×)	172.54 (5.75×)	370.55 (4.05×)	200.31 (6.63×)
	PrimeMatch	292.14	256.48	576.28	505.04	855.39	750.79	1130.89	992.81	1499.44	1327.37

Table 3

Total sent bytes (KB), total received bytes (KB), and runtime (ms) of FilterMatch vs. PrimeMatch for varying number of mutually interested stocks with fixed stock universe size.

	Framework	Number of Mutually Interested Stock									
		20		40		60		80		100	
		Server	Client	Server	Client	Server	Client	Server	Client	Server	Client
Total Sent Bytes	FilterMatch	11.76 (1.00×)	57.04 (80.54×)	11.80 (0.99×)	67.20 (68.36×)	11.84 (0.99×)	77.35 (59.39×)	11.88 (0.99×)	87.51 (52.49×)	11.91 (0.98×)	97.66 (47.04×)
	PrimeMatch	11.72	4593.75	11.72	4593.75	11.72	4593.75	11.72	4593.75	11.72	4593.75
Total Received Bytes	FilterMatch	57.20 (108.17×)	34.32 (43.88×)	67.51 (91.65×)	39.34 (38.28×)	77.82 (79.51×)	44.36 (33.95×)	88.13 (70.21×)	49.38 (30.50×)	98.45 (62.85×)	54.39 (27.69×)
	PrimeMatch	6187.51	1505.86	6187.51	1505.86	6187.51	1505.86	6187.51	1505.86	6187.51	1505.86
Runtime in WAN	FilterMatch	619.32 (4.80×)	512.25 (5.59×)	523.14 (5.68×)	417.00 (6.87×)	517.98 (5.73×)	409.06 (7.00×)	532.34 (5.58×)	425.90 (6.73×)	534.76 (5.55×)	426.93 (6.71×)
	PrimeMatch	2970.53	2864.28	2970.53	2864.28	2970.53	2864.28	2970.53	2864.28	2970.53	2864.28
Runtime in LAN	FilterMatch	229.40 (3.73×)	125.18 (6.00×)	232.24 (3.68×)	130.03 (5.77×)	225.25 (3.80×)	124.28 (6.04×)	231.49 (3.70×)	131.98 (5.69×)	234.19 (3.65×)	132.67 (5.66×)
	PrimeMatch	855.39	750.79	855.39	750.79	855.39	750.79	855.39	750.79	855.39	750.79

Environment: All experiments are conducted on a dedicated Linux server running Ubuntu 20.04.6 LTS (Focal Fossa) with Linux kernel version 5.15.0-139-generic. The machine is equipped with two Intel(R) Xeon(R) Silver 4210R CPUs @ 2.40 GHz, each providing 10 physical cores and 20 hardware threads, for a total of 40 hardware threads, and 1 TiB of RAM. Both clients and the server in our three-party protocol are instantiated as separate processes on the same machine and communicate over the loopback interface (127.0.0.1). To emulate a Wide-Area Network (WAN) environment, we use the Linux traffic control tool ‘tc’ on the loopback device. In this configuration, we attach an HTB queuing discipline, which throttles the available bandwidth to 50 Mbps. We also employ netem to introduce a 20 ms one-way delay with 5 ms jitter. Finally, the settings include a 0.1% packet loss rate and a queue limit of 250 packets. In contrast, the Local-Area Network (LAN) environment corresponds to running over the unshaped loopback interface without additional delay or bandwidth limits.

Baseline: To ensure a fair comparison, we evaluate FilterMatch against a semi-honest implementation of PrimeMatch’s core secure comparison protocol. This baseline follows the same security model as FilterMatch and excludes the additional mechanisms required for malicious security, such as MAC-based checks, consistency checks, commitments, or zero-knowledge proofs.

Workload: The protocol efficiency is mainly determined by two workload factors: the stock universe size u and the number of mutually interested stocks m . Specifically, u and m dictate the communication overhead of the equality test and the comparison of Protocol 2, respectively. Other factors are not separately considered here. To reflect practical financial scenarios, we set u and m according to the practical scale reported by PrimeMatch, which has been deployed in production at J.P. Morgan.

6.2. Efficiency Evaluation

The efficiency evaluation considers two cases: varying the stock universe size with the fixed number of mutually interested stocks (Table 2), and varying the number of mutually interested stocks with fixed stock universe size (Table 3). For both cases, we evaluate the performance of the server and client sides using four key metrics: total sent and received bytes, and runtime in WAN and LAN environments. We vary the stock universe size from 2000 to 10000 to reflect the typical scale of real-world trading. Additionally, we vary the number of mutually interested stocks from 20 to 100 to reflect sparse mutual trading interest. Together, these metrics comprehensively capture the efficiency of FilterMatch.

6.2.1. Impact of Varying Stock Universe Size

Total Sent Bytes: On the server side, the total sent bytes are almost identical for both frameworks. This is because the server only needs to send the final output (c_0, c_1) of Protocol 2 to each client, leaving little room for further reduction. In contrast, on the client side, FilterMatch achieves a marked reduction over PrimeMatch, ranging from 33.22 $\times$ at 2000 stocks to 70.50 $\times$ at 10000 stocks. This is because PrimeMatch transmits all intermediate results as additive secret shares, whereas FilterMatch utilizes boolean secret shares. The size of a boolean share is approximately one-sixtieth of an additive share. FilterMatch uses additive shares only for those mutually interested stocks, so that the advantage of FilterMatch becomes more evident as the stock universe size increases.

Total Received Bytes: On the server side, FilterMatch achieves a substantial reduction over PrimeMatch, ranging from 44.29 $\times$ at 2000 stocks to 94.55 $\times$ at 10000 stocks. On the client side, FilterMatch achieves a reduction over PrimeMatch ranging from 20.22 $\times$ at 2000 stocks to 39.28 $\times$ at 10000 stocks. Both reductions can be attributed to the use of boolean secret shares in FilterMatch. Furthermore, a complete inventory matching process typically involves multiple pairs of clients. In the cases with tens of clients, the resulting reduction of total received bytes can amount to tens or even hundreds of GB.

Runtime in WAN: On the server side, FilterMatch achieves a speedup over PrimeMatch ranging from 3.58 $\times$ at 2000 stocks to 6.63 $\times$ at 10000 stocks. On the client side, FilterMatch achieves a speedup over PrimeMatch ranging from 3.89 $\times$ at 2000 stocks to 8.63 $\times$ at 10000 stocks. This speedup makes FilterMatch more scalable and efficient in practice.

Runtime in LAN: On the server side, FilterMatch achieves a limited speedup over PrimeMatch, ranging from 3.32 $\times$ at 2000 stocks and up to 4.05 $\times$ at 10000 stocks. This is expected since LAN environments offer much lower round-trip latency and much higher bandwidth. The communication optimization introduced by FilterMatch is weakened, resulting in a smaller relative speedup. On the client side, FilterMatch achieves a speedup over PrimeMatch ranging from 4.81 $\times$ at 2000 stocks and up to 6.63 $\times$ at 10000 stocks. This advantage is also less pronounced than that in WAN environments for the same reason.

6.2.2. Impact of Varying Number of Mutually Interested Stocks

Total Sent Bytes: On the server side, FilterMatch incurs more total sent bytes. This is because in FilterMatch, the server has an additional round of communication in Stage 1 for filtering. This is acceptable since the extra bytes are limited to several KB. On the client side, FilterMatch achieves a reduction over PrimeMatch, ranging from 80.54 $\times$ at 20 mutually interested stocks to 47.04 $\times$ at 100 mutually interested stocks. PrimeMatch processes every stock uniformly, regardless of whether it is of interest. In contrast, FilterMatch invokes an expensive secure comparison sub-protocol (part 2 of Protocol 2) for every mutually interested stock. Therefore,

Table 4

Client-side total sent/received bytes and runtime in LAN/WAN under varying values of security parameter λ .

Security Parameter λ	Total Sent Bytes (KB)	Total Received Bytes (KB)	WAN Runtime (ms)	LAN Runtime (ms)
80	77.41	44.42	667.24	259.90
112	77.42	44.42	661.11	257.87
128	77.42	44.43	662.44	259.25
192	77.43	44.43	664.16	261.40
256	77.43	44.44	665.87	260.67

the advantage of FilterMatch diminishes as the number of mutually interested stocks increases. However, FilterMatch still maintains a substantial advantage within the tested range.

Total Received Bytes: On the server side, FilterMatch achieves a reduction over PrimeMatch, ranging from 108.17 $\times$ at 20 mutually interested stocks to 62.85 $\times$ at 100 mutually interested stocks. On the client side, FilterMatch achieves a reduction over PrimeMatch, ranging from 43.88 $\times$ at 20 mutually interested stocks to 27.69 $\times$ at 100 mutually interested stocks.

Runtime in WAN: On the server side, FilterMatch achieves a speedup over PrimeMatch ranging from 4.80 $\times$ to 5.73 $\times$. On the client side, FilterMatch achieves a speedup over PrimeMatch ranging from 5.59 $\times$ to 7.00 $\times$.

Runtime in LAN: On the server side, FilterMatch achieves a speedup over PrimeMatch ranging from 3.65 $\times$ to 3.80 $\times$. On the client side, FilterMatch achieves a speedup over PrimeMatch ranging from 5.66 $\times$ to 6.04 $\times$.

6.3. Sensitivity to Security Parameter λ

To evaluate the efficiency of FilterMatch under different values of the security parameter λ , we vary λ among 80, 112, 128, 192, and 256 and keep all other experimental settings fixed. Table 4 reports the client-side results under different values of λ . The main reason is that the security of FilterMatch relies on information-theoretic hiding rather than computational hardness assumptions. Therefore, λ is used only for secure channel encryption, which incurs one-time and limited communication overhead. Changing λ has little effect on the overall efficiency. The server-side results follow the same trend and are omitted for brevity.

6.4. Ablation Evaluation

To evaluate the contribution of each optimization in FilterMatch, we incorporate the two-stage mechanism (TS) and the new three-way comparison protocol (TC) into PrimeMatch separately. The two resulting variants, denoted as PrimeMatch-TS and PrimeMatch-TC, are evaluated alongside the original PrimeMatch with respect to total sent bytes, total received bytes, and runtime in LAN and WAN. As shown in Figure 2 and Figure 3, across both the server and client sides, the two-stage mechanism provides the dominant improvement among the four metrics, particularly for the total sent and received bytes. The new three-way comparison protocol offers an auxiliary improvement mainly in terms of the runtime. The two-stage mechanism ensures that

FilterMatch

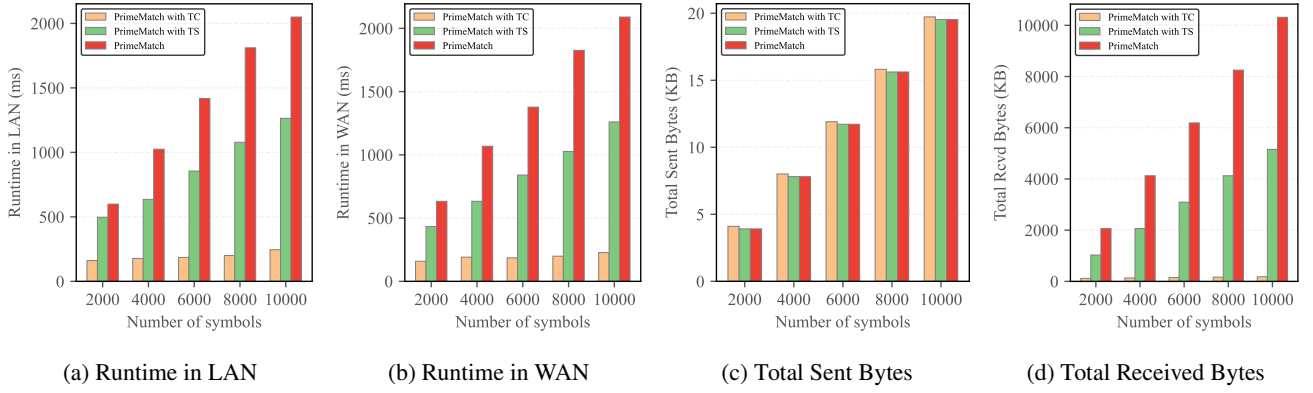


Figure 2: Server-side runtime in LAN/WAN and total sent/received bytes of PrimeMatch-TS, PrimeMatch-TC, and FilterMatch.

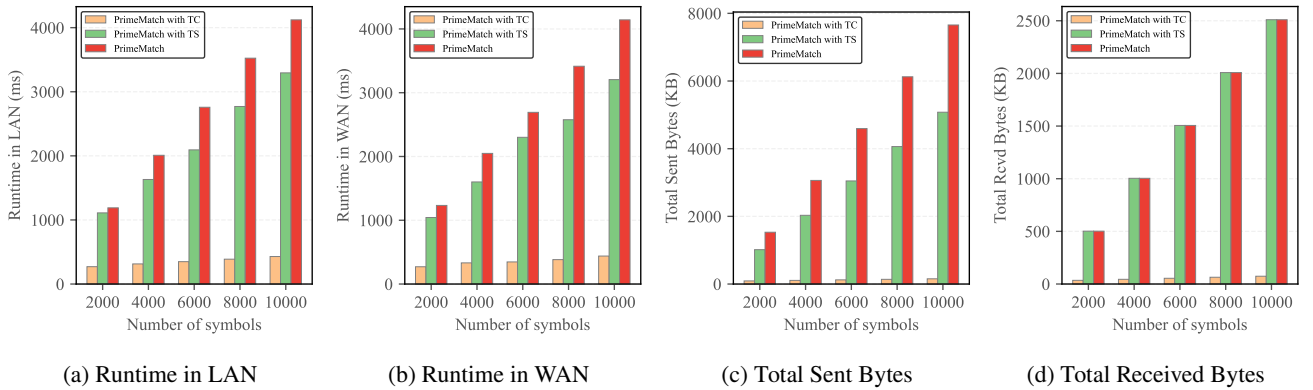


Figure 3: Client-side runtime in LAN/WAN and total sent/received bytes of PrimeMatch-TS, PrimeMatch-TC, and FilterMatch.

the computation and communication costs are no longer proportional to the size of the entire stock universe, but instead scale only with the number of mutually interested stocks, which greatly improves the efficiency and scalability. The new three-way comparison protocol further halves the communication and computation costs from the algorithmic level. Although its impact is less pronounced than that of the two-stage mechanism, it still provides a meaningful improvement.

7. Discussion and Future Work

Extension to Malicious Security. In the malicious adversary model, both clients and the market maker may deviate from the protocol specification. The client may send inconsistent intermediate results, and the market maker may return manipulated final results. Preventing such behaviors can be achieved by integrating commitment schemes and zero-knowledge proofs into the current framework. Each client can make commitments to its input values and use zero-knowledge proofs to ensure the consistency of intermediate results. The market maker can then verify the commitments and proofs before computing the final results. This requires a separate security proof and performance evaluation. We leave malicious security as future work.

Extension to Range-Based Matching. FilterMatch can also be extended to support range-based order specifications, where each participant provides a minimum and maximum acceptable quantity rather than a single value. Similar to the range mechanism in PrimeMatch, FilterMatch can support range-based orders by running the secure comparison protocol twice—once on minimum thresholds and once on full quantities. This design keeps unmatched or partially matched ranges hidden and preserves the same privacy guarantees, enabling FilterMatch to accommodate more expressive trading constraints without modifying the core protocol.

Extension to Two-Party Setting. In the two-party setting, the market maker directly matches its own inventory against the client's orders. There is no independent intermediary to assist in the computation. Both parties must learn the matched stocks without leaking their full trading interests. The secure three-way comparison protocol needs to be redesigned. One possible direction is to use homomorphic encryption to let both parties evaluate matching over encrypted orders. We keep this setting outside the scope of the current work.

8. Conclusion

In this paper, we propose FilterMatch, an efficient privacy-preserving inventory matching framework, and optimize the

communication overhead in two ways: (1) We present a two-stage matching mechanism to make the overhead insensitive to the stock universe size. (2) We design a new secure three-way comparison protocol to avoid repeated comparison computations. Experimental results demonstrate that FilterMatch achieves a $20.22\times \sim 108.17\times$ reduction in communication size and a $3.32\times \sim 8.63\times$ speedup in runtime compared to PrimeMatch.

Acknowledgment

This paper is supported by NSFC (Grant No. 92370120).

References

- [1] A. Polychroniadou, G. Asharov, B. Diamond, T. Balch, H. Buehler, R. Hua, S. Gu, G. Gimler, M. Veloso, Prime Match: A privacy-preserving inventory matching system, in: Proceedings of the 32nd USENIX Security Symposium (USENIX Security 23), USENIX Association, 2023, pp. 6417–6434.
- [2] H. Zhu, [Do Dark Pools Harm Price Discovery?](#), The Review of Financial Studies 27 (2014) 747–789.
- [3] G. Asharov, T. H. Balch, A. Polychroniadou, M. Veloso, PDPs: Privacy-preserving dark pools, in: Proceedings of the 19th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 20), International Foundation for Autonomous Agents and Multiagent Systems, 2020.
- [4] T. Balch, B. E. Diamond, A. Polychroniadou, SecretMatch: Inventory matching from fully homomorphic encryption, in: Proceedings of the First ACM International Conference on AI in Finance (ICAIF 20), ACM, 2020, pp. 1–7.
- [5] J. Cartledge, N. P. Smart, Y. Talibi Alaoui, MPC joins the dark side, in: Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security (AsiaCCS 19), ACM, 2019, pp. 148–159.
- [6] J. Cartledge, N. P. Smart, Y. Talibi Alaoui, Multi-party computation mechanism for anonymous equity block trading: A secure implementation of Turquoise Plato Uncross, Intelligent Systems in Accounting, Finance and Management 28 (2021) 239–267.
- [7] M. B. da Gama, J. Cartledge, A. Polychroniadou, N. P. Smart, Y. T. Alaoui, Kicking-the-bucket: Fast privacy-preserving trading using buckets, in: Proceedings of the International Conference on Financial Cryptography and Data Security (FC 22), Springer, 2022, pp. 20–37.
- [8] A. Polychroniadou, G. Cipriani, R. Hua, T. Balch, [Atlas-X equity financing: Unlocking new methods to securely obfuscate axe inventory data based on differential privacy](#), in: Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 24), International Foundation for Autonomous Agents and Multiagent Systems, 2024, pp. 1585–1592.
- [9] A. Polychroniadou, T.-H. H. Chan, A. Agrawal, [Indifferential privacy: A new paradigm and its applications to optimal matching in dark pool auctions](#), in: Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 25), International Foundation for Autonomous Agents and Multiagent Systems, 2025.
- [10] N. Koti, M. Pancholi, A. Patra, A. Suresh, SWIFT: Super-fast and robust privacy-preserving machine learning, in: Proceedings of the 30th USENIX Security Symposium (USENIX Security 21), USENIX Association, 2021, pp. 2651–2668.
- [11] S. Tan, B. Knott, Y. Tian, D. J. Wu, CryptGPU: Fast privacy-preserving machine learning on the GPU, in: Proceedings of the 2021 IEEE Symposium on Security and Privacy (S&P 21), IEEE, 2021, pp. 1021–1038.
- [12] P. Mohassel, P. Rindal, ABY3: A mixed protocol framework for machine learning, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS 18), ACM, 2018, pp. 35–52.
- [13] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, T. Rabin, Falcon: Honest-majority maliciously secure framework for private deep learning, arXiv preprint arXiv:2004.02229 (2020).
- [14] S. Wagh, D. Gupta, N. Chandran, SecureNN: 3-party secure computation for neural network training, Proceedings on Privacy Enhancing Technologies (2019).
- [15] D. Demmler, T. Schneider, M. Zohner, ABY: A framework for efficient mixed-protocol secure two-party computation, in: Proceedings of the Network and Distributed System Security Symposium (NDSS 15), Internet Society, 2015.
- [16] P. Mohassel, Y. Zhang, SecureML: A system for scalable privacy-preserving machine learning, in: Proceedings of the 2017 IEEE Symposium on Security and Privacy (S&P 17), IEEE, 2017, pp. 19–38.
- [17] A. Patra, T. Schneider, A. Suresh, H. Yalame, ABY2.0: Improved mixed-protocol secure two-party computation, in: Proceedings of the 30th USENIX Security Symposium (USENIX Security 21), USENIX Association, 2021, pp. 2165–2182.
- [18] H. Chaudhari, A. Choudhury, A. Patra, A. Suresh, Astra: High throughput 3PC over rings with application to secure prediction, in: Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop (CCSW 19), ACM, 2019, pp. 81–92.
- [19] W. Z. Srinivasan, P. Akshayaram, P. R. Ada, Delphi: A cryptographic inference service for neural networks, in: Proceedings of the 29th USENIX Security Symposium (USENIX Security), USENIX Association, 2019.
- [20] H. Chaudhari, R. Rachuri, A. Suresh, Trident: Efficient 4PC framework for privacy preserving machine learning, arXiv preprint arXiv:1912.02631 (2019).
- [21] I. Damgård, M. Geisler, M. Krøigård, Homomorphic encryption and secure comparison, International Journal of Applied Cryptography 1 (2008) 22–31.
- [22] I. Damgård, M. Geisler, M. Krøigård, A correction to “efficient and secure comparison for on-line auctions”, International Journal of Applied Cryptography 1 (2009) 323–324.
- [23] M. Fischlin, A cost-effective pay-per-multiplication comparison method for millionaires, in: Proceedings of the Cryptographers’ Track at the RSA Conference (CT-RSA 01), Springer, 2001, pp. 457–471.
- [24] E. Boyle, N. Gilboa, Y. Ishai, Function secret sharing, in: Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 15), Springer, 2015, pp. 337–367.
- [25] E. Boyle, N. Gilboa, Y. Ishai, Function secret sharing: Improvements and extensions, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS 16), ACM, 2016, pp. 1292–1303.
- [26] E. Boyle, N. Gilboa, Y. Ishai, Secure computation with preprocessing via function secret sharing, in: Proceedings of the Theory of Cryptography Conference (TCC 19), Springer, 2019, pp. 341–371.
- [27] P. Xing, H. Li, M. Hao, H. Chen, J. Hu, D. Liu, [Distributed function secret sharing and applications](#), in: Proceedings of the 32nd Network and Distributed System Security Symposium (NDSS 25), Internet Society, 2025.
- [28] Q. Hu, W. Chen, T. Shen, X. Yao, N. Zhang, H. Cui, S.-M. Yiu, [OPERA: Achieving secure and high-performance OLAP with parallelized homomorphic comparisons](#), in: Proceedings of the 2025 IEEE Symposium on Security and Privacy (S&P 25), IEEE, 2025, pp. 2360–2377. doi:10.1109/SP61157.2025.00173.
- [29] L. Xiong, W. Zhou, Z. Xia, Q. Gu, J. Weng, Efficient privacy-preserving computation based on additive secret sharing, arXiv preprint arXiv:2009.05356 (2020).
- [30] D. Wang, L. Zhang, N. Ma, X. Li, Two secret sharing schemes based on boolean operations, Pattern Recognition 40 (2007) 2776–2785.
- [31] T. P. Pedersen, Non-interactive and information-theoretic secure verifiable secret sharing, in: Proceedings of the Annual International Cryptology Conference (CRYPTO 91), Springer, 1991, pp. 129–140.
- [32] B. Barak, Constant-round coin-tossing with a man in the middle or realizing the shared random string model, in: Proceedings of the

43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 02), IEEE, 2002, pp. 345–355.

- [33] O. Goldreich, Foundations of Cryptography: Volume 2, Basic Applications, volume 2, Cambridge University Press, 2001.
- [34] Y. Lindell, How to simulate it: A tutorial on the simulation proof technique, Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich (2017) 277–346.