Secure Error-Tolerant Graph Matching Protocols

Kalikinkar Mandal\(^1\), Basel Alomair\(^2\), and Radha Poovendran\(^1\)

\(^1\) Network Security Lab, Department of Electrical Engineering, University of Washington, Seattle, WA, 98195, USA.
Email: \{kmandal,rp3\}@uw.edu
\(^2\) National Center for Cybersecurity Technologies, King Abdulaziz City for Science and Technology (KACST), Riyadh, Saudi Arabia. Email: alomair@uw.edu

Abstract. We consider a setting where there are two parties, each party holds a private graph and they wish to jointly compute the structural dissimilarity between two graphs without revealing any information about their private input graph. Graph edit distance (GED) is a widely accepted metric for measuring the dissimilarity of graphs. It measures the minimum cost for transforming one graph into the other graph by applying graph edit operations. In this paper we present a framework for securely computing approximated GED and as an example, present a protocol based on threshold additive homomorphic encryption scheme. We develop several new sub-protocols such as private maximum computation and optimal assignment protocols to construct the main protocol. We show that our protocols are secure against semi-honest adversaries. The asymptotic complexity of the protocol is \(O(n^5\ell \log^*\ell)\) where \(\ell\) is the bit length of ring elements and \(n\) is the number of nodes in the graph.

Keywords: Secure two-party computation, Graph edit distance, Privacy, Graph algorithms

1 Introduction

Graph matching is a task of assessing the structural similarity of graphs. There are two types of graph matching, namely exact matching and error-tolerant matching (also known as inexact matching) [1, 26, 29]. The exact graph matching aims to determine, whether two graphs – a source graph and a target graph – are identical. The later one aims to find a distortion or dissimilarity between two graphs. Graph edit distance is a metric that measures the structural dissimilarity between two graphs. The graph edit distance is quantified as the minimum costs of edit operations required to transform the source graph into the target graph. We consider an attribute graph consisting of a set of nodes, a set of edges and labels assigned to nodes and edges. Examples of such graphs are social network graphs and fingerprint graphs [24, 20]. A standard set of graph edit operations on an attribute graph includes insertion, and deletion and substitution.
of edges and nodes and substitution of vertex and edge labels. Unfortunately, there is no polynomial time algorithm for computing the exact graph edit distance between two graphs. However, several algorithms have been developed for computing approximated or suboptimal graph edit distance in polynomial time [26, 29, 1, 9, 24]. A common strategy used for computing the GED is to find an optimal assignment between each node of one graph to each node of the other graph with minimum cost. The optimal assignment is computed by solving an assignment problem with a cost matrix derived using the structure of the graphs and the costs of graph edit operations. Graph edit distance has many applications in social network graph computation, pattern recognition and biometrics such as in fingerprint identification systems [24, 20].

Our Contributions. In this paper, for the first time, we consider secure two-party graph edit distance computation where each party has a private graph and they wish to jointly compute an approximated graph edit distance between two private graphs, without leaking any information about their input graph. A private graph is meant by the structure of the graph represented by an adjacency matrix, node labels and edge labels are private, only the number of nodes is public. First, we propose a general framework for securely computing approximated graph edit distance, which consists of securely computing the entries of the cost matrix from the private input graphs, securely solving the assignment problem and securely processing an optional phase to obtain the graph edit distance. Then, as an example, we develop a protocol for securely computing an approximated graph edit distance, determining the error-tolerant graph matching, based on the algorithm by Riesen and Bunke [26]. Our protocol construction relies on threshold additive homomorphic encryption scheme [13] instantiated by the threshold Paillier encryption scheme [25]. The reason for choosing homomorphic encryption in the construction is to design efficient protocols by exploiting the structures of the GED algorithms. To construct the main protocol, we develop several sub-protocols such as a private maximum computation protocol and an optimal assignment protocol based on the Hungarian algorithm. We prove the security of the protocol in the semi-honest model. The difference between the workloads of the parties is negligible. The asymptotic complexity for the proposed protocol is $O(n^3(\ell \log^*(\ell)))$, where $\ell$ is the bit length of ring elements and $n$ is the maximum among the numbers of nodes in two graphs.

2 Related Work

Secure Two-party Computation. Secure two-party computation is a powerful tool that enables two parties to jointly compute a function on their private inputs without revealing any information about the inputs except the output of the function. Works on secure two-party computation began with the seminal work of Yao [28] that showed that any function can be securely evaluated in the presence of semi-honest adversaries by first generating a garbled circuit computing that function and then sending it to the other party. Then the other party
can obtain the output by evaluating the garbled circuit using a 1-out-of-2 Oblivious Transfer (OT) protocol. A series of work on secure two-party computation have been done under different security settings and on optimization of garbled circuits [17, 4, 15], to name a few and a number of tools and compilers such as Fairplay [19] and TASTY [14] have been developed for secure computation.

Secure Processing of Graph Algorithms. Graph algorithms have a wide variety of use in many secure applications. Recently secure and data oblivious graph algorithms have been studied in [2, 5, 6]. Aly et al. [2] proposed secure data-oblivious algorithms for shortest path and maximum flow algorithms. In [6], Blanton et al. proposed secure data-oblivious algorithms for breadth-first search, single-source single-destination shortest path, minimum spanning tree, and maximum flow problems. In [5], Blanton and Saraph proposed secure data-oblivious algorithms for finding maximum matching size in a bipartite graph. In our work, as a sub-task, we need to find a perfect matching for computing the optimal cost in a complete weighted bipartite graph.

Secure Edit Distance Computation. An edit distance measures the dissimilarity (similarity) between two strings. In [3], Atallah et al. proposed a privacy-preserving protocol for computing an edit distance between two strings based on an additive homomorphic encryption scheme. Jha et al. [16] presented privacy-preserving protocols for computing edit distance between two strings. The protocols are constructed using oblivious transfer and Yao’s garbled circuits method. Later on, Huang et al. [15] developed a faster protocol for edit distance computation with the garbled circuit approach. Recently, Cheon et al. [7] proposed a privacy-preserving scheme for computing edit distance for encrypted strings. Their protocol is based on a somewhat homomorphic encryption scheme.

3 Preliminaries

In our construction, we use the threshold Paillier encryption scheme (TPS) TPS = (πDistKeyGen, πDistSk, Enc, πDistDec) in the two-party setting, due to Hazay el al. [13] where πDistKeyGen is the protocol for distributively generating a RSA modulus $N = pq$, πDistSk is the protocol for distributed generation of shared private key and πDistDec is the protocol for the distributed Paillier decryption of shared private key. The encryption algorithm Enc is defined as follows. For a plaintext message $m$ with randomness $r \in R \mathbb{Z}_N$ the ciphertext is computed as $c = Enc(m, r) = r^N(N + 1)^m \mod N^2$, where $N = pq$ and $p$ and $q$ are two large primes of equal length. Assume that the bit length of $N$ is $\ell$. The Paillier encryption scheme has 1) additive homomorphic property: $E(m_1 + m_2) = Enc(m_1) \cdot Enc(m_2)$ and $Enc(k m_1) = Enc(m_1)^k$ and 2) rerandomization property meaning for a ciphertext $c$, without knowing the private key, another ciphertext $c' = Rand(pk, Enc(m; r), r') = r'^N r^N(N + 1)^m = (rr')^N(N + 1)^m$ can be created. For the details about other protocols, the reader is referred to [13].

The computation of the GED involves operations on negative numbers as well. We represent the negative numbers in modular arithmetic in the encryption
as $[\lceil \frac{N}{2} \rceil, N - 1] \equiv [-\lfloor \frac{N}{2} \rfloor, -1]$. The positive numbers lie in the range $[0, \lfloor \frac{N}{2} \rfloor]$ and the negative numbers lie in the range $[\lceil \frac{N}{2} \rceil, N - 1]$.

4 Problem Formulation

We consider an undirected attribute graph $G = (V, E, l_G, \zeta_G)$ where $V$ is a finite set of vertices, $E$ is the set of edges, and $l_G$ is the vertex labeling function and $\zeta_G$ is the edge labeling function. Assume that the graph $G$ does not contain any multi-edges and self-loops. Let $G_1 = (V_1, E_1, l_{G_1}, \zeta_{G_1})$ be a source graph and $G_2 = (V_2, E_2, l_{G_2}, \zeta_{G_2})$ be a target graph. The graph edit distance \cite{1, 2} between $G_1$ and $G_2$ is defined by $f_{GED}(G_1, G_2) = \min_{c(eo_1, \ldots, eo_k) \in \Gamma(G_1, G_2)} \sum_{i=1}^{k} c(eo_i)$ where $\Gamma(G_1, G_2)$ is the set of all edit paths that transform $G_1$ into $G_2$ and $c(eo_i)$ denotes the cost for the edit operation $eo_i$. The reader is referred to Appendix \cite{5} for the details about graph edit operations.

In this work we consider a setting where there are two parties $P_1$ and $P_2$, $P_1$ has a private graph $G_1$ and $P_2$ has another private graph $G_2$. The parties wish to compute an approximated graph edit distance $f_{GED}(G_1, G_2)$ between $G_1$ and $G_2$ without leaking anything about their input graph, where $f_{GED}$ is a function running in polynomial time computing an approximated graph edit distance between $G_1$ and $G_2$. At the end of the execution of the protocol, each party $P_i$ should learn nothing about other party’s input graph $G_{3-i}$, beyond the edit distance value $f_{GED}(G_1, G_2)$, $i = 1, 2$. A private graph is meant by node and edge labels and the structure of the graph represented by an adjacency matrix are private, only the number of nodes in the graph is public.

Adversary model. We define the security of the protocol for the GED computation against honest-but-curious or semi-honest adversaries where a party compromised by an adversary follows the prescribed actions of the protocol and aims to learn some unintended information from the execution of the protocol. Let $A$ be a probabilistic polynomial time adversary that can corrupt at most one party at the beginning of the execution of the protocol. The adversary $A$ sends all input messages of the corrupted party during the execution of the protocol and receives messages from the honest party. The honest party follows the instruction of the protocol.

Let $A$ corrupts the party $P_i$. We denote the view of $P_i$ in the real execution of the protocol $\Pi$ by $\text{VIEW}^\Pi_{P_i}(1^\lambda, G_1, G_2) = \{G_i, R_i, m_1, m_2, \ldots, m_T\}$, $i = 1$ or 2, where $G_i$ is $P_i$’s private input graph, $m_1, m_2, \ldots, m_T$ are the messages received from $P_{3-i}$ and $R_i$ is $P_i$’s random tape used during the execution of the protocol.

Definition 1. Let $f_{GED}(G_1, G_2)$ be the functionality computing an approximated graph edit distance. We say that a two-party protocol $\Pi$ securely evaluates $f_{GED}(G_1, G_2)$ in the presence of semi-honest adversaries if there exists a PPT simulator $S = (S_{P_1}, S_{P_2})$ such that for all $G_1$ and $G_2$, it holds that

$$\{S_{P_i}(1^\lambda, G_i, f_{GED}(G_1, G_2))\} \approx \{\text{VIEW}^\Pi_{P_i}(1^\lambda, G_1, G_2)\}$$

where $\approx$ denotes the computational indistinguishability of two distribution ensembles.
5 Description of Proposed GED Protocols

This section presents a framework for the two-party graph edit distance computation based on the assignment problem. As an example, we present a protocol for the graph edit distance computation and prove its security in the semi-honest model.

5.1 A Framework for Two-party GED Computation

Fig. 1 provides the process of an approximated GED computation. At a high level, the graph edit distance computation consists of three phases, namely the construction of the cost matrix, solving the optimal assignment problem with the cost matrix and further processing (optional processing) using the results from the assignment problem and inputs graphs to improve the approximated GED. The cost matrix construction phase takes graph inputs from the parties and computes the entries of the matrix in terms of the costs of graph edit operations. Solving the assignment problem does not take any graph inputs from parties. Based on the approximation factor of the approximated GED, the optional processing is performed. The general structure of the protocols for two-party graph edit distance computation consists of secure two-party evaluations of the cost matrix construction, the optimal assignment problem and optional processing. At the end of secure processing of each phase, we ensure that there is no leakage of information from the output, except the final output that will be known to both parties.

![Fig. 1. A block diagram for two-party graph edit distance computation](image-url)

In the current paper, we perform the secure evaluation of graph edit distance, following the above framework, using the threshold Paillier additive homomorphic encryption scheme. The private key of the encryption scheme is shared between two parties. First, the parties construct an encrypted cost matrix using the input graphs and then they run the optimal assignment protocol on the encrypted cost matrix. The encrypted outputs from the optimal assignment protocol along with the input graphs if needed are used in the optional processing phase to obtain the graph edit distance. In Section 5.3, we present an approximated graph edit distance computation protocol.
5.2 Sub-Protocols

Secure equality testing and comparison protocols have been extensively studied in the literature under different two-party computation settings, e.g., in [8, 11, 27, 18]. We present a variant of encrypted equality test protocol, denoted by $\pi_{EQ}$ in the Appendix. We use the greater-than protocol of Toft [27] with the modification that we replace the equality test protocol by $\pi_{EQ}$. In this section we present two sub-protocols Private Maximum Computation protocol and Optimal Assignment protocol that are necessary for the main protocols for graph edit distance. As our protocol construction uses an equality check, comparison, oblivious transfer and oblivious polynomial evaluation protocol, we denote the functionalities by $F_{EQ}$, $F_{CMP}$, $F_{OT}$ and $F_{OPE}$ and corresponding protocols by $\pi_{EQ}$, $\pi_{CMP}$, $\pi_{OT}$ and $\pi_{OPE}$, respectively.

Private Maximum Computation Protocol Let $P_1$ and $P_2$ hold a vector of encrypted numbers $c = (c_1, c_2, ..., c_n)$ with $c_i = \text{Enc}(x_i)$ for the plaintext vector $x = (x_1, x_2, ..., x_n)$. Let $x_{mi}$ be the maximum value in $x$ for index $mi$, $1 \leq mi \leq n$. The private maximum computation (PMC) protocol is to jointly compute the encrypted maximum value $\text{Enc}(x_{mi})$ and the encrypted index $\text{Enc}(mi)$ from $c$ without revealing $x_{mi}$ and $mi$.

We develop a two-party protocol for private maximum computation. The basic idea behind the construction of the PMC protocol is that one party shuffles the order of the elements of $c$ through a secret permutation $\pi_1$ and after shuffling, each element is re-randomized using $\text{Rand}(\cdot, \cdot, \cdot)$. We denote the resultant vector by $c'$. Next, the other party chooses a random permutation $\pi_2$ and using this permutation, it obliviously picks up an element from $c'$ by running a 1-out-of-$n$ oblivious transfer (OT) protocol [23], denoted by $OT_n^{1}$, and then randomizes the chosen element. Both parties then run a comparison protocol to determine the maximum value. This procedure is repeated $(n - 1)$ times for $\pi_2(i), 2 \leq i \leq n$ to compute the maximum among $n$ encrypted elements. The encrypted index $\text{Enc}(mi)$ for the maximum value is computed through an oblivious polynomial evaluation (OPE) protocol. We use the FNP oblivious polynomial evaluation protocol [10] to obtain the encrypted index $\text{Enc}(mi)$. We describe the details of the protocol in Fig. 2.

Complexity. We evaluate the communication and computation overhead of the $\pi_{PMC}$ protocol, which is composed of $\pi_{CMP}$, an $OT_1^n$ protocol and an OPE protocol. Since the round complexity of $\pi_{CMP}$ is $O(\log(\ell) \log^*(\ell))$, the total communication complexity for $\pi_{CMP}$ is $(n \log(\ell) \log^*(\ell))$. The communication overhead for $OT_1^n$ is $O(n)$. Therefore the overall communication complexity for $\pi_{PMC}$ is $O(n^2 + n\ell \log(\ell) \log^*(\ell))$. It is easy to see that the computation complexity of the protocol is also $O(n^2 + n\ell \log(\ell) \log^*(\ell))$.

Theorem 1. The protocol $\pi_{PMC}$ securely computes the encrypted maximum value and its encrypted maximum index, in the presence of semi-honest adversaries.

Proof. The proof follows from the semantic security of the Paillier encryption scheme. The details of the proof can be found in the full paper [21].
Protocol: Private MAX Computation πPMC

**Input:** A ciphertext vector \( \mathbf{c} = (c_1, c_2, \ldots, c_n) \) of \( \mathbf{x} = (x_1, x_2, \ldots, x_n) \) where \( c_i = \Enc(x_i), 1 \leq i \leq n \).

**Output:** Encryption of the maximum value \( \Enc(x_{\hat{m}}) \) and its encrypted position \( \Enc(m) \).

1. \( P_1 \) chooses a random permutation \( \pi_1 \) on \( \{1, 2, \ldots, n\} \) and computes \((c_{\pi_1(1)}, c_{\pi_1(2)}, \ldots, c_{\pi_1(n)}) \). It then randomizes this vector and obtains \( \mathbf{c}' = (c'_1, c'_2, \ldots, c'_n) \) where \( c'_i = \Rand(pk, c_{\pi_1(i)}, r_i), 1 \leq i \leq n \) where \( r_i \) is a random number.

2. \( P_2 \) chooses a random and secret permutation \( \pi_2 \) on \( \{1, 2, \ldots, n\} \). It then runs an OT\( ^n_1 \) protocol with inputs \( \mathbf{c}' \) from \( P_1 \) and \( \pi_2(1) \) from \( P_2 \). Let \( c'_{\pi_2(1)} \) be the output of the OT protocol. \( P_2 \) randomizes \( c'_{\pi_2(1)} \) as \( c''_{\pi_2(1)} = \Rand(pk, c'_{\pi_2(1)}, r'_1) \) and sends \( c''_1 \) to \( P_1 \).

3. Both parties set \( c_{Index} \leftarrow c''_1 \). \( P_2 \) assigns \( \Index \leftarrow \pi_2(1) \).

4. For each \( t \in [2, n] \), \( P_1 \) and \( P_2 \) perform the following steps:
   (a) \( P_2 \) chooses \( \pi_2(t) \).
   (b) \( P_1 \) and \( P_2 \) run the OT\( ^n_1 \) protocol with inputs \( \mathbf{c}' \) from \( P_1 \) and \( \pi_2(t) \) from \( P_2 \). Let \( c'_{\pi_2(t)} \) be the output of the OT protocol received by \( P_2 \).
   (c) \( P_2 \) randomizes \( c_{\pi_2(t)} \) as \( c''_{\pi_2(t)} = \Rand(pk, c'_{\pi_2(t)}, r'_t) \) and sends \( c''_t \) to \( P_1 \).
   (d) \( P_1 \) and \( P_2 \) run the comparison protocol \( \pi_{\GM} \) with inputs \( c''_t \) and \( c_{Index} \) and let \( \Enc(b_{t-1}) \) be the output. They run the threshold decryption protocol \( \DistDec(\Enc(b_{t-1})) \). If \( b_{t-1} = 1 \), both parties update \( c_{Index} \leftarrow c''_t \) and \( P_2 \) updates \( \Index \leftarrow \pi_2(t) \).

5. \( P_1 \) computes the polynomial representation of \( \pi^{-1}_1 \) using the Lagrange interpolation with coefficients in \( \mathbb{Z}_n \) and let \( Q_{\pi^{-1}_1}(x) = \sum_{j=0}^{n-1} Q_j x^j \) be the polynomial of degree at most \( (n-1) \).

6. \( P_1 \) and \( P_2 \) run the FNP OPE protocol with inputs \( Q_{\pi^{-1}_1}(x) \) from \( P_1 \) and \( \Index \) from \( P_2 \) to compute the encrypted index \( \Enc(m) \) where \( m = Q_{\pi^{-1}_1}(\Index) \).
   (a) \( P_1 \) encrypts the coefficients of \( Q_{\pi^{-1}_1}(x) \) as \( (\Enc(Q_0), \Enc(Q_1), \ldots, \Enc(Q_{n-1})) \) and sends it to \( P_2 \).
   (b) \( P_2 \) computes \( \Enc(Q_{\pi^{-1}_1}(\Index)) = \prod_{j=0}^{n-1} (\Enc(Q_j))^{\Index j} \) and sends \( \Enc(Q_{\pi^{-1}_1}(\Index)) \) to \( P_1 \).

Fig. 2. Protocol for private maximum computation

**Optimal Assignment (OA) Protocol** The assignment problem is one of the fundamental optimization problems. Given two sets \( X = \{u_1, u_2, \ldots, u_n\} \) and \( Y = \{v_1, v_2, \ldots, v_n\} \) and a cost matrix \( W = (w_{ij})_{n \times n} \) where \( w_{ij} \) is the cost of assigning \( u_i \) to \( v_j \), the assignment problem is to find a permutation \( \rho \) on \( [1, n] \) that maximizes \( \sum_{i=1}^{n} w_{\rho(i)i} \). We denote an assignment problem instance and its solution by \( (\rho, \sum_{i=1}^{n} w_{\rho(i)i}) \leftarrow \text{AssignProb}(X, Y, W) \), which can be solved by the Hungarian algorithm with time complexity \( O(n^3) \) [22]. The assignment problem can also be viewed as the problem of finding a perfect bipartite matching in a complete weighted bipartite graph \( G = (V, E, W) \) with \( V = X \cup Y, X \cap Y = \phi \) where the cost matrix \( W \) is the weight matrix consisting of weights of the edges. In this paper, we consider the perfect bipartite matching variant of the Hungarian
algorithm. An optimal assignment $\rho$ that minimizes $\sum_{i=1}^{n} w_{i\rho(i)}$ can be obtained from this by making the entries of the cost matrix $W$ negative.

Given an encrypted cost matrix $W = (\text{Enc}(w_{ij}))_{n \times n}$ for $\text{AssignProb}(X, Y, W)$, we develop a two-party protocol for the assignment protocol based on the Hungarian algorithm for computing $\text{Enc}(\sum_{i=1}^{n} w_{i\rho(i)})$ for an optimal assignment $\rho$. In the secure two-party computation protocol, we resolve the following challenges a) securely computing and updating the labeling of nodes in $X$ and $Y$; b) hiding the edges in the perfect matching set as it eventually determines the optimal assignment $\rho$; and c) securely computing augmenting paths and updating the matching set. Since the order of node and/or edge operations during the execution of the algorithm leaks information about the assignment, we prevent this by encrypting the matching set $M$ and shuffling the order of nodes while keeping the assignment problem invariant. We make the following observation about the assignment problem when it solved using the Hungarian algorithm.

**Observation 1** Let $(\rho, \sum_{i=1}^{n} w_{i\rho(i)}) \leftarrow \text{AssignProb}(X, Y, W)$ be an assignment problem as described above. Let $\pi$ be a permutation on $[1, n]$. Define $X^\pi = \{u_{\pi(1)}, \ldots, u_{\pi(n)}\}$ and $Y^\pi = \{v_{\pi(1)}, \ldots, v_{\pi(n)}\}$ and $W^\pi = (\text{Enc}(w_{\pi(i)\pi(j)}))_{n \times n}$. If the assignment problems $(X, Y, W)$ has an optimal value $\sum_{i=1}^{n} w_{i\rho(i)}$ with assignment mapping $\rho$, then the assignment problem $\text{AssignProb}(X^\pi, Y^\pi, W^\pi)$ has the same optimal value with assignment mapping $\rho_1 = \pi \circ \rho \circ \pi^{-1}$.

Our main idea for constructing the OA protocol is to choose a secret permutation $\pi$ shared between two parties and transform the problem $\text{AssignProb}(X, Y, W)$ into $\text{AssignProb}(X^\pi, Y^\pi, W^\pi)$ and then securely execute the steps of the bipartite matching algorithm on the encrypted cost matrix. The party $P_1$ chooses a secret permutation $\pi_1$ and $P_2$ chooses another secret permutation $\pi_2$. Then they jointly construct the encrypted cost matrix $W^\pi = (\text{Enc}(w_{\pi(i)\pi(j)}))$ where $\pi = \pi_2 \circ \pi_1$. We compute the initial labelings of nodes in $X$ using the private maximum computation protocol $\pi_{\text{PMC}}$. We encrypt node identities $u_i \in X$ and $v_j \in Y$ of the bipartite graph and their labels, denoted by $\text{Enc}(X(u_i))$ and $\text{Enc}(Y(v_j))$ for $v_j \in Y$ and construct 2-tuple sequences as $(\text{Enc}(u_i), \text{Enc}(Y(v)))$, $1 \leq i \leq n$ for both $X$ and $Y$. We use the same permutation $\pi$ to hide the order of each sequence of 2-tuple encrypted values component-wise for both $X$ and $Y$. Denoting $M = \{(\text{Enc}(u), \text{Enc}(v)) : u \in X, v \in Y\}$ by the matching set containing encrypted edges, $\{\text{Enc}(u) : u \in X\}$ the set all encrypted head node ids in $M$ by $M \ast X$ and $\{\text{Enc}(v) : v \in Y\}$ the set all encrypted head node ids in $M$ by $M \ast Y$. The initial matching is found by using the $\pi_{\text{EQ}}$ and $\pi_{\text{DistDec}}$ protocol. An encrypted equality graph $\text{EQ}^{\pi_{\text{Bo}}}$ represented by an encrypted adjacency matrix is constructed from encrypted labels for $X$ and $Y$ and encrypted cost matrix $W$ using the $\pi_{\text{EQ}}$ protocol. The perfect matching is found by extending the matching set by finding an encrypted augmenting path. An encrypted augmenting path is found by executing the breadth-first-search (BFS) algorithm on the encrypted equality graph $\text{EQ}^{\pi_{\text{Bo}}}$ where the source and target vertices are free vertices in $X$ and $Y$. We adopt a variant of Blanton et al.’s BFS algorithm [6] in our setting where the secret key for the decryption algorithm is shared between two parties and we denote this protocol by $\pi_{\text{BFS}}$. We don’t provide the technical details
due to space limit. For an encrypted equality graph \( P := \text{Enc}(t_0) - \text{Enc}(t_1) - \text{Enc}(t_2) - \cdots - \text{Enc}(t_k) \) of length \( k-1 \), the set of encrypted edges are given by \( P_{\text{edge}} = \{(\text{Enc}(t_0), \text{Enc}(t_1)), (\text{Enc}(t_2), \text{Enc}(t_3)), \cdots, (\text{Enc}(t_{k-1}), \text{Enc}(t_k))\} \). After finding \( P_{\text{edge}} \), the matching set is updated as \( M \leftarrow M \Delta P_{\text{edge}} \) where \( \Delta \) is the symmetric difference set operation. We use two dummy counters of length \( n \) for keeping track of encrypted free nodes of \( X \) and \( Y \). For computing the GED, we only need the maximum value \( \sum_{i=1}^{n} w_{ij} \)(i). Thus the protocol outputs only \( \sum_{i=1}^{n} w_{ij}(i) \). Fig. 8 presents the details of our secure protocol for the assignment problem.

**Complexity.** From \( \pi_{OA} \), it can be seen that the time complexity for finding the initial matching (Step 3 to Step 7) is \( O(n^3 + n^2 \ell \log(\ell) \log^*(\ell)) \). If the initial matching is not a perfect matching, the computational complexity for terminating the protocol is \( O(n^5 \ell \log^*(\ell) + n^4 \ell \log(\ell) \log^*(\ell)) = O(n^5 \ell \log^*(\ell)) \). An insecure version of the Hungarian algorithm runs in \( O(n^3) \) steps. The overhead of the protocol due to security is \( (n^2 \ell \log^*(\ell)) \).

**Theorem 2.** The protocol \( \pi_{OA} \) securely computes the encrypted optimal value in the presence of semi-honest adversaries.

**Proof.** We prove the security of the protocol in the hybrid model. In the protocol, one party receives messages from the other party and also from the trusted third party computing a functionality. The simulator also needs to simulate the outputs for the trusted third party functionalities. We construct two different simulators for the view of the adversary.

**When \( P_1 \) is corrupted.** Let \( A \) be the adversary controlling the party \( P_1 \), \( S_1 \) chooses \( 2n \) uniformly random tapes \( r_0 = \{(r_{ij}^0, r_{ij}^1)\}_{ij=1}^{n} \) from \( \mathbb{Z}_N \) and computes the encrypted 2-tuple vector \( LBL^Y \). It emulates the outputs \( \text{Enc}(b_1|X(u)) \) and \( \text{Enc}(d_i), 1 \leq i \leq n \) for the trusted third party functionality \( F_{PMC} \) on \( i \)th row \( W_i \). \( S_1 \) chooses \( n \) random tapes \( r_1 = \{r_{ij}^1\}_{ij=1}^{n} \) uniformly at random for \( P_1 \) and computes the encryptions of \( i, 1 \leq i \leq n \). \( S_1 \) picks a permutation \( \pi_1 \) and \( (6n + n^2) \) random tapes \( r_2 = \{(r_{ij}^2, r_{ij}^3, r_{ij}^4, r_{ij}^5, r_{ij}^6, r_{ij}^7, r_{ij}^8)\}_{i=1}^{n} \) uniformly at random for \( P_1 \) and computes \( LBL^X, 1, LBL^Y_2, 1, D^\pi_1, W^\pi_1 \) and rerandomizes each encrypted value. \( S_1 \) chooses \( \pi_2 \) and \( (6n + n^2) \) random tapes \( r_3 = \{(r_{ij}^2, r_{ij}^3, r_{ij}^4, r_{ij}^5, r_{ij}^6, r_{ij}^7, r_{ij}^8)\}_{i=1}^{n} \) uniformly at random and computes \( LBL^X, r_{ij}^3, LBL^Y_2, r_{ij}^3, D^\pi_2, W^\pi_2 \) and rerandomizes each encrypted value using \( r_3 \). In Step 5, for each \( \text{Enc}(d_{ij}(i)), 1 \leq i \leq n \), the simulator generates \( b_{ij} \) at random and computes \( \text{Enc}(b_{ij}) \) for \( F_{EQ} \) with inputs \( \text{Enc}(d_{ij}(i)) \) and \( \text{Enc}(d_{ij}(i)), 1 \leq j \leq i - 1 \) and obtains \( b_1 = (b_1, b_2, \cdots, b_n) \). \( S_1 \) computes \( R \) from \( b_1 \). \( S_1 \) computes \( M \). In Steps 10 - 14, \( S_1 \) simulates the output of the functionalities \( F_{EQ}, F_{DistDec}, F_{PMC} \) and \( F_{BFS} \) while ensuring the loop terminates in \( O(n^3) \) steps. The outputs at \( \ell \)-th iteration for Steps 10 - 14 are \( b_2 = (b_1, b_2, \cdots, b_n) \) (Step 10); \( z_\ell = (z_1, z_2, \cdots, z_n) \) and \( b_1^\ell = (b_1, b_2, \cdots, b_n) \) (Step 11); \( b_5^\ell = (b_1, b_2, \cdots, b_{|\mathcal{N}(u)|}) \) (Step 12); \( b_6^\ell = (b_1, b_2, \cdots, b_\ell, t, \leq n, (\text{Step 13}); \) \( b_7^\ell = (b_1, b_2, \cdots, b_\ell, t, \leq n, P_{\text{sim}} = \text{Enc}(u) - \text{Enc}(v) \) (simulated augmented path), \( b_8^\ell = (b_1, b_2, \cdots, b_{|\mathcal{P}_{\text{edge}}|}) \) and \( b_9^\ell = (b_1, b_2, \cdots, b_\ell, t, \leq n \) (Step 14). Define \( B^\ell = (b_3^\ell, z_\ell, b_5^\ell, b_6^\ell, P_{\text{sim}}, b_8^\ell, b_9^\ell) \). The output of \( S_1 \) is \( S_1(1^\lambda, n, X, Y, W) = \).
Protocol: Optimal Assignment based on the Hungarian algorithm \( \pi_{\text{OA}} \)

**Input:** The cost matrix \( \text{Enc}(W) = (\text{Enc}(w_{ij}))_{n \times n} \) \( w_{ij} = \text{cost}(u_i, v_j) \).

**Output:** Optimal assignment value \( \text{Enc}(\sum_{i=1}^{n} w_{i\rho(i)}) \).

1. \( P_1 \) computes \( \text{Enc}(\text{lbl}_Y(v_i)) : 1 \leq i \leq n \) with \( \text{lbl}_Y(v_i) = 0, \forall \nu \in Y \) and \( \text{Enc}(i) : 1 \leq i \leq n \) and constructs \( LBL^Y = (\text{Enc}(\text{lbl}_Y(v_i)), \text{Enc}(i)) : 1 \leq i \leq n \) and sends it to \( P_2 \).
2. \( L \leftarrow \phi; V_P \leftarrow \phi; M \leftarrow \phi \)
3. For each \( u_i \in X, P_1 \) and \( P_2 \) run \( \pi_{\text{PMC}} \) with input ith row \( W_i = (w_{i1}, w_{i2}, \ldots, w_{in}) \) and obtain output \( \text{Enc}(\text{lbl}_X(u_i)) \) and \( \text{Enc}(d_i) \) where \( \text{lbl}_X(u_i) = w_{id_i} \) \( \max \{w_{ij} | u_i \in X \land 1 \leq d_i \leq n \} \).
   (a) Construct \( LBL^X = (\text{Enc}(\text{lbl}_X(u_i)), \text{Enc}(i)) : 1 \leq i \leq n \).
   (b) Construct \( D = (\text{Enc}(i), \text{Enc}(d_i)) : 1 \leq i \leq n \).
   (c) Update \( L \leftarrow L \cup \{ (\text{Enc}(\text{lbl}_X(u_i)), \text{Enc}(\text{lbl}_Y(v_i))) \} \).
4. \( P_1 \) chooses a random perm \( \pi_1 \) and computes the following and sends all to \( P_2 \)
   (a) \( LBL^{\pi_1} = (\text{Enc}(\text{lbl}_Y(v_{\pi_1}(j))), \text{Enc}(\pi_1(j))) : 1 \leq j \leq n \), \( LBL^{\pi_1} = (\text{Enc}(\text{lbl}_X(u_{\pi_1}(i))), \text{Enc}(\pi_1(i))) : 1 \leq i \leq n \).
   (b) \( D^{\pi_1} = (\text{Enc}(\pi_1(j)), \text{Enc}(d_{\pi_1}(j))) : 1 \leq j \leq n \).
   (c) \( W^{\pi_1} = (\text{Enc}(w_{\pi_1(i)\pi_1(j)}))_{n \times n} \).
   (d) Randomize each encrypted value above.
5. \( P_2 \) chooses a random perm \( \pi_2 \) and computes the following and sends all to \( P_1 \)
   (a) \( LBL^{\pi_2\pi_1} = (\text{Enc}(\text{lbl}_Y(v_{\pi_2\pi_1}(j))), \text{Enc}(\pi_2 \circ \pi_1(j))) : 1 \leq j \leq n \).
   (b) \( LBL^{\pi_2\pi_1} = (\text{Enc}(\text{lbl}_X(u_{\pi_2\pi_1}(i))), \text{Enc}(\pi_2 \circ \pi_1(i))) : 1 \leq i \leq n \).
   (c) \( D^{\pi_2\pi_1} = (\text{Enc}(\pi_2 \circ \pi_1(j)), \text{Enc}(d_{\pi_2\pi_1}(j))) : 1 \leq j \leq n \).
   (d) \( W^{\pi_2\pi_1} = (\text{Enc}(w_{\pi_2\pi_1(i)\pi_2\pi_1(j)}))_{n \times n} \).
   (e) Randomize each encrypted value above. Set \( \pi = \pi_2 \circ \pi_1 \).
6. For each \( (\text{Enc}(\pi(i)), \text{Enc}(d_{\pi(i)})) \in D^n, i = 1, \ldots, n, P_1 \) and \( P_2 \) run \( \pi_{\text{EQ}} \) protocol with inputs \( \text{Enc}(d_{\pi(i)}) \) and \( \text{Enc}(d_{\pi(j)}) \) and obtain \( \text{Enc}(b_{ij}) \). \( b_{ij} \in \{0, 1\} \) for \( j = 1, \ldots, i-1 \). Compute \( R = \prod_{j=1}^{i-1} \text{Enc}(b_{ij}). P_1 \) and \( P_2 \) run \( \pi_{\text{EQ}} \) protocol with \( \text{inputs} \text{Enc}(\text{lbl}_X(u_{\pi(i)})) \) and \( \text{Enc}(\text{lbl}_Y(v_{\pi(j)})) \). After \( P_1 \) and \( P_2 \) then jointly decrypt \( \text{Enc}(b_i) \). If \( b_i = 1 \), perform \( M \leftarrow M \cup \{ (\text{Enc}(\pi(i)), \text{Enc}(d_{\pi(i)})) \} \).
7. If \( |M| = n \), return the encrypted optimal value as \( \text{Enc}(\sum_{i=1}^{N} \text{lbl}_X(u_{\pi(i)}) + \sum_{i=1}^{n} \text{lbl}_Y(v_{\pi(i)})) = \prod_{i=1}^{n} \text{Enc}(\text{lbl}_X(u_{\pi(i)})) \prod_{i=1}^{n} \text{Enc}(\text{lbl}_Y(v_{\pi(i)})) \).
8. \( P_1 \) and \( P_2 \) construct a matrix \( EQ^{\pi_1} = (\text{Enc}(e_{ij}))_{n \times n} \) by running the \( \pi_{\text{EQ}} \) protocol with inputs \( \text{Enc}(\text{lbl}_X(u_{\pi(i)} + \text{lbl}_Y(v_{\pi(j)}))) \) and \( \text{Enc}(w_{\pi(i)\pi(j)}) \) where \( \text{Enc}(e_{ij}) \) is the output \( 1 \leq i, j \leq n \) and \( e_{ij} \in \{0, 1\} \).
9. Initialize \( S \leftarrow \emptyset \) and \( T \leftarrow \phi \).
10. \( P_1 \) and \( P_2 \) find \( \text{Enc}(u_{\pi(i)}) \) such that \( \text{Enc}(u_{\pi(i)}) \notin M \times X \), then \( S \leftarrow S \cup \{ \text{Enc}(u_{\pi(i)}) \} \).
11. \( P_1 \) and \( P_2 \) compute \( N_{\text{Enc}}(S) \) for each \( \text{Enc}(u_{\pi(i)}) \in S \)
   (a) For row \( EQ^b = (\text{Enc}(e_{i1}), \text{Enc}(e_{i2}), \ldots, \text{Enc}(e_{in})) \) of \( EQ^{\pi_1} \), compute \( Z_i = (\text{Enc}(e_{i1} \cdot v_1), \text{Enc}(e_{i2} \cdot v_2), \ldots, \text{Enc}(e_{in} \cdot v_n)) \) from \( EQ^b \) where \( \text{Enc}(e_{ik} \cdot k) = \text{Enc}(e_{ik}) \).
   (b) Run \( \pi_{\text{EQ}} \) protocol with input \( Z_i = (\text{Enc}(e_{i1} \cdot v_1), \text{Enc}(e_{i2} \cdot v_2), \ldots, \text{Enc}(e_{in} \cdot v_n)) \) and obtain the output \( Z_i' = (z_1, z_2, \ldots, z_n) \).
   (c) Run \( \pi_{\text{EQ}} \) with inputs \( z_j \) and \( \text{Enc}(0) \) and obtain the output \( \text{Enc}(b_j) \). Run \( \text{DistDec} \) on input \( \text{Enc}(b_j) \) and obtain \( b_j \) for \( 1 \leq j \leq n \). If \( b_j = 0 \), perform \( N_{\text{Enc}}(S) \leftarrow N_{\text{Enc}}(S) \cup \{ \text{Enc}(v_j) \} \).
Protocol: Optimal Assignment $\pi_{OA}$ (Cont.)

12. $P_1$ and $P_2$ check the equality of sets $N_{lb1}(S)$ and $T$ running $\pi_{EQ}$ and $\pi_{DistDec}$ protocols.

13. If $N_{lb1}(S) = T$
   (a) $P_1$ and $P_2$ compute $T = ((LBL_{1}^{Y} \ast Y) - T)$ from sets $LBL_{1}^{Y} \ast Y$ and $T$ by running $\pi_{EQ}$ and $\pi_{DistDec}$ protocols.
   (b) For each $Enc(u_{\pi(i)}) \in S$ and $Enc(v_{j}) \in T$, $P_1$ and $P_2$ compute $Enc(bl_{i}) = Enc(bl_{i}X(u_{\pi(i)}) + bl_{1}Y(v_{j}) - w_{\pi(i)\pi(j)})$.
   (c) $P_1$ and $P_2$ compute $Enc(\delta_{lb1}) = \min\{Enc(bl_{i}) : Enc(i) \in S, Enc(j) \in T\}$ using the $\pi_{PMC}$ protocol.
   (d) $P_1$ and $P_2$ update the label $lb_{i}$ as
   \[
   Enc(bl_{1}X(u)) = Enc(bl_{1}X(u)) \cdot Enc(\delta_{lb1})^{-1} \quad \text{if } E(u) \in S \\
   Enc(bl_{1}Y(v)) = Enc(bl_{1}Y(v)) \cdot Enc(\delta_{lb1}) \quad \text{if } E(v) \in T
   \]

14. If $N_{lb1}(S) \neq T$
   (a) $P_1$ and $P_2$ choose $Enc(v_{j}) \in N_{lb1}(S) - T$.
   (b) If $Enc(v_{j}) \notin M \ast Y$, find an augmenting path $P := Enc(u_{k}) - Enc(v_{j})$ by running the $\pi_{BFS}$ protocol with inputs $EQ^{lb1}$ and $Enc(v_{j})$.
   (c) Update $M \leftarrow M \Delta P_{\text{edge}}$. Goto Step 7.
   (d) If $Enc(v_{j}) \in M \ast Y$ and $(Enc(u_{\pi(t)}), Enc(v_{j})) \in M$, extend alternating tree $S \leftarrow S \cup \{Enc(u_{\pi(t)})\}$ and $T \leftarrow T \cup \{Enc(v_{j})\}$. Goto Step 11.

Fig. 3. Secure Optimal Assignment Protocol based on the Hungarian Algorithm

$(r_{1}, \pi_{1}, r_{2}, LBL^{X \ast \pi_{2} \ast \pi_{1}}, LBL^{Y \ast \pi_{2} \ast \pi_{1}}, D^{\pi_{2} \ast \pi_{1}}, W^{\pi_{2} \ast \pi_{1}}, EQ^{lb1}, b_{1}, b_{2}, \{B^{\delta}\})$. The distributions for $LBL^{X \ast \pi_{1}}, LBL^{Y \ast \pi_{1}}, D^{\pi_{1}}, W^{\pi_{1}}, EQ^{lb1}, b_{1}, b_{2}, \{B^{\delta}\}$ in the real and ideal executions are identically distributed since the random tapes for $r_{1}$ and $r_{2}$ were chosen uniformly at random, the Pailler encryption scheme is semantically secure and the permutation $\pi_{2}$ for the honest party in the real execution of the protocol is unknown to $S_{1}$.

When $P_{2}$ is corrupted. Let the adversary $A$ controlling the party $P_{2}$. The construction of the simulator is similar to that of $S_{1}$, except Step 1. We don’t provide the details of the simulator $S_{2}$. The view of $A$ output by $S_{2}$ is $S_{2}(1^{k}, n, X, Y, W) = (r_{3}, \pi_{2}, LBL^{X \ast \pi_{1}}, LBL^{Y \ast \pi_{1}}, D^{\pi_{1}}, W^{\pi_{1}}, EQ^{lb1}, b_{1}, b_{2}, \{B^{\delta}\})$. Applying the similar argument, the views for the adversary in the real and ideal execution of the protocol are identically distributed.

As the protocols $\pi_{EQ}$, $\pi_{PMC}$, $\pi_{SR}$, $\pi_{DistDec}$ and $\pi_{BFS}$ are secure, applying the composition theorem, $\pi_{OA}$ is secure in the hybrid model against semi-honest adversaries and hence $\pi_{OA}$ is secure in the real execution of the protocol.

5.3 The Main Protocol for Graph Edit Distance

In this section we present a secure realization of the approximated GED computation by Riesen and Bunke [26], based on bipartite graph. We consider the two-party computation in the semi-honest model. We consider a setting where there are two parties $P_{1}$ and $P_{2}$, each party has a private graph $G_{i} = (V_{i}, E_{i}, l_{G_{i}}, \zeta_{G_{i}})$
with \( n_i = |V_i| \geq 3 \) and \( n = n_1 + n_2 \). For simplicity, we consider the cost matrix \( W \) that includes only the costs of node edit operations. The parties start the protocol execution by computing a cost matrix. We start by explaining how the parties jointly construct the cost matrix.

**Encrypted Cost Matrix Construction** We assume that each party secretly defines the costs for the graph edit operations deletion, insertion and substitution of nodes and/or edges. The edit operation costs for nodes are defined as follows. Let \( l_g(u_i) = \alpha_i \in \mathbb{Z}_N \) be the node labeling function of \( G_1 \) and \( l_g(v_j) = \beta_j \in \mathbb{Z}_N \) be the node labeling function of \( G_2 \). The party \( P_1 \) chooses the edit costs of node insertion and deletion operations as \( c(u_i \rightarrow \epsilon) = c(\epsilon \rightarrow u_i) = C_1 \in \mathbb{Z}_N \). Similarly, the party \( P_2 \) decides the costs of insertion and deletion operations for nodes as \( c(\epsilon \rightarrow v_j) = c(v_j \rightarrow \epsilon) = C_2 \in \mathbb{Z}_N \). The cost of the node substitution operation is defined as \( w_{ij} = c(u_i \rightarrow v_j) = \min\{(c(u_i \rightarrow \epsilon) + c(\epsilon \rightarrow v_j)), c'(u_i \rightarrow v_j)\} = \min\{(C_1 + C_2), |\alpha_i - \beta_j|\} \) where \( c'(u_i \rightarrow v_j) = |\alpha_i - \beta_j|, \alpha_i, \beta_j \in \mathbb{Z}_N \). This definition of the cost function can be found in [24]. Each entry of the cost matrix is computed by running a joint protocol.

We now explain how to construct the encrypted cost matrix \( W = (\text{Enc}(w_{ij}))_{n \times n} \). For insertion and deletion operations, the party \( P_2 \) encrypts its cost \( \text{Enc}(C_i) \) and sends it to the other party. For the substitution cost, the parties exchange respective encrypted costs of insertion and deletion operations and encrypted node labels. Let the parties \( P_1 \) and \( P_2 \) have encryptions \( \text{Enc}(d_1) \) and \( \text{Enc}(d_2) \) of numbers \( d_1 \) and \( d_2 \), respectively and they would like to compute \( \text{Enc}(|d_1 - d_2|) \) where \( |d_1 - d_2| \) is the absolute difference between \( d_1 \) and \( d_2 \). The absolute difference between \( d_1 \) and \( d_2 \) can be computed as \( |d_1 - d_2| = (d_1 - d_2) + b(d_2 - d_1) = (1 - b)d_1 + (b - 1)d_2 \) where \( b = 0, d_1 < d_2 \); otherwise, \( b = 1 \). We use this relation to compute encrypted absolute difference between two encrypted numbers. We provide the details of the protocol in Figure 6 in Appendix A.

**Description of the Protocol** We are now ready to describe the protocol. The parties \( P_1 \) and \( P_2 \) initiate the protocol by generating the public key and the shares of the private key for the threshold Paillier encryption scheme using \( \pi_{\text{DistKeyGen}} \) and \( \pi_{\text{Dist}} \), respectively. Each party encrypts its node labels for the construction of the encrypted cost matrix. The computation of GED consists of two main phases. First, the parties construct the encrypted cost matrix \( \text{Enc}(W) = (\text{Enc}(w_{ij}))_{n \times n} \) using the function defined above and then solve the assignment problem with input as the encrypted cost matrix \( \text{Enc}(W) \) to find an optimal of the nodes of the graphs. The parties use the distributed decryption protocol \( \text{DistDec} \) to obtain the graph edit distance \( f_{\text{GED}}(G_1, G_2) = (\sum_{i=1}^{n} w_{id_i}) \). Fig. 4 presents the details of the approximated GED computation protocol.

**Complexity of PiGED.** For computing encrypted node labels, each party performs \( O(n) \) operations. The computation complexity for constructing the encrypted cost matrix is \( O(n^2) \). The parties run the optimal assignment protocol

---

3 Several constructions of cost matrix can be found in [20, 9] for the improvement of the approximation of the actual GED. However, the two-party computation of GED remains same, except the cost matrix construction.
Protocol: Protocol $\Pi_{GED}$

**Input:** Graph $G_1 = (V_1, E_1, l_{G_1})$ from $P_1$ and $G_2 = (V_2, E_2, l_{G_2})$ from $P_2$.

**Output:** Approximated graph edit distance $d = f_{GED}(G_1, G_2)$ or $\perp$.

1. $P_1$ and $P_2$ run the distributed key generation protocol $\pi_{Dist\text{KeyGen}}$, followed by the distributed shared secret key generation protocol $\pi_{Dist\text{Sk}}$.
2. $P_1$ encrypts the cost for node insertion and deletion $\text{Enc}(C_1)$ and $\{\text{Enc}(\alpha_i) : u_i \in V_1\}$.
3. $P_2$ encrypts the cost for node insertion and deletion $\text{Enc}(C_2)$ and $\{\text{Enc}(\beta_j) : v_j \in V_2\}$.
4. For each $u_i \in V_1$ and $v_j \in V_2$, $P_1$ and $P_2$ run $\pi_{Sub}$ with inputs $\text{Enc}(C_1)$ and $\text{Enc}(\alpha_i)$ from $P_1$ and inputs $\text{Enc}(C_2)$ and $\text{Enc}(\beta_j)$ from $P_2$ and obtain the output $\text{Enc}(w_{ij})$.
5. $P_1$ and $P_2$ make all the entries of $E(W)$ negative, i.e., $W' = (\text{Enc}(w'_{ij}))_{n \times n} = (\text{Enc}(w_{ij})^{-1})_{n \times n}$ where $w'_{ij} = -w_{ij}$.
6. Run the optimal assignment protocol $\pi_{OA}$ with input $\text{Enc}(W')$ and obtain the output the encrypted the minimum cost $\text{Enc}(\sum_{i=1}^{n} w'_{id_i})$.
7. $P_1$ and $P_2$ jointly run the distributed decryption protocol $\pi_{Dist\text{Dec}}$ on $\text{Enc}(\sum_{i=1}^{n} w'_{id_i})^{-1}$ and obtain the approximated graph edit distance $d = (\sum_{i=1}^{n} w_{id_i})$.

**Fig. 4.** Protocol for computing an approximated graph edit distance based on bipartite graph

on the encrypted matrix. The computational complexity and the communication complexity of the graph edit distance protocol $\Pi_{GED}$ is atmost $O(n^5\ell \log^*\ell)$. The complexity of the protocol is dominated by that of the optimal assignment protocol. In the protocol execution, the parties do almost an equal amount of computation.

**Theorem 3.** Assuming the threshold Paillier encryption scheme is secure, the protocol $\Pi_{GED}$ is secure in the presence of the semi-honest adversaries.

**Proof.** The protocol $\Pi_{GED}$ sequentially invokes the protocols for distributed key generation $\pi_{Dist\text{KeyGen}}$ and $\pi_{Dist\text{Sk}}$, the cost matrix construction $\pi_{Sub}$ and the optimal assignment $\pi_{OA}$, and the distributed decryption $\pi_{Dist\text{Dec}}$ for the approximated GDE. The protocols $\pi_{Dist\text{KeyGen}}$, $\pi_{Dist\text{Sk}}$ and $\pi_{Dist\text{Dec}}$ are secure according to [13]. The construction of $\pi_{Sub}$ based on $\pi_{CMP}$. The security of the $\pi_{Sub}$ protocol relies on that of $\pi_{CMP}$, which is proven secure in [27]. Theorem 2 guarantees the parties securely solves the assignment problem. According to the sequential composition theorem [12], $\Pi_{GED}$ is secure against semi-honest adversaries in the real execution of the protocol.  

\[ \square \]

**6 Conclusions**

In this paper we considered secure two-party computation of graph edit distance measuring the dissimilarity between two graphs where each party has a
private graph and they wish to jointly compute graph edit distance of two private graphs. We proposed a framework for the graph edit distance computation and, as an example, developed a protocol for computing of graph edit distance. To construct main protocols for graph edit distance, we developed sub-protocols such as private maximum computation and optimal assignment protocol based on the Hungarian algorithm. The asymptotic complexities of both protocols are $O(n^2(\ell \log^* \ell))$. Our protocol is secure against semi-honest adversaries and has applications in two-party social network graph computations for measuring structural similarity and fingerprint identifications.

Acknowledgement. This work was supported by ONR grant N00014-14-1-0029 and a grant from the King Abdulaziz City for Science and Technology (KACST). The authors would like to thank Seny Kamara for conducting several discussions during the initial phase of this work. The authors also thank the anonymous reviewers of CANS 2016 for bringing the references [3, 16] into our attention and for their helpful comments.

References


A Description of Sub-Protocols

A.1 Encrypted Equality Test Protocol and Comparison Protocol

Given encryptions Enc(y₁) and Enc(y₂) of y₁ and y₂, respectively, where y₁ and y₂ are of ℓ-bit numbers in \( \mathbb{Z}_N \). In our setting, the secure equality testing protocol outputs the encrypted value Enc(b) where \( b = 0 \) if \( y₁ \neq y₂ \) and \( b = 1 \) if \( y₁ = y₂ \), without revealing \( y₁, y₂ \) and \( b \) where \( y₁ \) and \( y₂ \) are of \( ℓ \) bits. Our equality testing protocol is based on the idea of plaintext-space reduction introduced in [11], which can also be found in [18]. The setting of the equality check is different from the one proposed in [11]. In our case, the private key is shared between the parties, but in [11], one party holds the private key and the other party holds the encrypted numbers. We describe a secure encrypted equality test protocol based on plaintext-space reduction in Fig. 5. We use the greater-than protocol of Toft [27] with the modification that we replace the equality test protocol by \( \pi_{\text{EQ}} \). We denote this protocol by \( \pi_{\text{CMP}} \) which takes inputs Enc(x) and Enc(y) and outputs Enc(b) where \( b = 1 \) iff \( x \geq y \) and \( b = 0 \), otherwise. Its round complexity is \( O(\log(\ell)) \) and computation complexity is \( O(\ell \log(\ell) \log^*(\ell)) \).
Protocol: Equality Test $\pi_{EQ}$

Input: Two encrypted numbers $\text{Enc}(y_1)$ and $\text{Enc}(y_2)$.

Output: $\text{Enc}(b)$ where $b = 0$ if $y_1 \neq y_2$ and $b = 1$ if $y_1 = y_2$.

1. Denote $y = y_1 - y_2$. $P_1$ and $P_2$ perform the operation: $\text{Enc}(y) = \frac{\text{Enc}(y_1)}{\text{Enc}(y_2)}$.

2. $P_1$ generates a random number $A^1$ and represents it in binary as $A^1 = A^1_{\ell-1}A^1_{\ell-2}...A^1_0$ and computes $c_1 \leftarrow \text{Enc}(y + A^1_{\ell-1})$ and $C^1_i \leftarrow \text{Enc}(A^1_i), 0 \leq i \leq \ell - 1$ and send $\{c_1, C^1_i, 0 \leq i \leq \ell - 1\}$ to $P_2$.

3. $P_2$ generates a random number $A^2$ and represents it in binary as $A^2 = A^2_{\ell-1}A^2_{\ell-2}...A^2_0$ and computes $c_2 \leftarrow \text{Enc}(y + A^1_{\ell-1} + A^2_{\ell-1})$ and computes $C^2_i \leftarrow \text{Enc}(A^2_i), 0 \leq i \leq \ell - 1$ and send $\{c_2, C^2_i, 0 \leq i \leq \ell - 1\}$ to $P_1$.

4. $P_1$ and $P_2$ run $\pi_{\text{DetEnc}}$ to decrypt $c_2$ to obtain $x$ where $x = y + A^1 + A^2 = y + A$ and $A = A^1 + A^2$.

5. $P_1$ and $P_2$ compute ciphertexts $\text{Enc}(A_i)$ with $A = A_{\ell-1}...A_0$ using $C^1_i, C^2_i, 0 \leq i \leq \ell - 1$ and additive circuits of two integers as follows. $P_1$ and $P_2$ compute $\text{Enc}(s_0)$ with $s_0 = 0$. For $i = 0$ to $\ell - 1$, $P_1$ and $P_2$ execute the following steps:

   (a) $P_1$ computes $\text{Enc}(A_i s_i) = \text{Enc}(s_i) A^1_{\ell-1}$ and $\text{Enc}(2 A_i s_i) = \text{Enc}(s_i) 2 A^1_{\ell-1}$ from $\text{Enc}(s_i)$ and send these two to $P_2$.

   (b) $P_2$ computes $\text{Enc}(A_i^2 s_i) = \text{Enc}(s_i) A^2_{\ell-1}$ and $\text{Enc}(2 A_i^2 s_i) = \text{Enc}(s_i) 2 A^2_{\ell-1}$ from $\text{Enc}(s_i)$ and send these two to $P_1$.

   (c) Using $\text{Enc}(2 A_i^2 s_i)$, $P_1$ computes $\text{Enc}(2 A_i^2 A^1_i s_i) = \text{Enc}(2 A_i^2 s_i) A^1_{\ell-1}$ and using $\text{Enc}(A_i^1 s_i)$. $P_2$ computes $\text{Enc}(2 A_i^1 A^2_i s_i) = \text{Enc}(A_i^1 s_i) A^2_{\ell-1}$.

   (d) $P_1$ computes $\text{Enc}(s_{i+1}) = (C^1_i A^1_i) \cdot \text{Enc}(A_i^1 s_i) \cdot \text{Enc}(A_i^2 s_i) = \text{Enc}(A_i^1 A^2_i + A_i^1 s_i + A_i^2 s_i)$.

   (e) $P_2$ computes $\text{Enc}(s_{i+1}) = (C^2_i A^2_i) \cdot \text{Enc}(A_i^1 s_i) \cdot \text{Enc}(A_i^2 s_i) = \text{Enc}(A_i^1 A^2_i + A_i^1 s_i + A_i^2 s_i)$.

6. $P_1$ and $P_2$ independently perform the following steps:

   (a) Compute $\text{Enc}(A_i) = \text{Enc}(A_i^1 A^2_i \text{Enc}(A_i^1 A_i) \text{Enc}(A_i^2 A_i))$,\text{Enc}(A_i A_i)

   (b) Compute $\text{Enc}(Z_i) = \text{Enc}(A_i) \text{Enc}(x_i)$ where $Z_i = x_i \otimes A_i = x_i + A_i - 2x_i A_i$.

   (c) Compute $\text{Enc}(Z)$ as $\text{Enc}(Z) = \prod_{i=0}^{\ell-1} \text{Enc}(Z_i)$ where $Z = \sum_{i=0}^{\ell-1} Z_i$.

7. Set $\text{Enc}(y) \leftarrow \text{Enc}(Z)$. $P_1$ and $P_2$ execute Step 2 to Step 6 log $\log(\ell)$ times.

8. $P_1$ and $P_2$ compute $\text{Enc}(1 - Z)$. Output $\text{Enc}(1 - Z)$.

Fig. 5. Protocol for equality of encrypted numbers using Paillier threshold encryption

A.2 Substitution Cost Protocol

Fig. 6 presents the protocol for computing the substitution cost for constructing the cost matrix. The proof of the security of the protocols can be found in the full paper.

B Graph Edit Operations and Cost Matrix

A standard set of graph edit operations are node insertion (e $\rightarrow$ u), deletion (u $\rightarrow$ e) and substitution (u $\rightarrow$ v) and edge insertion (e $\rightarrow$ e), deletion (e $\rightarrow$ e)
Protocol: Substitution Cost $\pi_{\text{Sub}}$

**Input:** $P_1$’s inputs $\text{Enc}(C_1)$ and $\text{Enc}(\alpha_i)$; $P_2$’s inputs $\text{Enc}(C_2)$ and $\text{Enc}(\beta_j)$;

**Output:** $\text{Enc}(w_{ij})$ with $w_{ij} = c(u_i \rightarrow v_j) = \min\{(c(u_i \rightarrow \epsilon) + c(\epsilon \rightarrow v_j)), |\alpha_i - \beta_j|\}$.

1. $P_1$ computes $\text{Enc}(C_1)$ and $\text{Enc}(\alpha_i)$ and sends these to $P_2$.
2. $P_2$ computes $\text{Enc}(C_2)$ and $\text{Enc}(\beta_j)$ and sends these to $P_1$.
3. $P_1$ and $P_2$ compute $\text{Enc}(C_1 + C_2) = \text{Enc}(c(u_i \rightarrow \epsilon) + c(\epsilon \rightarrow v_j))$ by multiplying $\text{Enc}(C_1)$ and $\text{Enc}(C_2)$.
4. $P_1$ and $P_2$ run $\pi_{\text{CMP}}$ with inputs $\text{Enc}(\alpha_i)$ and $\text{Enc}(\beta_j)$ and obtain $\text{Enc}(b)$.
5. $P_1$ first computes $\text{Enc}(1 - b)$ from $\text{Enc}(b)$ and then computes $\text{Enc}((1 - b)\alpha_i)$ and sends $\text{Enc}((1 - b)\alpha_i)$ to $P_2$.
6. $P_2$ first computes $\text{Enc}(b - 1)$ from $\text{Enc}(b)$ and then computes $\text{Enc}((b - 1)\beta_j)$ and sends $\text{Enc}((b - 1)\beta_j)$ to $P_1$.
7. Both parties compute $\text{Enc}((\alpha_i - \beta_j)) = \text{Enc}((1 - b)\alpha_i) \cdot \text{Enc}((b - 1)\beta_j)$.
8. $P_1$ and $P_2$ run $\pi_{\text{GME}}$ with inputs $\text{Enc}(C_1 + C_2)$ and $\text{Enc}((\alpha_i - \beta_j))$ and obtain $\text{Enc}(b')$. If $b' = 0$, output $\text{Enc}(w_{ij}) = \text{Enc}(C_1 + C_2)$), otherwise output $\text{Enc}((\alpha_i - \beta_j))$.

**Fig. 6.** Protocol for computing node substitution cost $c(u_i \rightarrow v_j)$

and substitution $(e_1 \rightarrow e_2)$ and substitution of node and edge labels where $\epsilon$ denotes empty nodes or edges. The edge edit operations can be defined in terms of the node edit operations as follows. Let $e_1 = (u_1, u_2) \in E_1$ and $e_2 = (v_1, v_2) \in E_2$ where $u_1, u_2 \in V_1 \cup \{\epsilon\}$ and $v_1, v_2 \in V_2 \cup \{\epsilon\}$. An edge substitution operation between $e_1$ and $e_2$, denoted by $e_1 \rightarrow e_2$, is defined as the node substitution operations $u_1 \rightarrow v_1$ and $u_2 \rightarrow v_2$. If there is no edge $e_1$ in $E_1$ and $e_2$ in $E_2$, then the edge insertion in $G_1$, denoted by $(\epsilon \rightarrow e_2)$ is defined by $\epsilon \rightarrow v_1$ and $\epsilon \rightarrow v_2$. Similarly, if there is an edge $e_1$ in $E_1$ and no edge $e_2$ in $E_2$, then the edge deletion, denoted by $(e_1 \rightarrow \epsilon)$ is defined by $u_1 \rightarrow \epsilon$ and $v_2 \rightarrow \epsilon$.

The cost matrix is constructed by considering substitution costs of vertices and the costs of vertex insertions and deletions. The structure of the edit cost matrix $W = (w_{ij})_{(n+m)\times(n+m)}$ has the following form [26]:

$$
C = \begin{bmatrix}
w_{11} & w_{12} & \cdots & w_{1m} & w_{1e} & \infty & \cdots & \infty \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
w_{n1} & w_{n2} & \cdots & w_{nm} & w_{ne} & \infty & \cdots & \infty \\
\infty & w_{c1} & \cdots & \infty & 0 & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\
\infty & \infty & \cdots & w_{cm} & 0 & 0 & \cdots & 0 \\
\end{bmatrix} = \begin{bmatrix}W_1 & W_2 \\
W_3 & W_4 \end{bmatrix}
$$

where the submatrix $W_1$ is corresponding to the cost assignment of nodes $(i \rightarrow j)$, $W_2$ and $W_3$ are corresponding to the cost assignment of node deletion $(i \rightarrow \epsilon)$ and insertion $(\epsilon \rightarrow i)$ of nodes. The insertion and deletion of edges are not taken care of in the cost matrix. However, it is not hard to incorporate the edge substitution cost into the matrix entries. We omit the details here.