Efficient Commodity Matching for Privacy-Preserving Two-Party Bartering

Fabian Förg, Susanne Wetzel
Stevens Institute of Technology
Hoboken, NJ, USA
{ffoerg, swetzel}@stevens.edu

Ulrike Meyer
RWTH Aachen University
Aachen, Germany
meyer@itsec.rwth-aachen.de

ABSTRACT

Current bartering platforms place the burden of finding simultaneously executable quotes on their users. In addition, these bartering platforms do not keep quotes private. To address these shortcomings, this paper introduces a privacy-preserving bartering protocol secure in the semi-honest model. At its core, the novel bartering protocol uses a newly-developed bipartite matching protocol which determines simultaneously executable quotes in an efficient manner. While the new privacy-preserving bipartite matching protocol does not always yield the maximal set of simultaneously executable quotes, it keeps the parties' quotes private at all times. Moreover, our new privacy-preserving bipartite matching protocol is more efficient than existing solutions in that it only requires linear communication in the number of quotes the parties specify.

1. INTRODUCTION

In the past, bartering occurred predominantly one-to-one within local communities. Nowadays, online platforms such as *NATE* [20], *Read It Swap It* [22], *Craigslist* [6], and *TradeYa* [26] facilitate bartering beyond local communities. These bartering platforms typically allow their users to advertise their offered commodities and to indicate which demanded commodities they are willing to accept for an offer. This naturally leads to pairs of an offered and a demanded commodity which we refer to as quotes. If the offered commodity of a quote is equal to the demanded commodity of another user's quote and vice versa, we say that the quotes are compatible. As users commonly hold an offered commodity only at a single quantity (e.g., for books [22]), some quotes of a user may be competing in the sense that the offered or demanded commodities coincide. To facilitate an efficient exchange, it is desirable to determine as many simultaneously executable quotes as possible, i.e., to find a multitude of compatible quotes where the offered and demanded commodities are pairwise different. However, today's platforms generally require users to manually determine which of their quotes are compatible with a quote from another user and force users to disclose their quotes.

To the best of our knowledge, the work from [7] is the only approach to date addressing privacy-preserving bartering for two parties that keeps quotes private and eliminates the need for a trusted third party. However, a shortcoming of the protocol from [7] is that as input it allows only one quote per party and the protocol is therefore restricted to a narrow use case. By combining [7] and [27] in a straightforward fashion, it is possible to build a bartering protocol which always yields a maximal set of simultaneously executable quotes but is inefficient in that its computation and communication complexities range from $\Omega(n_0 \cdot n_1)$ to $\mathcal{O}(\min(n_0, n_1) \cdot n_0 \cdot n_1)$ where n_0 and n_1 denote the number of quotes from the first party and second party, respectively.

It is in this context that this paper proposes a novel and efficient two-party bartering protocol that determines simultaneously executable quotes based on set intersection and bipartite matching. Our main contribution is a probabilistic privacy-preserving bipartite matching protocol whose computation and communication complexities are in $\mathcal{O}(\min(n_0, n_1))$. Unlike existing privacy-preserving bipartite matching protocols, our protocol does not yield a maximal matching in all cases, but allows us to keep compatible quotes as well as the quotes of both parties private. To the best of our knowledge, it is impossible to keep this information private using existing protocols. Our bartering protocol requires merely $\mathcal{O}((n_0+n_1)\log(n_0+n_1))$ computation and communication. We prove the security of our protocols in the semi-honest adversary model.

CODASPY '17, March 22–24, 2017, Scottsdale, AZ, USA.

DOI: http://dx.doi.org/10.1145/3029806.3029831

2. PRELIMINARIES

2.1 General Notations

Let $\mathbb{N} := \{1, 2, \dots\}$ denote the set of natural numbers. Then, $\mathbb{N}_i := \{1, 2, \dots, i\}$ for $i \in \mathbb{N}$. For $n \in \mathbb{N}$, Σ_n is the set of all permutations of $\mathbb{N}_n$. For a non-empty set S, $s \leftarrow_\$ S$ denotes that $s \in S$ is sampled uniformly at random from S. For $\ell \in \mathbb{N}$, $V := (V[\![1]\!], \dots, V[\![\ell]\!]) \in S^\ell$ denotes an ℓ-dimensional column vector with components $V[\![1]\!], \dots, V[\![\ell]\!] \in S$. For $n \in \mathbb{N}$ column vectors $V_i \in S^\ell$ ($i \in \mathbb{N}_n$), we write $W := [V_1 \cdots V_n] \in S^{\ell \times n}$ to denote the $\ell \times n$ matrix with columns $V_1, \dots, V_n$ and $\ell \cdot n$ entries $V_i[\![j]\!] \in S$ ($j \in \mathbb{N}_\ell$). Similarly, for matrices $W_1 \in S^{\ell \times n_1}$ and $W_2 \in S^{\ell \times n_2}$, $[W_1\ W_2] \in S^{\ell \times (n_1+n_2)}$ denotes the $\ell \times (n_1+n_2)$ matrix formed by concatenating the columns of W_1 and W_2 ($\ell, n_1, n_2 \in \mathbb{N}$). For a plaintext $m \in \mathbb{P}$ and the corresponding ciphertext $\mathrm{E}(m) \in \mathbb{C}$, we use the abbreviation $\overline{m} := \mathrm{E}(m)$.

We define $\mathbb{P}$, $\mathbb{C}$, and $\mathrm{E}(\cdot)$ for Paillier in Section 2.4. For vector $V' \in \mathbb{P}^\ell$ ($\ell \in \mathbb{N}$), $\overline{V'} \in \mathbb{C}^\ell$ is a component-wise encryption of V'. Analogously, for a matrix $W' \in \mathbb{P}^{\ell \times n}$ ($\ell, n \in \mathbb{N}$), $\overline{W}' \in \mathbb{C}^{\ell \times n}$ is an entry-wise encryption of W'. For a sequence of $n \in \mathbb{N}$ integers $x_1, \ldots, x_n \in \mathbb{Z}$, the prefix sum is the sequence $y_1, \ldots, y_n \in \mathbb{Z}$ with $y_i := \sum_{k=1}^{i} x_k$ for $i \in \mathbb{N}_n$. For $a, b \in \mathbb{Z}$ and $\star \in \{<, >, \leq, \geq, =\}$, $a \stackrel{?}{\star} b \in \{0, 1\}$ indicates whether $a \star b$ holds. The two parties participating in the protocols are denoted as P_0 and P_1.

2.2 Bipartite Matching

Following the definitions from [5], a *bipartite graph* $G := (A, B, M_0)$ is an undirected graph with nodes $A \cup B$ where $A \cap B = \emptyset$ and edges $M_0 \subseteq A \times B$. A *matching* in G is a subset of edges $M \subseteq M_0$ such that for all nodes $a \in A$ at most one edge of M is incident on a and for all nodes $b \in B$ at most one edge in M is incident on b. A node $a \in A$ (a node $b \in B$) is *matched*, if some edge in M is incident on a (on b). Otherwise, the node is *unmatched*. A *maximal matching* in G is a matching M such that for any edge $(a, b) \in M_0 \setminus M$, $M \cup \{(a, b)\}$ is not a matching. A *maximum matching* in G is a matching M such that for any matching M' it holds that $|M| \geq |M'|$.

2.3 Semantic Security

A formal treatment of semantic security is provided in [12]. The intuition behind semantic security is that adversaries should be unable to derive any partial information about the plaintext from the ciphertext [16]. There is an equivalent and much simpler definition of semantic security with regard to indistinguishable encryptions under a Chosen Plaintext Attack (CPA) [16]:

Definition 1. The $\mathrm{PubK}^{\mathrm{cpa}}_{\mathscr{A},(\mathrm{Gen},\mathrm{E},\mathrm{D})}(n)$ experiment consists of the of the following steps [16]:

1. Key generation algorithm Gen is run to obtain $(\mathrm{pk}, \mathrm{sk})$.
2. Adversary $\mathscr{A}$ is given pk as well as oracle access to encryption function $\mathrm{E}_{\mathrm{pk}}(\cdot)$. The adversary outputs a pair of messages m_0, m_1 of the same length. (These messages must be in the plaintext space associated with pk.)
3. A random bit $b \leftarrow \{0, 1\}$ is chosen, and then a ciphertext $\overline{m}_b \leftarrow \mathrm{E}_{\mathrm{pk}}(m_b)$ is computed and given to $\mathscr{A}$. We call $\overline{m}_b$ the challenge ciphertext.
4. $\mathscr{A}$ continues to have access to $\mathrm{E}_{\mathrm{pk}}(\cdot)$, and outputs a bit b'.
5. The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise.

Definition 2. A function f is *negligible* if for every polynomial $p(\cdot)$ there exists an n' such that for all integers $n > n'$ it holds that $f(n) < \frac{1}{p(n)}$. [16]

Definition 3. A public-key encryption scheme (Gen, E, D) has *indistinguishable encryptions under a chosen-plaintext attack* (or is *CPA secure*) if for all probabilistic polynomial-time adversaries $\mathscr{A}$, there exists a negligible function negl s.t. $\Pr\left(\mathrm{PubK}^{\mathrm{cpa}}_{\mathscr{A},(\mathrm{Gen},\mathrm{E},\mathrm{D})}(n) = 1\right) \leq \frac{1}{2} + \mathrm{negl}(n)$. [16]

2.4 Threshold Paillier

Our protocols require an additively homomorphic $(2, 2)$ threshold cryptosystem that is semantically secure. As the threshold variant of the Paillier cryptosystem [21] from [8] satisfies all of our requirements, we use it throughout this paper.

A (t, n) *threshold scheme* divides data d into n pieces $d_1, \ldots, d_n$ such that knowledge of any t or more d_i pieces allow the computation of d, whereas knowledge of any $t - 1$ or fewer d_i pieces leaves d undetermined [23].

We use the key generation, encryption, and decryption phases for the threshold cryptosystem by Fouque *et al.* [8].

Key Generation. Select a modulus $N = pq$ where p and q are strong primes and $p = 2p' + 1$ and $q = 2q' + 1$. Let security parameter κ denote the size of N in bits and set $N' := p'q'$. Pick $\beta \leftarrow_{\$} \mathbb{Z}^*_N$, $(a, b) \leftarrow_{\$} \mathbb{Z}^* \times \mathbb{Z}^*$ and set $g := (1 + N)^a \cdot b^N \mod N^2$. Share the private key $\beta N'$ among the two parties P_0 and P_1 using Shamir's $(2, 2)$ threshold scheme [23] such that party P_i ($i \in \{0, 1\}$) obtains share sk_i. The public key is $\mathrm{pk} := (g, N, \theta)$ where $\theta := L(g^{N'\beta}) = aN'\beta \mod N$ with $L(u) = \frac{u-1}{N}$.

Encryption. To encrypt plaintext $m \in \mathbb{P}$ in plaintext space $\mathbb{P} := \mathbb{Z}_N$, choose $r \leftarrow_{\$} \mathbb{Z}^*_N$, and compute $\mathrm{E}(m) := g^m r^N \mod N^2$. Ciphertext $\mathrm{E}(m)$ is in the ciphertext space $\mathbb{C} := \mathbb{Z}^*_{N^2}$.

Decryption. To decrypt $\overline{m} \in \mathbb{C}$, each party P_i ($i \in \{0, 1\}$) computes its decryption share $\breve{m}_i = \overline{m}^{2\Delta s_i} \mod N^2$ for $\Delta = 2$. The parties then jointly combine their shares to recover the corresponding plaintext m by computing $\mathrm{D}(\overline{m}) := L(\prod_{j \in \{0,1\}} \breve{m}_j^{2\mu_j} \mod N^2) \cdot \frac{1}{4\Delta^2\theta} \mod N = m$ with $\mu_0 = 2 \cdot \Delta$ and $\mu_1 = -\Delta$. Protocol $\pi_{(2,2)\text{-Dec}}$ denotes the corresponding threshold decryption protocol.

Properties. The plaintext space $\mathbb{P}$ and the addition $+$ modulo N form the group $(\mathbb{P}, +)$, while the ciphertext space $\mathbb{C}$ and multiplication $\cdot$ modulo N^2 form the group $(\mathbb{C}, \cdot)$. For all $\overline{m}_1$, $\overline{m}_2 \in \mathbb{C}$, threshold Paillier enables *homomorphic addition* denoted as $\overline{m}_1 +_h \overline{m}_2 := \overline{m}_1 \cdot \overline{m}_2 \mod N^2$. As we have $\mathrm{D}(\overline{m}_1 +_h \overline{m}_2) = m_1 + m_2$, the group $(\mathbb{C}, +_h)$ is additive. Threshold Paillier allows for *re-randomizing* a ciphertext $\overline{m} \in \mathbb{C}$ by computing $\mathrm{Rnd}(\overline{m}) := \overline{m} +_h \mathrm{E}(0)$ where $\mathrm{E}(0)$ is a random encryption of 0 and thus $\mathrm{Rnd}(\overline{m})$ is a random encryption of m. For security reasons, it is typically necessary to re-randomize a ciphertext that results from one or more homomorphic additions. For $\overline{V} \in \mathbb{C}^\ell$ ($\ell \in \mathbb{N}$), $\mathrm{Rnd}(\overline{V})$ denotes a component-wise re-randomization of $\overline{V}$. Similarly, for $\overline{W} \in \mathbb{C}^{\ell \times n}$ ($\ell, n \in \mathbb{N}$), $\mathrm{Rnd}(\overline{W})$ denotes the entry-wise re-randomization of $\overline{W}$.

2.5 Security Model

In the following, we present the definitions necessary to describe the security model for our protocols. We consider a semi-honest adversary, i.e., the adversary controls a set of corrupted parties which follow the protocol but attempt to learn new information from their internal coin tosses and received messages. Furthermore, we assume that the adversary is computationally bounded and non-adaptive. The latter means that the corrupted parties are fixed throughout each protocol execution. Since we consider two parties, up to one party can be corrupted. We assume authentic communication channels which prevent tampering with exchanged messages but allow eavesdropping.

The next definition characterizes the information a party gathers during a protocol execution.

Definition 4. Let π be a two-party protocol for a functionality $\mathcal{F} : \{0, 1\}^* \times \{0, 1\}^* \to \{0, 1\}^* \times \{0, 1\}^*$ on inputs (x_0, x_1) where x_i ($i \in \{0, 1\}$) is the input of party P_i. Let $\mathring{r}_i$

denote the outcome of coin tosses on the random tape of P_i and let $m_{i,j}$ ($j \in \mathbb{N}_{n_i}$ with $n_i \in \mathbb{N}$) denote the jth message P_i received during an execution of π. The *view* of P_i during an execution of π is denoted as $\text{VIEW}_i^\pi(x_0, x_1) := (x_i, \mathring{r}_i, m_{i,1}, \ldots, m_{i,n_i})$. The *output* after an execution of π on inputs (x_0, x_1) is denoted as $\text{OUTPUT}^\pi(x_0, x_1) := (\text{OUTPUT}_0^\pi(x_0, x_1), \text{OUTPUT}_1^\pi(x_0, x_1)))$ where $\text{OUTPUT}_i^\pi(x_0, x_1)$ is the output of P_i. [11]

In the following, we define our adversary model. The underlying idea is that a party following the protocol specification is unable to learn anything from executing the protocol besides what it can deduce from its own input and its protocol output.

Definition 5. Let π be a two-party protocol for a functionality $\mathcal{F} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ on inputs (x_0, x_1) where x_0 is the input of P_0 and x_1 is the input of P_1. Let $\mathcal{F}_i$ denote the output of party P_i where $i \in \{0,1\}$. Protocol π computes functionality $\mathcal{F}$ *securely in the semi-honest model* if two probabilistic polynomial-time algorithms $\mathcal{S}_0$, $\mathcal{S}_1$ exist such that for all $i \in \{0,1\}$, it holds that $\{\mathcal{S}_i(x_i, \mathcal{F}_i(x_0, x_1)), \mathcal{F}(x_0, x_1)\}_{x_0,x_1} \overset{c}{\equiv} \{\text{VIEW}_i^\pi(x_0, x_1), \text{OUTPUT}^\pi(x_0, x_1)\}_{x_0,x_1}$ where $\overset{c}{\equiv}$ denotes computational indistinguishability. [11]

Simulators $\mathcal{S}_0$ and $\mathcal{S}_1$ must run in polynomial-time in the security parameter of the underlying primitive. We assume that the security parameter is sufficiently large and omit the parameter in Definition 5 and the remainder of this paper.

Our security proofs utilize the *Sequential Composition Theorem* [3]. Assuming that a protocol π for functionality f is composed of a set of subprotocols $\rho_1, \ldots, \rho_m$ which securely implement functionalities $g_1, \ldots, g_m$, the theorem states that if a protocol π'—which replaces the subprotocol calls $\rho_1, \ldots, \rho_m$ of π by calls of a trusted third party computing $g_1, \ldots, g_m$—securely computes f, then π securely implements f. The calls of the trusted third party must happen sequentially rather than concurrently.

Throughout this paper, we write $(o_0, o_1) \leftarrow \mathcal{F}(x_0, x_1)$ to denote that on input x_i ($i \in \{0,1\}$) from party P_i, functionality $\mathcal{F}$ provides P_i with output o_i. In case both parties provide input x and both parties receive output o, we write $(o) \leftarrow \mathcal{F}(x)$. We use analogous notation for a protocol π.

3. INTUITION AND APPROACH

The goal of our bartering protocol is to determine a set of simultaneously executable quotes based on the quotes from P_0 and P_1 in a privacy-preserving manner. By viewing the offered and demanded commodities from P_0 as nodes in a graph where an offered and a demanded commodity is connected by an edge iff the commodities form a quote from P_0 for which a compatible quote from P_1 exists, we can reduce the problem of selecting simultaneously executable quotes to bipartite matching.[1] To obtain this graph, we need to first find the quotes from P_0 which are compatible with a quote from P_1. With P_0 and P_1 having a set of quotes, we are able to reduce this problem to the well-studied set intersection problem. Consequently, the components of our privacy-preserving bartering protocol are privacy-preserving protocols for set intersection and bipartite matching.

Existing privacy-preserving set intersection protocols typically operate on integer elements. To make these protocols work with quotes, we encode each commodity and each quote as distinct integers. Specifically, the encoding of quotes is such that an encoded quote from P_0 is equal to an encoded quote from P_1 iff the quotes are compatible. Moreover, in bartering where the intersection corresponds to compatible quotes, the intersection is an intermediate result that must be kept private. To enable the further use of the intermediate result, we augment each quote with an encrypted indicator signifying quote compatibility. To implement set intersection, we choose the protocol by Huang *et al.* [13] as a basis owing to its efficiency. Furthermore, the generic design of the garbled circuit-based protocol allows us to adapt the protocol and implement it using threshold homomorphic encryption.

The main challenge for bipartite matching is to keep the compatible quotes and the quotes of both parties private at all times. In our approach, we leverage the fact that although P_0 must not learn which of its quotes are compatible, it naturally knows its own quotes. Thus, we strive to base our protocol on a suitable non-privacy-preserving matching algorithm. Candidates include the Hungarian algorithm or the flow network-based bipartite matching algorithm from [5]. However, to make these algorithms privacy-preserving, both parties would at least need to know the number of nodes (corresponding to the commodities from P_0), or the number of edges (corresponding to the compatible quotes), or both. As we must not disclose this information, to the best of our knowledge it is impossible to achieve our privacy requirements using these algorithms. To keep the information that these algorithms disclose by construction private, we base our approach on online bipartite matching. In online bipartite matching, the edges of the graph are revealed step-by-step rather than at once. The algorithm makes irreversible matching decisions when new edges are revealed. We use the online bipartite matching algorithm OBLIVIOUS by Mastin and Jaillet [18] as a starting point. OBLIVIOUS takes a bipartite graph $G = (A, B, M_0)$ as its input (edges are revealed iteratively per node $a \in A$) and outputs a matching in G. When the edges for a node $a \in A$ are revealed, the algorithm matches node a to a random node $b' \in \{b \in B \mid (a, b) \in M_0\}$, as long as b' is unmatched. If b' was matched before, OBLIVIOUS never matches node a.

OBLIVIOUS potentially leaks information about the edges to a party which learns the order in which the algorithm attempts to match the nodes $a \in A$. To illustrate this leakage, consider the bipartite graph $(\{a_1, a_2\}, \{b_1\}, \{(a_2, b_1)\})$ for which the output of OBLIVIOUS is $\{(a_2, b_1)\}$. If a party learns that OBLIVIOUS attempted to match a_1 before a_2, then the party can deduce that edge (a_1, b_1) does not exist. To overcome this issue, we derive the non-privacy-preserving matching algorithm PRUNE (Algorithm 1). PRUNE also allows us to leverage the fact that P_0 knows A and B but not the edges between them.

To implement PRUNE in a privacy-preserving fashion such that the compatible quotes are not disclosed to any party and the quotes from P_0 are kept private from P_1, both parties first obliviously shuffle the encrypted quotes from P_0. Then, they jointly decrypt the encrypted offered commodi-

[1]Note that in case the parties hold different numbers of quotes, lower complexities are achievable if the party with fewer quotes assumes the role of P_0.

Algorithm 1: PRUNE

Input: A bipartite graph $G = (A, B, M_0 \subseteq A \times B)$
Output: Matching M_2 in G

$M_1 := \emptyset$
for $a \in A$ **do**
  $N_1(a) := \{b \in B \mid (a,b) \in M_0\}$
  if $|N_1(a)| > 0$ **then**
    $b \leftarrow_\$ N_1(a)$
    $M_1 := M_1 \cup \{(a,b)\}$
$M_2 := \emptyset$
for $b \in B$ **do**
  $N_2(b) := \{a \in A \mid (a,b) \in M_1\}$
  if $|N_2(b)| > 0$ **then**
    $a \leftarrow_\$ N_2(b)$
    $M_2 := M_2 \cup \{(a,b)\}$
return M_2

ties such that only P_0 obtains the result. As both parties shuffle the quotes and only the offered commodities are decrypted (which P_0 already knows), P_0 will not learn the order among the quotes with the same offered commodity. For each distinct offered commodity, the parties obliviously select only the leftmost compatible quote by updating the encrypted quote compatibility indicators. In doing so, P_0 obliviously computes the prefix sum over the indicators of each offer-subsequence using homomorphic addition. For each quote, the parties then obliviously compute an indicator signifying whether both the corresponding prefix sum value is 1 and the corresponding quote compatibility indicator is 1. This idea of expressing the indicator condition using a prefix sum allows us to save communication, since P_0 can compute the prefix sums locally. As the quotes are shuffled, the parties select a random compatible quote for each distinct offered commodity. This procedure implements the first loop of PRUNE. We implement the second loop using the same idea. Finally, the parties obliviously filter the quotes w.r.t. the indicator to obtain the simultaneously executable quotes. To achieve low complexities for bipartite matching and the oblivious shuffling building block, we chose to implement our bartering protocol using threshold homomorphic encryption.

In the full version of this paper we show that PRUNE and OBLIVIOUS yield matchings of the same size (the matchings are potentially different), provided that both algorithms make the same random choices when they attempt to match a node $a \in A$. Consequently, we can apply the matching size analysis for OBLIVIOUS from [18] also to PRUNE.

4. BUILDING BLOCKS

This section discusses both existing building blocks and introduces novel building blocks used as part of our novel bartering protocol. We emphasize that if the input of a protocol is encrypted, then the parties are *not* required to know the corresponding plaintext. If the output of a protocol is encrypted, then a party will *not* learn the corresponding plaintext (unless the party can derive the plaintext from its input). In the following, all proofs are omitted and deferred to the full version of this paper.

4.1 Existing Building Blocks

Comparison. Functionality $\mathcal{F}_{\text{LT}}^{\text{CI-SO}}$ takes ciphertexts $\overline{m}_0$ and $\overline{m}_1 \in \mathbb{C}$ as its input and provides each party with an XOR-share of the bit $m_0 <^? m_1$.

Definition 6 ($\mathcal{F}_{\text{LT}}^{\text{CI-SO}}$: Less Than (LT) Comparison with Ciphertext Input (CI) and Shared Output (SO) [19])**.** Let P_0 and P_1 both hold encryptions $\overline{m}_0$ and $\overline{m}_1$ of $m_0, m_1 \in \mathbb{P}$ as their inputs. Then, functionality $\mathcal{F}_{\text{LT}}^{\text{CI-SO}}$ is given by $(b^{(0)}, b^{(1)}) \leftarrow \mathcal{F}_{\text{LT}}^{\text{CI-SO}}((\overline{m}_0, \overline{m}_1))$ with $b^{(0)} \leftarrow_\$ \{0,1\}$ such that $b^{(1)} := b^{(0)} \oplus (m_0 <^? m_1)$.

Owing to the properties of XOR and $<^?$, $b^{(0)}$ and $b^{(1)}$ in Definition 6 are chosen such that we have $b^{(0)} \oplus b^{(1)} = 1 \Leftrightarrow m_0 < m_1$. To allow for the comparing of plaintexts, we assume that plaintexts are represented by their congruent integer in $\{0, 1, \ldots, |\mathbb{P}|-1\}$. We refer to the variants of $\mathcal{F}_{\text{LT}}^{\text{CI-SO}}$ for Greater Than (GT), Less Than or Equal to (LTE), and Greater Than or Equal to (GTE) as $\mathcal{F}_{\text{GT}}^{\text{CI-SO}}$, $\mathcal{F}_{\text{LTE}}^{\text{CI-SO}}$, and $\mathcal{F}_{\text{GTE}}^{\text{CI-SO}}$, respectively. The definitions for $\mathcal{F}_{\text{GT}}^{\text{CI-SO}}$, $\mathcal{F}_{\text{LTE}}^{\text{CI-SO}}$, $\mathcal{F}_{\text{GTE}}^{\text{CI-SO}}$ follow from replacing $<^?$ by $>^?$, $\leq^?$, and $\geq^?$, respectively, in Definition 6.

A protocol implementing $\mathcal{F}_{\text{LT}}^{\text{CI-SO}}$ is introduced in [19] which extends the comparison protocol from [17] with shared output. Both protocols enforce an upper bound on the plaintexts corresponding to the encrypted inputs based on $(2,2)$ threshold Paillier.

Oblivious Merge. The input of functionality $\mathcal{F}_{\text{O-Merge}}^{j^*,k^*}$ is an encrypted matrix with n columns such that the first k^* columns and the last $n-k^*$ columns are each in ascending order with respect to the plaintexts of the j^*th row. The functionality reorders the columns of the input matrix such that all columns of the re-randomized output matrix are in ascending order with respect to the plaintexts of the j^*th row.

Definition 7 ($\mathcal{F}_{\text{O-Merge}}^{j^*,k^*}$: Oblivious Merge)**.** Let P_0 and P_1 hold $[\overline{V}_1 \cdots \overline{V}_n]$ with $\overline{V}_i \in \mathbb{C}^\ell$ $(i \in \mathbb{N}_n)$ such that $V_1[\![j^*]\!] \leq \cdots \leq V_{k^*}[\![j^*]\!]$ and $V_{k^*+1}[\![j^*]\!] \leq \cdots \leq V_n[\![j^*]\!]$ where $k^* \in \mathbb{N}_{n-1}$ and $j^* \in \mathbb{N}_\ell$ are fixed. Then, functionality $\mathcal{F}_{\text{O-Merge}}^{j^*,k^*}$ is given by $(\text{Rnd}([\overline{V}_{\sigma(1)} \cdots \overline{V}_{\sigma(n)}])) \leftarrow \mathcal{F}_{\text{O-Merge}}^{j^*,k^*}([\overline{V}_1 \cdots \overline{V}_n])$ with $\sigma \in \Sigma_n$ such that $V_{\sigma(1)}[\![j^*]\!] \leq \cdots \leq V_{\sigma(n)}[\![j^*]\!]$.

Jónsson *et al.* [14] present a secret sharing-based protocol for $\mathcal{F}_{\text{O-Merge}}^{j^*,k^*}$ named odd-even-merge. Implementing the protocol from [14] using homomorphic encryption requires replacing the compare-exchange building block by the oblivious greater than swap protocol from [27].

Oblivious Shuffle. Given an encrypted matrix as its input, functionality $\mathcal{F}_{\text{O-Shuffle}}$ randomly shuffles the columns and re-randomizes all entries.

Definition 8 ($\mathcal{F}_{\text{O-Shuffle}}$: Oblivious Shuffle [27])**.** Let P_0 and P_1 hold $[\overline{V}_1 \cdots \overline{V}_n]$ with $\overline{V}_i \in \mathbb{C}^\ell$ $(i \in \mathbb{N}_n)$. Then, functionality $\mathcal{F}_{\text{O-Shuffle}}$ is given by $(\text{Rnd}([\overline{V}_{\sigma(1)} \cdots \overline{V}_{\sigma(n)}]))$ $\leftarrow \mathcal{F}_{\text{O-Shuffle}}([\overline{V}_1 \cdots \overline{V}_n])$ with $\sigma \leftarrow_\$ \Sigma_n$.

Oblivious Filtering. The input of functionality $\mathcal{F}_{\text{O-Filter}}^{j^*}$ is an encrypted matrix where the plaintexts corresponding to the entries in the j^*th row are either 0 or 1. The functionality determines a matrix which contains only the columns from the input matrix where the entry in the j^*th row is 1.

The resulting matrix does not include the j^*th row of the input matrix.

Definition 9 ($\mathcal{F}^{j^*}_{\text{O-Filter}}$: Oblivious Filtering)**.** Let P_0 and P_1 hold $[\overline{V}_1 \cdots \overline{V}_n]$ with $\overline{V}_i \in \mathbb{C}^\ell$ and $V_i[\![j^*]\!] \in \{0,1\}$ ($i \in \mathbb{N}_n$) for a fixed $j^* \in \mathbb{N}_\ell$. Then, functionality $\mathcal{F}^{j^*}_{\text{O-Filter}}$ is given by $(\text{Rnd}([\overline{V}'_{\sigma(1)} \cdots \overline{V}'_{\sigma(n')}])) \leftarrow \mathcal{F}^{j^*}_{\text{O-Filter}}([\overline{V}_1 \cdots \overline{V}_n])$ with $n' := \#\{i \in \mathbb{N}_n \mid V_i[\![j^*]\!] = 1\}$, $\sigma \leftarrow_\$ \Sigma_{n'}$, and $\{\overline{V}'_1, \ldots, \overline{V}'_{n'}\} = \{(\overline{V}_i[\![1]\!], \ldots, \overline{V}_i[\![j^*-1]\!], \overline{V}_i[\![j^*+1]\!], \ldots, \overline{V}_i[\![\ell]\!]) \in \mathbb{C}^{\ell-1} \mid i \in \mathbb{N}_n \wedge V_i[\![j^*]\!] = 1\}$.

Protocols for $\mathcal{F}_{\text{O-Shuffle}}$ and $\mathcal{F}^{j^*}_{\text{O-Filter}}$ based on $(2,2)$ threshold Paillier are presented by Wüller *et al.* in [27].

Shared Permutation. Functionality $\mathcal{F}_{\text{SP}}$ receives the same ciphertext pair from each party as its input. In addition, $\mathcal{F}_{\text{SP}}$ receives a share of a bit from each party as part of the input. If the bit is 0, $\mathcal{F}_{\text{SP}}$ outputs a re-randomization of the ciphertext pair. Otherwise, $\mathcal{F}_{\text{SP}}$ swaps and re-randomizes the components of the ciphertext pair and outputs the result.

Definition 10 ($\mathcal{F}_{\text{SP}}$: Shared Permutation (SP))**.** Let P_0 and P_1 hold $(\overline{m}_0, \overline{m}_1) \in \mathbb{C}^2$. In addition, let P_j ($j \in \{0,1\}$) hold bit $b^{(j)} \in \{0,1\}$ as its input. Then, functionality $\mathcal{F}_{\text{SP}}$ is given by $(\text{Rnd}((\overline{m}_b, \overline{m}_{1-b}))) \leftarrow \mathcal{F}_{\text{SP}}((\overline{m}_0, \overline{m}_1, b^{(0)}), (\overline{m}_0, \overline{m}_1, b^{(1)}))$ where $b := b^{(0)} \oplus b^{(1)}$.

In conjunction with a comparison protocol with shared output, shared permutation allows us to implement oblivious swapping [27] and oblivious predicate evaluations such as an equality test. It is possible to derive a protocol for $\mathcal{F}_{\text{SP}}$ from the main protocol in [7]: P_0 sends $\text{Rnd}((\overline{m}_0, \overline{m}_1))$ to P_1 iff $b^{(0)} =^? 0$. Otherwise, P_0 sends $\text{Rnd}((\overline{m}_1, \overline{m}_0))$. P_1 proceeds analogously based on the received ciphertexts as well as its bit $b^{(1)}$ and eventually transmits the result to P_0.

4.2 Novel Building Blocks

4.2.1 Oblivious Duplicate Marker

Functionality. $\mathcal{F}^{j^*}_{\text{O-Dup}}$ takes an encrypted matrix as its input where the plaintexts corresponding to the entries in the j^*th row are in ascending order and determines duplicates among these plaintexts. In doing so, it obliviously checks all adjacent entry-pairs for equality, thereby generating encrypted rows $l+1$ and $l+2$. For an entry in the j^*th row, the corresponding entries in rows $l+1$ and $l+2$ indicate a duplicate with the left and right j^*th row neighbor, respectively.

Definition 11 ($\mathcal{F}^{j^*}_{\text{O-Dup}}$: Oblivious Duplicate Marker)**.** Let P_0 and P_1 hold $\overline{V} := [\overline{V}_1 \cdots \overline{V}_n] \in \mathbb{C}^{\ell \times n}$ such that for a fixed $j^* \in \mathbb{N}_\ell$, it holds that $V_1[\![j^*]\!] \leq \cdots \leq V_n[\![j^*]\!]$. Then, functionality $\mathcal{F}^{j^*}_{\text{O-Dup}}$ is given by $(\overline{V}') \leftarrow \mathcal{F}^{j^*}_{\text{O-Dup}}(\overline{V})$ with $\overline{V}' := [\overline{V}'_1 \cdots \overline{V}'_n] \in \mathbb{C}^{(\ell+2)\times n}$ such that $\overline{V}'_i[\![j]\!] := \overline{V}_i[\![j]\!]$ ($i \in \mathbb{N}_n, j \in \mathbb{N}_\ell$), $\overline{V}'_{i'}[\![\ell+1]\!] := \bar{b}^{(0,1)}_{i'}$ ($i' \in \mathbb{N}_n \setminus \{1\}$), $\overline{V}'_{i''}[\![\ell+2]\!] := \bar{b}^{(0,1)}_{i''+1}$ ($i'' \in \mathbb{N}_{n-1}$), and $(\overline{V}'_1[\![\ell+1]\!], \overline{V}'_n[\![\ell+2]\!]) := (\bar{b}_0, \bar{b}_0)$ where $\bar{b}^{(0,1)}_{i'} := \text{E}(V_{i'-1}[\![j^*]\!] =^? V_{i'}[\![j^*]\!])$ and $\bar{b}_0 := \text{E}(0)$.

Protocol. Implementing $\mathcal{F}^{j^*}_{\text{O-Dup}}$ requires an oblivious equality check for each pair of adjacent entries in the j^*th row of the input matrix. The oblivious equality check must yield a ciphertext of the result to not leak intermediate results when used in bartering. The idea is to express equality as the conjunction of a $\leq^?$ and a $\geq^?$ comparison. Thus, the parties execute $\pi^{\text{CI-SO}}_{\text{LTE}}$ as well as $\pi^{\text{CI-SO}}_{\text{GTE}}$ and combine the shared outputs using π_{SP} to determine a ciphertext of the equality check.[2] In Steps 4 to 7, each party locally builds the output from these ciphertexts and the input.

$\boldsymbol{P_0}$: $[\overline{V}_1 \cdots \overline{V}_n] \in \mathbb{C}^{\ell \times n}$	$\boldsymbol{P_1}$: $[\overline{V}_1 \cdots \overline{V}_n] \in \mathbb{C}^{\ell \times n}$
1. $(\bar{b}_0, \bar{b}_1) := (\text{E}(0), \text{E}(1))$	
	2. $\xrightarrow{\bar{b}_0, \bar{b}_1}$
3. **for** $i' \in \mathbb{N}_n \setminus \{1\}$	
$(b^{(0)}_{i',1}, b^{(1)}_{i',1}) \leftarrow \pi^{\text{CI-SO}}_{\text{LTE}}(\overline{V}_{i'-1}[\![j^*]\!], \overline{V}_{i'}[\![j^*]\!])$	
$(b^{(0)}_{i',2}, b^{(1)}_{i',2}) \leftarrow \pi^{\text{CI-SO}}_{\text{GTE}}(\overline{V}_{i'-1}[\![j^*]\!], \overline{V}_{i'}[\![j^*]\!])$	
$((\bar{b}''^{(0,1)}_{i'}, \bar{b}'''^{(0,1)}_{i})) \leftarrow \pi_{\text{SP}}((\bar{b}_0, \bar{b}_1, b^{(0)}_{i',1}), (\bar{b}_0, \bar{b}_1, b^{(1)}_{i',1}))$	
$((\bar{b}^{(0,1)}_{i'}, \bar{b}'^{(0,1)}_{i'})) \leftarrow \pi_{\text{SP}}((\bar{b}_0, \bar{b}''^{(0,1)}_{i'}, b^{(0)}_{i',2}), (\bar{b}_0, \bar{b}''^{(0,1)}_{i'}, b^{(1)}_{i',2}))$	
end	
/ Each party locally builds the output: */*	
4. $\forall i \in \mathbb{N}_n \forall j \in \mathbb{N}_\ell$, $\overline{V}'_i[\![j]\!] := \overline{V}_i[\![j]\!]$	
5. $\forall i' \in \mathbb{N}_n \setminus \{1\}$, $\overline{V}'_{i'}[\![\ell+1]\!] := \bar{b}^{(0,1)}_{i'}$	
6. $\forall i'' \in \mathbb{N}_{n-1}$, $\overline{V}'_{i''}[\![\ell+2]\!] := \bar{b}^{(0,1)}_{i''+1}$	
7. $(\overline{V}'_1[\![\ell+1]\!], \overline{V}'_n[\![\ell+2]\!]) := (\bar{b}_0, \bar{b}_0)$	
8. **output** $[\overline{V}'_1 \cdots \overline{V}'_n]$	

Protocol 1: $\pi^{j^*}_{\text{O-Dup}}$.

Lemma 1. *Let P_0 and P_1 hold $[\overline{V}_1 \cdots \overline{V}_n] \in \mathbb{C}^{\ell \times n}$ such that for a fixed $j^* \in \mathbb{N}_l$, it holds that $V_1[\![j^*]\!] \leq \cdots \leq V_n[\![j^*]\!]$. Then, $\pi^{j^*}_{\text{O-Dup}}$ computes functionality $\mathcal{F}^{j^*}_{\text{O-Dup}}$ securely in the semi-honest model.*

Theorem 1. *If the subprotocols of $\pi^{j^*}_{\text{O-Dup}}$ are implemented as outlined above and composed as in $\pi^{j^*}_{\text{O-Dup}}$, then the communication and round complexities of $\pi^{j^*}_{\text{O-Dup}}$ are in $\mathcal{O}(n)$ and its computation complexity is in $\mathcal{O}(n \cdot \ell)$.*

4.2.2 Oblivious Prune

Functionality. Given $[\overline{V}_1 \cdots \overline{V}_n] \in \mathbb{C}^{\ell \times n}$ and $\{(V_i[\![j^*_1]\!], V_i[\![j^*_2]\!]) \mid i \in \mathbb{N}_n\}$ as the input, $\mathcal{F}^{j^*_1, j^*_2, j^*}_{\text{O-Prune}}$ updates $V_1[\![j^*]\!], \ldots, V_n[\![j^*]\!]$ such that $\{(V_i[\![j^*_1]\!], V_i[\![j^*_2]\!]) \mid i \in \mathbb{N}_n \wedge V'_i[\![j^*]\!] = 1\}$ is a matching in the bipartite graph $(\{V_i[\![j^*_1]\!] \mid i \in \mathbb{N}_n\}, \{V_i[\![j^*_2]\!] \mid i \in \mathbb{N}_n\}, \{(V_i[\![j^*_1]\!], V_i[\![j^*_2]\!]) \mid i \in \mathbb{N}_n \wedge V_i[\![j^*]\!] = 1\})$.

Definition 12 ($\mathcal{F}^{j^*_1, j^*_2, j^*}_{\text{O-Prune}}$: Oblivious Prune)**.** Let j^*_1, j^*_2, $j^* \in \mathbb{N}_\ell$ be fixed and pairwise different. Let P_0 and P_1 hold $\overline{V} := [\overline{V}_1 \cdots \overline{V}_n] \in \mathbb{C}^{\ell \times n}$ such that $V_i[\![j^*]\!] \in \{0,1\}$ ($i \in \mathbb{N}_n$) and $G := (\{V_i[\![j^*_1]\!] \mid i \in \mathbb{N}_n\}, \{V_i[\![j^*_2]\!] \mid i \in \mathbb{N}_n\}, \{(V_i[\![j^*_1]\!], V_i[\![j^*_2]\!]) \mid i \in \mathbb{N}_n \wedge V_i[\![j^*]\!] = 1\})$ is a bipartite graph. Let P_0 additionally hold $V^{(j^*_1, j^*_2)} := \{(V_i[\![j^*_1]\!], V_i[\![j^*_2]\!]) \mid i \in \mathbb{N}_n\}$. Then, functionality $\mathcal{F}^{j^*_1, j^*_2, j^*}_{\text{O-Prune}}$ is given by $(\text{Rnd}([\overline{V}'_{\sigma(1)} \cdots \overline{V}'_{\sigma(n)}])) \leftarrow \mathcal{F}^{j^*_1, j^*_2, j^*}_{\text{O-Prune}}((\overline{V}, V^{(j^*_1, j^*_2)}), \overline{V})$ with $\sigma \leftarrow_\$ \Sigma_n$

[2] Our privacy-preserving equality test protocol supports (i) threshold Paillier, (ii) ciphertext input, and (iii) ciphertext output. Moreover, it is (iv) correct with non-negligible probability and (v) efficient in terms of complexity. Existing equality test protocols such as [25] do not exhibit all of these properties.

$\boldsymbol{P_0}$: $[\overline{V}_1 \cdots \overline{V}_n] \in \mathbb{C}^{\ell \times n}$; $\{(V_i[\![j_1^*]\!], V_i[\![j_2^*]\!]) \mid i \in \mathbb{N}_n\}$	$\boldsymbol{P_1}$: $[\overline{V}_1 \cdots \overline{V}_n] \in \mathbb{C}^{\ell \times n}$
1. $[\overline{V}'_1 \cdots \overline{V}'_n] := [\overline{V}_1 \cdots \overline{V}_n]$	$[\overline{V}'_1 \cdots \overline{V}'_n] := [\overline{V}_1 \cdots \overline{V}_n]$
2a. **for** $b = 1$ **to** 2	
$([\overline{V}_{1,b} \cdots \overline{V}_{n,b}]) \leftarrow \pi_{\text{O-Shuffle}}([\overline{V}'_1 \cdots \overline{V}'_n])$	
2b. **for** $i \in \mathbb{N}_n$	
$(v_i^{(j_b^*)}, \bot) \leftarrow \pi_{(2,2)\text{-Dec}}(\overline{V}_{i,b}[\![j_b^*]\!])$	
end	
2c. $(\bar{s}_{1,b}^{(j^*)}, \ldots, \bar{s}_{n,b}^{(j^*)}, \bar{b}_0, \bar{b}_1) := (\mathrm{E}(0), \ldots, \mathrm{E}(0), \mathrm{E}(0), \mathrm{E}(1))$	
2d. **for** $v^{(j_b^*)} \in \{v_1^{(j_b^*)}, \ldots, v_n^{(j_b^*)}\}$	
$\bar{s} := \mathrm{E}(0)$	
$I := \{i \in \mathbb{N}_n \mid v_i^{(j_b^*)} = v^{(j_b^*)}\}$	
for $i \in I$ **from** $\min(I)$ **to** $\max(I)$	
$\bar{s}_{i,b}^{(j^*)} := \overline{V}_{i,b}[\![j^*]\!] +_h \bar{s}$	
$\bar{s} := \bar{s}_{i,b}^{(j^*)}$	
end	
end	
2e. $(\bar{s}_{1,b}'^{(j^*)}, \ldots, \bar{s}_{n,b}'^{(j^*)}) := \mathrm{Rnd}((\bar{s}_{1,b}^{(j^*)}, \ldots, \bar{s}_{n,b}^{(j^*)}))$	
2f. $\bar{s}_{1,b}'^{(j^*)}, \ldots, \bar{s}_{n,b}'^{(j^*)}, \bar{b}_0, \bar{b}_1 \longrightarrow$	
2g. **for** $i \in \mathbb{N}_n$	
$(\beta_{i,b}^{(0)}, \beta_{i,b}^{(1)}) \leftarrow \pi_{\text{GT}}^{\text{CI-SO}}((\bar{s}_{i,b}'^{(j^*)}, \bar{b}_1))$	
$((\overline{V}_{i,b}^{(j^*)}, \overline{V}_{i,b}'^{(j^*)})) \leftarrow \pi_{\text{SP}}((\overline{V}_{i,b}[\![j^*]\!], \bar{b}_0, \beta_{i,b}^{(0)}), (\overline{V}_{i,b}[\![j^*]\!], \bar{b}_0, \beta_{i,b}^{(1)}))$	
2h. $\overline{V}_{i,b}[\![j^*]\!] := \overline{V}_{i,b}^{(j^*)}$	$\overline{V}_{i,b}[\![j^*]\!] := \overline{V}_{i,b}^{(j^*)}$
end	
2i. $[\overline{V}'_1 \cdots \overline{V}'_n] := [\overline{V}_{1,b} \cdots \overline{V}_{n,b}]$	$[\overline{V}'_1 \cdots \overline{V}'_n] := [\overline{V}_{1,b} \cdots \overline{V}_{n,b}]$
end	
3. $([\overline{V}''_1 \cdots \overline{V}''_n]) \leftarrow \pi_{\text{O-Shuffle}}([\overline{V}'_1 \cdots \overline{V}'_n])$	
4. **output** $[\overline{V}''_1 \cdots \overline{V}''_n]$	**output** $[\overline{V}''_1 \cdots \overline{V}''_n]$

Protocol 2: $\pi_{\text{O-Prune}}^{j_1^*, j_2^*, j^*}$.

and $\overline{V}'_i := (\overline{V}_i[\![1]\!], \ldots, \overline{V}_i[\![j^* - 1]\!], \overline{V}'_i[\![j^*]\!], \overline{V}_i[\![j^* + 1]\!], \ldots, \overline{V}_i[\![\ell]\!])$ $(i \in \mathbb{N}_n)$ where $V'_i[\![j^*]\!] \in \{0, 1\}$ such that $\{(V_i[\![j_1^*]\!], V_i[\![j_2^*]\!]) \mid i \in \mathbb{N}_n \wedge V'_i[\![j^*]\!] = 1\}$ is some matching in G.

Protocol. $\pi_{\text{O-Prune}}^{j_1^*, j_2^*, j^*}$ is a privacy-preserving implementation of PRUNE (Algorithm 1). In the bth iteration ($b \in \{1, 2\}$) of the outer for-loop, which comprises Steps 2a to 2i of Protocol 2, the parties match each node in $\{V_i[\![j_b^*]\!] \mid i \in \mathbb{N}_n\}$ to a random node contained in $\{V_i[\![j_{2-b}^*]\!] \mid i \in \mathbb{N}_n\}$. In Steps 2c to 2e of Protocol 2, P_0 locally computes the encrypted prefix sums which the parties use in Steps 2g to 2i to obliviously update the indicators $V_1[\![j^*]\!], \ldots, V_n[\![j^*]\!]$.[3] In Step 3, the parties shuffle the columns to prevent the parties from deriving any information about the original edges if $\pi_{\text{O-Prune}}^{j_1^*, j_2^*, j^*}$ is used as a subprotocol and its output is decrypted.

Lemma 2. *Let j_1^*, j_2^*, $j^* \in \mathbb{N}_\ell$ be fixed and pairwise different. Let P_0 and P_1 hold $\overline{V} := [\overline{V}_1 \cdots \overline{V}_n] \in \mathbb{C}^{\ell \times n}$ such that $V_i[\![j^*]\!] \in \{0, 1\}$ $(i \in \mathbb{N}_n)$ and $(\{V_i[\![j_1^*]\!] \mid i \in \mathbb{N}_n\}, \{V_i[\![j_2^*]\!] \mid i \in \mathbb{N}_n\}, \{(V_i[\![j_1^*]\!], V_i[\![j_2^*]\!]) \mid i \in \mathbb{N}_n \wedge V_i[\![j^*]\!] = 1\})$ is a bipartite graph. Let P_0 additionally hold $V^{(j_1^*, j_2^*)} := \{(V_i[\![j_1^*]\!], V_i[\![j_2^*]\!]) \mid i \in \mathbb{N}_n\}$. Then, $\pi_{\text{O-Prune}}^{j_1^*, j_2^*, j^*}$ computes functionality $\mathcal{F}_{\text{O-Prune}}^{j_1^*, j_2^*, j^*}$ securely in the semi-honest model.*

Theorem 2. *If the subprotocols of $\pi_{\text{O-Prune}}^{j_1^*, j_2^*, j^*}$ are implemented as outlined above and composed as in $\pi_{\text{O-Prune}}^{j_1^*, j_2^*, j^*}$, then the communication and computation complexities of $\pi_{\text{O-Prune}}^{j_1^*, j_2^*, j^*}$ are in $\mathcal{O}(n \cdot \ell)$ and its round complexity is in $\mathcal{O}(n)$.*[4]

5. BARTERING PROTOCOL

A *commodity* c is a good or a service from the *domain of commodities* $\mathscr{C} := \{c_1, \ldots, c_i, \ldots, c_k\}$. We encode commodity c_i using its unique index $i \in \mathbb{N}_k$. We assume that the domain of commodities $\mathscr{C}$ is public.

A *quote* $\gamma^{(j)}$ of P_j ($j \in \{0, 1\}$) is a pair $\gamma^{(j)} = (c_o^{(j)}, c_d^{(j)})$ of an *offered commodity* $c_o^{(j)} \in \mathbb{N}_k$ and a *demanded commodity* $c_d^{(j)} \in \mathbb{N}_k$. Quote $\gamma^{(j)}$ expresses that P_j is willing to give $c_o^{(j)}$ in exchange for receiving $c_d^{(j)}$.

Quote $\gamma^{(j)} = (c_o^{(j)}, c_d^{(j)})$ ($j \in \{0, 1\}$) and quote $\gamma^{(1-j)} = (c_o^{(1-j)}, c_d^{(1-j)})$ are *compatible* iff it holds that $(c_o^{(j)} = c_d^{(1-j)}) \wedge (c_d^{(j)} = c_o^{(1-j)})$. Otherwise, $\gamma^{(j)}$ and $\gamma^{(1-j)}$ are *incompatible*. The bijective functions $f_0 : \mathbb{N}_k^2 \to \mathbb{N}_{k^2}, (c_o, c_d) \mapsto (c_o - 1) \cdot k + c_d$ used by P_0 and $f_1 : \mathbb{N}_k^2 \to \mathbb{N}_{k^2}, (c_o, c_d) \mapsto (c_d - 1) \cdot k + c_o$ used by P_1 encode a quote as an integer, enabling us to check quote compatibility using an integer equality test.

The *set of quotes* from P_j is denoted as $\Gamma^{(j)} = \{(c_{o,1}^{(j)}, c_{d,1}^{(j)}), \ldots, (c_{o,n_j}^{(j)}, c_{d,n_j}^{(j)})\}$ where $\Gamma_o^{(j)} := \{c_{o,1}^{(j)}, \ldots, c_{o,n_j}^{(j)}\}$ ($\Gamma_d^{(j)} := \{c_{d,1}^{(j)}, \ldots, c_{d,n_j}^{(j)}\}$) is the *set of offered* (*demanded*) *commodities* associated with $\Gamma^{(j)}$.[5]

Given the sets of quotes $\Gamma^{(0)}$ and $\Gamma^{(1)}$, the *set of compatible quotes* $\Gamma_\cap^{(0,1)}$ from the perspective of P_0 is the set $\Gamma_\cap^{(0,1)} := \{(c_o^{(0)}, c_d^{(0)}) \in \Gamma^{(0)} \mid \exists (c_o^{(1)}, c_d^{(1)}) \in \Gamma^{(1)} : (c_o^{(0)} = c_d^{(1)}) \wedge (c_d^{(0)} = c_o^{(1)})\}$. Using f_0 and f_1, we can rewrite $\Gamma_\cap^{(0,1)}$ as $\Gamma_\cap^{(0,1)} = \{(c_o^{(0)}, c_d^{(0)}) \in \Gamma^{(0)} \mid \exists (c_o^{(1)}, c_d^{(1)}) \in \Gamma^{(1)} : f_0(c_o^{(0)}, c_d^{(0)}) = f_1(c_o^{(1)}, c_d^{(1)})\}$.

For the sets of quotes $\Gamma^{(0)}$ and $\Gamma^{(1)}$, a *set of simultaneously executable quotes* from the perspective of P_0 is a subset of compatible quotes $\Gamma_{||}^{(0,1)} \subseteq \Gamma_\cap^{(0,1)}$ such that all offered commodities that are part of a quote in $\Gamma_{||}^{(0,1)}$ are pairwise different and the same holds for all demanded commodities.

Given $\Gamma^{(0)}$ and $\Gamma^{(1)}$ as its input, $\mathcal{F}_{\text{BSEQ}}$ computes a set of simultaneously executable quotes $\Gamma_{||}^{(0,1)}$.

Definition 13 ($\mathcal{F}_{\text{BSEQ}}$: Bartering with Simultaneously Executable Quotes (BSEQ))**.** Let P_j ($j \in \{0, 1\}$) hold the set of quotes $\Gamma^{(j)}$ with $|\Gamma^{(j)}| = n_j$. Additionally, let P_j hold n_{1-j}. Then, functionality $\mathcal{F}_{\text{BSEQ}}$ is given by $(\Gamma_{||}^{(0,1)}) \leftarrow \mathcal{F}_{\text{BSEQ}}(\Gamma^{(0)}, \Gamma^{(1)})$ where $\Gamma_{||}^{(0,1)}$ is a set of simultaneously executable quotes from the perspective of P_0.

A protocol π_{BSEQ} implementing $\mathcal{F}_{\text{BSEQ}}$ is given in Protocol 3. As motivated in Section 3, the protocol has two main

[3] The set $\{v_1^{(j_b^*)}, \ldots, v_n^{(j_b^*)}\}$ from Step 2d potentially contains fewer than n elements. A computation complexity of $\mathcal{O}(n)$ in Step 2d can be achieved using a hash table that associates the indices I with each node $v^{(j_b^*)}$. Each party carries out Steps 2h and 2i locally, i.e., without communicating with the other party.

[4] It is assumed that Step 2d of $\pi_{\text{O-Prune}}^{j_1^*, j_2^*, j^*}$ is implemented in $\mathcal{O}(n)$ time using, e.g., a hash table.

[5] Note that we assume that a party does not demand any commodity it offers, i.e., $\Gamma_o^{(j)} \cap \Gamma_d^{(j)} = \emptyset$ ($j \in \{0, 1\}$).

$\boldsymbol{P_0}$: $\Gamma^{(0)} = \{(c_{o,1}^{(0)}, c_{d,1}^{(0)}),$ $\ldots, (c_{o,n_0}^{(0)}, c_{d,n_0}^{(0)})\}; n_1$	$\boldsymbol{P_1}$: $\Gamma^{(1)} = \{(c_{o,1}^{(1)}, c_{d,1}^{(1)}),$ $\ldots, (c_{o,n_1}^{(1)}, c_{d,n_1}^{(1)})\}; n_0$
1. Compute $\sigma_0 \in \Sigma_{n_0}$ s.t. $f_0(c_{o,\sigma_0(1)}^{(0)}, c_{d,\sigma_0(1)}^{(0)}) < \cdots$ $< f_0(c_{o,\sigma_0(n_0)}^{(0)}, c_{d,\sigma_0(n_0)}^{(0)})$	Compute $\sigma_1 \in \Sigma_{n_1}$ s.t. $f_1(c_{o,\sigma_1(1)}^{(1)}, c_{d,\sigma_1(1)}^{(1)}) < \cdots$ $< f_1(c_{o,\sigma_1(n_1)}^{(1)}, c_{d,\sigma_1(n_1)}^{(1)})$
2. $\forall i \in \mathbb{N}_{n_0}$, set $\overline{U}_i^{(0)} :=$ $(\mathrm{E}(c_{o,\sigma_0(i)}^{(0)}), \mathrm{E}(c_{d,\sigma_0(i)}^{(0)}),$ $\mathrm{E}(f_0(c_{o,\sigma_0(i)}^{(0)}, c_{d,\sigma_0(i)}^{(0)})), \mathrm{E}(1))$	$\forall i' \in \mathbb{N}_{n_1}$, set $\overline{U}_{i'}^{(1)} :=$ $(\mathrm{E}(c_{d,\sigma_1(i')}^{(1)}), \mathrm{E}(c_{o,\sigma_1(i')}^{(1)}),$ $\mathrm{E}(f_1(c_{o,\sigma_1(i)}^{(1)}, c_{d,\sigma_1(i)}^{(1)})), \mathrm{E}(0))$
3. $\overline{U}^{(0)} := [\overline{U}_1^{(0)} \cdots \overline{U}_{n_0}^{(0)}]$ $\longleftrightarrow$ $\overline{U}^{(1)} := [\overline{U}_1^{(1)} \cdots \overline{U}_{n_1}^{(1)}]$	
4. $(\overline{U}^{(0,1)}) \leftarrow \pi_{\text{O-Merge}}^{3,n_0}([\overline{U}^{(0)}\ \overline{U}^{(1)}])$	
5. $(\overline{V}^{(0,1)}) \leftarrow \pi_{\text{O-Dup}}^{3}(\overline{U}^{(0,1)})$	
6. $([\overline{V}_1^{(0)} \cdots \overline{V}_{n_0}^{(0)}]) \leftarrow \pi_{\text{O-Filter}}^{4}(\overline{V}^{(0,1)})$	
7a. **for** $i \in n_0$ $(b_i^{(0)}, b_i^{(1)}) \leftarrow \pi_{\text{LT}}^{\text{CI-SO}}((\overline{V}_i^{(0)}[\![4]\!], \overline{V}_i^{(0)}[\![5]\!]))$ $((\overline{V}_{i,1}'^{(0)}, \overline{V}_{i,2}'^{(0)})) \leftarrow \pi_{\text{SP}}((\overline{V}_i^{(0)}[\![4]\!], \overline{V}_i^{(0)}[\![5]\!], b_i^{(0)}),$ $(\overline{V}_i^{(0)}[\![4]\!], \overline{V}_i^{(0)}[\![5]\!], b_i^{(1)}))$	
7b. $\overline{V}_i'^{(0)} :=$ $(\overline{V}_i^{(0)}[\![1]\!], \overline{V}_i^{(0)}[\![2]\!], \overline{V}_{i,1}'^{(0)})$	$\overline{V}_i'^{(0)} :=$ $(\overline{V}_i^{(0)}[\![1]\!], \overline{V}_i^{(0)}[\![2]\!], \overline{V}_{i,1}'^{(0)})$
end	
8. $(\overline{W}) \leftarrow \pi_{\text{O-Prune}}^{1,2,3}([\overline{V}_1'^{(0)} \cdots \overline{V}_{n_0}'^{(0)}])$	
9. $([\overline{W}_1' \cdots \overline{W}_w']) \leftarrow \pi_{\text{O-Filter}}^{3}(\overline{W})$	
10. $S := \emptyset$	$S := \emptyset$
11a. **for** $i \in \mathbb{N}_w$ $(c_o^{(0)}) \leftarrow \pi_{(2,2)\text{-Dec}}(\overline{W}_i'[\![1]\!])$ $(c_d^{(0)}) \leftarrow \pi_{(2,2)\text{-Dec}}(\overline{W}_i'[\![2]\!])$	
11b. $S := S \cup \{(c_o^{(0)}, c_d^{(0)})\}$	$S := S \cup \{(c_o^{(0)}, c_d^{(0)})\}$
end	
12. **output** S	**output** S

Protocol 3: π_{BSEQ}.

components: privacy-preserving set intersection for obliviously determining compatible quotes (Steps 1 to 7b) and privacy-preserving bipartite matching to select simultaneously executable quotes (Steps 8 to 12). The set intersection component starts with each party P_j ($j \in \{0,1\}$) locally sorting its quotes with respect to the encoding function f_j. Each party then augments its quotes with a bit indicating the owner and encrypts the augmented quote. Next, the parties exchange their sorted and encrypted quotes and obliviously merge them as columns into a matrix such that the columns of this matrix are ordered according to the encodings. Then, the parties run $\pi_{\text{O-Dup}}$ to augment each column with two encrypted bits to indicate whether there is a duplicate in the left or right neighboring columns, respectively. Next, the parties use $\pi_{\text{O-Filter}}$ to remove all columns originating from P_1 based on the bit indicating the owner while keeping the columns from P_0. At the end of the set intersection component, the parties obliviously combine the two duplicate-indicators of each column to a single indicator corresponding to the compatibility of the respective quote from P_0 with a quote from P_1. Subsequently, the parties run $\pi_{\text{O-Prune}}$ to obliviously select a set of simultaneously executable quotes. Protocol $\pi_{\text{O-Filter}}$ allows the parties to keep the simultaneously executable quotes marked by $\pi_{\text{O-Prune}}$ and eliminate all other quotes. Finally, the parties jointly decrypt the encrypted quotes and output the result.

Theorem 3. *Let P_j ($j \in \{0,1\}$) hold the set of quotes $\Gamma^{(j)}$ with $|\Gamma^{(j)}| = n_j$. Additionally, let P_j hold n_{1-j}. Then, π_{BSEQ} computes functionality $\mathcal{F}_{BSEQ}$ securely in the semi-honest model.*

Theorem 4. *If the subprotocols of π_{BSEQ} are implemented as outlined above and composed as in π_{BSEQ}, then the communication, round, and computation complexities of π_{BSEQ} are in $\mathcal{O}((n_0 + n_1) \cdot \log(n_0 + n_1))$.*

6. RELATED WORK

In the context of privacy-preserving bartering, Förg *et al.* present a protocol for two parties supporting a single quote per party [7]. In contrast, our work supports multiple quotes per party. As is the case with π_{BSEQ}, the protocol from [7] keeps the commodities private at all times. In [7], a commodity is encoded as a $|\mathscr{C}|$-dimensional vector and quote compatibility is checked using a privacy-preserving scalar product protocol. Instead, we encode a commodity as its index in the domain of commodities which allows us to reduce the computation and communication complexity of the quote compatibility check from [7] by factor $|\mathscr{C}|$. While the protocol from [7] cannot determine simultaneously executable quotes, we can combine it with the conditional random selection protocol from [27] to achieve this goal. The idea is to form the Cartesian product of the quotes from P_0 and P_1 and then have the parties obliviously check each pair of quotes for compatibility using the scalar product approach from [7]. To randomly select one compatible pair of quotes, the parties utilize the conditional random protocol from [27]. The parties then decrypt the selected quote. The parties can subsequently remove all quotes which are not simultaneously executable anymore from their set of quotes and repeat the process until no more quotes can be added to the set of simultaneously executable quotes. When the parties repeat the protocol, they learn how many quotes the other parties removed in the prior iteration. To fix this leakage, each party can introduce dummy quotes that are never compatible with any quote from the other party. The resulting communication complexity ranges from $\Omega(n_0 \cdot n_1 \cdot |\mathscr{C}|)$ to $\mathcal{O}(\min(n_0, n_1) \cdot n_0 \cdot n_1 \cdot |\mathscr{C}|)$ depending on the number of simultaneously executable quotes found.[6] An advantage of the approach is that it adds quotes to the (initially empty) current set of simultaneously executable quotes in a greedy fashion, i.e., until no more quotes can be added to the set. Applying the analysis from [18], it follows that π_{BSEQ} still finds at least 63.2% of the simultaneously executable quotes on average that this approach would determine.

Aimeur *et al.* [1] as well as Frikken and Opyrchal [10] contribute further related work. Some differences of [1, 10] to the work by Förg *et al.* [7] as discussed in [7] also apply to our work. In particular, the work from [1] is centered around privacy issues in e-commerce such as price negotiation and tracking of customer product searches. Although the work from [1] also addresses the problem of keeping commodities private, Aimeur *et al.* [1] focus on the e-commerce scenario where a customer buys one product at a time from the merchant. In contrast, a prevalent issue in bartering is to find multiple executable trades at once. As pointed out in [7], the two-party protocols by Frikken and Opyrchal [10]

[6]The factor $|\mathscr{C}|$ can be eliminated by encoding each commodity using its index.

forces the parties to disclose their commodities to the other party. In contrast, our solution keeps commodities private from the beginning. Moreover, the work from [10] requires all parties to specify a valuation for all commodities. In practice, however, several bartering platforms enable their users to express their preferences in terms of quotes which is the setting we focus on.

As pointed out in [7], privacy-preserving matchmaking (e.g., [9, 13, 24, 28]) which addresses the problem of determining whether inputs match in a privacy-preserving fashion is related to privacy-preserving bartering. We answer the question from [7] whether matchmaking protocols can be used in bartering in the affirmative by determining compatible quotes based on the set intersection protocol from [13].

As shown above, determining simultaneously executable quotes can be phrased as a bipartite matching problem. To the best of our knowledge, to date there are only two privacy-preserving bipartite matching protocols [2, 4] which both determine a maximum matching. The protocol by Blanton and Saraph [2] obliviously computes the rank of an adjacency matrix to find a matching. The protocol utilizes garbled circuits and has a communication complexity of $\mathcal{O}(|V|^3 \log |V|)$ where V denotes the nodes in the graph (in the bartering context, V corresponds to the distinct commodities P_0 offers or demands). Chu and Chang [4] utilize homomorphic encryption and garbled circuits to build a matching protocol based on the Hungarian algorithm which operates on a cost matrix of size $|V|^2$ and has a computation complexity of $\mathcal{O}(|V|^4)$ (V denotes the nodes in the graph as before). The computation and communication complexities of our bipartite matching protocol $\pi_{\text{O-Prune}}$ are both in $\mathcal{O}(n_0)$ where n_0 denotes the number of quotes from P_0 (i.e., the graph has at most n_0 edges). As there is at most one edge between every offered and demanded commodity, we have $n_0 \in \mathcal{O}(|V|^2)$ and thus the computation and communication complexities are both bound by $\mathcal{O}(|V|^2)$. Although our matching protocol $\pi_{\text{O-Prune}}$ is more efficient than both maximum matching protocols, it does not always achieve a maximum matching. However, building the matrices for [2, 4] in the bartering context would require us to disclose at least how many distinct commodities some party specifies as part of its quotes or how many of these commodities belong to a compatible quote. As we have to keep this information private, to the best of our knowledge it is not possible to apply [2, 4] to our work.

Note that the work by Kannan *et al.* [15] is orthogonal to our work since the privacy notion in [15] is marginal differential privacy.

7. ACKNOWLEDGMENTS

In part, this work was supported by DFG Award ME 3704/4-1.

8. REFERENCES

[1] E. Aïmeur, G. Brassard, and F. S. Mani Onana. Blind Electronic Commerce. *Journal of Computer Security*, 14(6):535–559, 2006.

[2] M. Blanton and S. Saraph. Oblivious Maximum Bipartite Matching Size Algorithm with Applications to Secure Fingerprint Identification. In *ESORICS*, 2015.

[3] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, Apr. 2000.

[4] W.-T. Chu and F.-C. Chang. A Privacy-Preserving Bipartite Graph Matching Framework for Multimedia Analysis and Retrieval. In *ICMR*, 2015.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[6] Craigslist. https://www.craigslist.org/.

[7] F. Förg, D. Mayer, S. Wetzel, S. Wüller, and U. Meyer. A Secure Two-Party Bartering Protocol Using Privacy-Preserving Interval Operations. In *PST*, 2014.

[8] P. Fouque, G. Poupard, and J. Stern. Sharing Decryption in the Context of Voting or Lotteries. In *FC*, 2001.

[9] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient Private Matching and Set Intersection. In *EUROCRYPT*, 2004.

[10] K. Frikken and L. Opyrchal. PBS: Private Bartering Systems. In *FC*, 2008.

[11] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 1st edition, 2009.

[12] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[13] Y. Huang, D. Evans, and J. Katz. Private Set Intersection: Are Garbled Circuits Better than Custom Protocols? In *NDSS*, 2012.

[14] K. V. Jónsson, G. Kreitz, and M. Uddin. Secure Multi-Party Sorting and Applications. In *ACNS*, 2011.

[15] S. Kannan, J. Morgenstern, R. Rogers, and A. Roth. Private Pareto Optimal Exchange. In *EC*, 2015.

[16] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2007.

[17] F. Kerschbaum and O. Terzidis. Filtering for Private Collaborative Benchmarking. In *ETRICS*, 2006.

[18] A. Mastin and P. Jaillet. Greedy Online Bipartite Matching on Random Graphs. *arXiv.org*, cs.DS, 2013.

[19] D. A. Mayer. *Design and Implementation of Efficient Privacy-Preserving and Unbiased Reconciliation Protocols*. PhD thesis, Stevens Institute of Technology Hoboken, NJ, 2012.

[20] National Association of Trade Exchanges. http://www.natebarter.com/.

[21] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT*, 1999.

[22] Read It Swap It. http://www.readitswapit.co.uk/.

[23] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, Nov. 1979.

[24] J. S. Shin and V. D. Gligor. A New Privacy-Enhanced Matchmaking Protocol. In *NDSS*, 2008.

[25] T. Toft. Sub-Linear, Secure Comparison with Two Non-Colluding Parties. In *PKC*, 2011.

[26] TradeYa! http://www.tradeya.com/.

[27] S. Wüller, U. Meyer, F. Förg, and S. Wetzel. Privacy-Preserving Conditional Random Selection. In *PST*, 2015.

[28] K. Zhang and R. Needham. A Private Matchmaking Protocol, 2001.