Mathematical Modeling and Algorithm Application
ISSN: 3006-0842 | Vol. 8, No. 3, 2026

Dual-Cloud Collaborative Graph Edit Distance Secure Computation Protocol

Yufeng Wang*

School of Computer Science and Technology, Henan Polytechnic University, Jiaozuo, 454000, China

* **Corresponding author**: Yufeng Wang (Email: 212309020074@home.hpu.edu.cn)

Abstract: To address the challenge of securely computing graph edit distance in directed graph scenarios without compromising parties' privacy data, this paper proposes a homomorphic encryption-based, dual-cloud-assisted secure graph edit distance computation protocol. The protocol encodes graph structures as adjacency matrices and introduces random parity noise to protect the original data's parity. It then leverages homomorphic encryption to compute edit distance in ciphertext form. Employing a dual-cloud server architecture, the protocol offloads most computational tasks to the cloud, eliminating direct interaction between data users. This approach enhances computational efficiency while ensuring privacy security. Theoretical analysis and simulation experiments demonstrate the protocol's security under a semi-honest model, with superior user-side computational overhead compared to existing solutions.

Keywords: Privacy Protection, Cloud Computing, Graph Theory, Graph Edit Distance.

1. Introduction

With the rapid advancement of information technology, the internet has not only greatly facilitated information sharing and collaborative computing but also imposed higher demands on data privacy protection. If inadequately safeguarded, sensitive data is vulnerable to leakage during transmission, storage, and processing[1]. Against this backdrop, Secure Multi-Party Computation (MPC) has emerged as a crucial approach for privacy-preserving computation. This technology enables joint computations while safeguarding the privacy of all participants' data, garnering widespread attention[2][3]. Social networks[4], as quintessential web applications, have become vital platforms for establishing and maintaining social relationships. The vast amounts of interaction data generated in these scenarios exhibit distinct graph-structured characteristics. Graph theory, as an effective tool for describing relationships between entities, provides robust support for modeling and analyzing such complex systems[5]. Graph structures intuitively represent entities and their connections, expressing complex information in a visual and orderly manner. Consequently, they are widely used to characterize various practical scenarios such as social relationships, transportation topologies, and biological networks. Particularly in social networks, graphs are often employed to represent users and their interaction relationships: Vertices correspond to users, while edges reflect behavioral connections such as follows, friendships, and reposts. Furthermore, by modeling users' interactions over time (e.g., follow chains, like paths, repost trees) as temporal directed graphs and comparing graph structures across different users or different periods for the same user, similar behavioral patterns can be effectively identified or anomalous behaviors detected. Consequently, efficient management of graph data and its privacy protection have become critical research directions[6][7].

Currently, secure multi-party computation based on graph theory has developed multiple branches, primarily including: secure computation of graph intersections and unions[8], secure subgraph queries[9][10], secure graph path queries[11], and secure measurement of graph edit distance[12][13]. Among these, Graph Edit Distance (GED)[14]serves as a common metric for measuring the similarity between two graphs. It is defined as the minimum cost of operations required to transform one graph into another through a series of edit operations (such as adding, deleting, or replacing vertices or edges). If two graphs have identical structures, the edit distance is zero; a larger distance indicates lower similarity. Considering that replacement operations involve simultaneous adjustments to both vertices and edges, making implementation complex, this paper primarily focuses on the two fundamental operations: insertion and deletion. Existing research has proposed various methods for the secure computation of graph edit distance. For instance, [8]designed a protocol for securely computing graph intersections and unions under malicious models. By combining threshold encryption, zero-knowledge proofs, and encoding techniques, it ensures both the security and practicality of the algorithm. Reference[9]proposes a property subgraph query protocol for cloud environments based on secret sharing and secure shuffling techniques; Reference[10] introduces a subgraph query search protocol that achieves authentication for subgraph queries in large-scale dynamic graph databases using two graph similarity metrics and multiple structures; Reference[11]explores secure multi-party computation protocols for finding shortest paths in graphs. Reference[12]proposes a privacy-preserving subgraph matching protocol based on homomorphic encryption. By computing the graph edit distance between two subgraphs, untrusted cloud servers can perform subgraph matching without decryption. Reference[13]proposes a secure graph edit distance computation protocol based on homomorphic encryption. It designs an encoding computation method and hash function to compute the edit distance of undirected graphs under malicious models. Although existing work has advanced secure graph data processing at different levels, efficiently and accurately computing the edit distance of directed graphs while protecting their structural privacy remains a critical issue in secure multi-party computation and graph data management.

To address the aforementioned issues, this paper focuses on the secure computation of directed graph edit distance and proposes an efficient encoding and computation protocol suitable for Paillier homomorphic encryption. By employing cloud server-assisted computation and optimizing computational workflows, the protocol enhances computational efficiency and data communication efficiency among users while ensuring privacy security. The main contributions of this paper are as follows:

(1)Encoding complex graph structures into matrices through a graph encoding scheme, combined with homomorphic algorithms and random permutations, to securely compute graph edit distances in ciphertext form;

(2)Employing a dual-cloud server-assisted architecture with secure outsourcing to eliminate direct interaction between data users, enabling secure graph edit distance computation among them;

(3)It provides a security proof for the protocol under the semi-honest model, ensuring that even under collusion, the privacy of data users remains protected. Simulation experiments validate that the proposed protocol reduces client-side overhead.

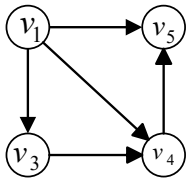
2. Preliminaries

2.1. Encoding Rules

This paper primarily investigates vertex-directed graphs. Based on the encoding method for undirected graphs in[15], we present the encoding rules for directed graphs in this study. Each vertex in the graph possesses a unique identifier, while each edge has a direction and no weight.

Consider a directed complete graph $G=(V,E)$ with vertex set $V=(v_1,v_2,...,v_n)$ and edge set $E=\left(e_1,e_2,...,e_{n(n-1)}\right)$. If the maximum vertex degree in the subgraph G_0 is m, then the graph G_0 can be encoded as an $m\times m$ adjacency matrix M_0 where $M_0[i][j]$ denotes the element at row i and column j of the matrix M_0, satisfying the following rule:

$$M_0[i][j]=\begin{cases}1 & \text{existence edge } v_i\to v_j\ (\text{if } i=j, \text{it implies that there exists a vertex } v_i)\\ 0 & \text{no edge exists } v_i\to v_j\ (\text{if } i=j, \text{it implies that th ere is no vertex } v_i)\end{cases} \tag{1}$$



G_0

As shown in Figure G_0, according to the above rule, it can be converted into the adjacency matrix :

$$M_0=\begin{bmatrix}1&0&1&1&1\\0&0&0&0&0\\0&0&1&1&0\\0&0&0&1&1\\0&0&0&0&1\end{bmatrix} \tag{2}$$

2.2. Paillier Cryptosystem

The Paillier cryptosystem is a public-key cryptosystem based on composite residue classes [16], featuring additive homogeneity and probabilistic encryption. Specifically, the Paillier cryptosystem comprises the following three algorithms:

Key Generation Algorithm $KeyGen(\kappa)\to\{pk,sk\}$: Given a security parameter κ, select two κ-bit prime numbers p and q, compute $N=pq$ and $\lambda=lcm(p-1,q-1)$; subsequently, select a random generator $g\in\mathbf{Z}^*_{N^2}$ satisfying $\gcd\left(L\left(g^\lambda \bmod n^2\right),n\right)=1$, where $L(x)=\dfrac{x-1}{N}$. Finally, output the public key $pk=(N,g)$ and private key $sk=(p,q)$ or λ.

Encryption Algorithm $Enc_{pk}(m)\to c$: Given plaintext $m\in\mathbf{Z}_N$, select a random value $r\in\mathbf{Z}^*_N$, and compute the ciphertext:

$$c=g^m\cdot r^N \bmod N^2 \tag{3}$$

Decryption Algorithm $Dec_{sk}(c)\to m$: For ciphertext $c\in\mathbf{Z}^*_{N^2}$, recover the corresponding plaintext by computing:

$$m=\frac{L\left(c^\lambda \bmod N^2\right)}{L\left(g^\lambda \bmod N^2\right)}\bmod N \tag{4}$$

where $L(u)=\dfrac{u-1}{n}$.

Additive Homogeneity: For two ciphertexts $Enc(m_1)$ and $Enc(m_2)$, the following holds:

$$Dec_{sk}\left(Enc_{pk}(m_2)\cdot Enc_{pk}(m_2)\bmod N^2\right)=m_1+m_2 \bmod N^2 \tag{5}$$

2.3. Random Permutation Operation

Random permutation is the operation of rearranging all elements in a set into a new order. This paper employs the Fisher-Yates shuffling algorithm to process the sequence[17][18]. Table 1details the random permutation generation process.

Table 1 Random Permutation Algorithm

Property	Description
Input	n-dimensional data sequence $A=(a_1,a_2,\ldots,a_n)$
Output	New sequence after random shuffling $A'=\left(a_1',a_2',\ldots,a_n'\right)$
Steps	For each index i from $n-1$ to 1, perform: Generate a random integer j such that $0\le j\le i$ and uniformly distributed; Swap the elements at indices i and j in the sequence A; The final new sequence A' is the random permutation of the original sequence.

2.4. Security Definition in the Semi-Honest Model

In this paper, all entities are assumed to be semi-honest participants. Each entity strictly follows protocol instructions during execution but records all collected information, attempting to infer additional private information from these intermediate outputs. Adversaries corrupt some participants, who adhere to the protocol execution flow while attempting to infer private inputs from other honest participants using all intermediate computations and messages observed during execution. Its formal definition is based on the Ideal/Real Paradigm[19][20], requiring that each participant's view during protocol execution can be generated by a simulator using only its inputs and outputs, and that the simulated view is computationally indistinguishable from the real view.

Definition 1: Let protocol implement functionality. In the semi-honest model, if for any probabilistic polynomial-time (PPT) adversary A corrupting some participants executing the real protocol, there exists a probabilistic polynomial-time simulator S simulating F in the ideal model such that for any legitimate input $\{x_i\}$, the adversary A 's view $View_{\Pi,\mathrm{A}}^{Real}$ and the simulator S 's view $View_{\mathrm{F},\mathrm{S}}^{Ideal}$ are computationally indistinguishable, i.e.:

$$View_{\Pi,\mathrm{A}}^{Real}\left(1^{\lambda},\{x_i\}\right)\triangleq View_{\mathrm{F},\mathrm{S}}^{Ideal}\left(1^{\lambda},\{x_i\}\right) \tag{6}$$

where: $View_{\Pi,\mathrm{A}}^{Real}$ denotes the adversary A 's view in the real protocol Π; $View_{\mathrm{F},\mathrm{S}}^{Ideal}$ denotes the simulator S 's view in the ideal function F;

If the above conditions hold, the protocol Π is said to securely implement the ideal function F.

3. Dual-Cloud-Assisted Graph Edit Distance Security Computation Protocol Π_{PGED}

3.1. System Model

This paper adopts a system model similar to that in[21], as illustrated in Figure 1. This model incorporates two distinct participant types: Data Users (DU) and Cloud Servers (CS). Two data users, Alice and Bob, are present in the model. They aim to compute the graph edit distance between their privacy graphs without disclosing personal privacy data. The cloud servers provide storage and computational services. Two cloud servers participate in the protocol, denoted as CS_1 and CS_2. CS_1 primarily handles storage, while CS_2 processes user data. CS is responsible for computing the graph edit distance on the ciphertext and returning the result to DU. In privacy-preserving computation, relevant computational operations are typically performed in a dual-cloud architecture, where one server stores and processes the relevant data information while the other server executes the computation[22]. In this model, after uploading data to the cloud, the data users can go offline. The two data users can obtain the final result without interacting with each other.

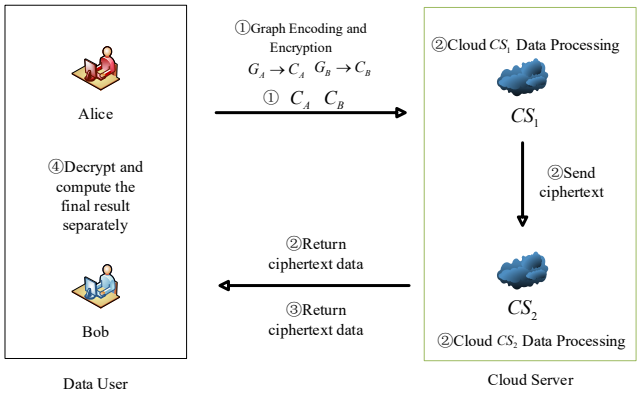


Figure 1 System model

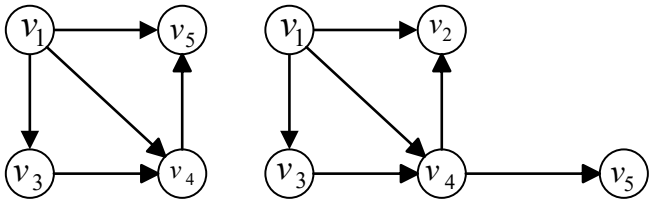
3.2. Calculation Principles

Suppose the vertex set V and edge set E form a directed complete graph $G=(V,E)$, where vertex set $V=\{v_1,v_2,...,v_n\}$ and edge set $E=\left(e_1,e_2,...,e_{n(n-1)}\right)$. Alice holds a subgraph G_A of graph G, and Bob holds a subgraph G_B of graph G. Both parties wish to secretly compute the edit distance between G_A and G_B without revealing information about their respective subgraphs.

Alice and Bob each use a distinct encoding scheme to transform graphs G_A and G_B into adjacency matrices M_A and M_B, storing both vertex and edge information within these matrices. If $M_A[i][j]\neq M_B[i][j]$, it indicates that the edge or vertex exists only in one graph and must be removed (or replaced/added) in the other.

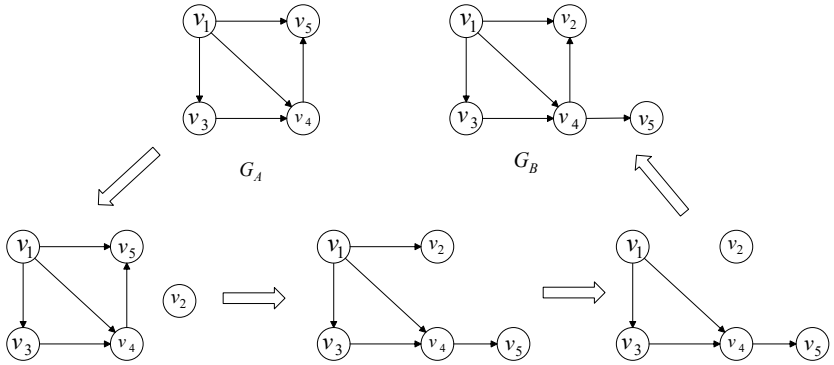
In this paper, the graph edit distance is computed by first performing addition on adjacency matrices M_A and M_B, then counting the number of odd results. This transforms the problem of calculating the edit distance between graphs G_A and G_B into counting the number of odd results when corresponding positions of matrices M_A and M_B are added.

For example, assume the complete set of vertices is $V=\{v_1,v_2,v_3,v_4,v_5\}$. Data holders Alice and Bob possess graphs G_A and G_B respectively, as shown below:



G_A G_B

Where the editing operations on graph $G_A \to G_B$ are: add vertex v_2, remove edge $v_1 \to v_5$, add edge $v_1 \to v_2$, add edge $v_4 \to v_2$.



Following the encoding rules, the adjacency matrices M_A and M_B corresponding to graphs G_A and G_B are obtained:

$$M_A = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, M_B = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

Performing matrix addition yields:

$$M = M_A + M_B = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 2 & 2 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 \\ 0 & 1 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix} \tag{8}$$

The graph edit distance is calculated by counting the number of odd elements in matrix M.

3.3. Protocol Description

Input: Alice inputs a directed graph G_A with maximum vertex degree m, Bob inputs a directed graph G_B with maximum vertex degree n.

Output: The graph edit distance $GED(G_A, G_B)$ between G_A and G_B.

Generate Alice's public-private key pair (pk_A, sk_A) and Bob's public-private key pair (pk_B, sk_B), and generate CS_2's public-private key pair (pk_{CS_2}, sk_{CS_2}). Each entity's private key sk_A、sk_B、sk_{CS_2} is kept strictly confidential, while public keys are disclosed via a trusted channel.

Protocol Flow:

(1) Graph Encoding and Encryption Phase

Given vertex sets $V = \{v_1, v_2, \ldots, v_n\}$ and as an undirected unweighted graph G_A, Alice generates the $m \times m$ adjacency matrix M_A for according to encoding rules. She then randomly generates an $m \times m$ even-multiple full-square matrix R_A and computes:

$$M_A' = M_A + R_A \tag{9}$$

and encrypts the matrix M_A' element-wise using the public key pk_{CS_2} to obtain the ciphertext matrix $C_A[i][j] = E_{pk_{CS_2}}(M_A'[i][j])$. This ciphertext matrix C_A is subsequently sent to the cloud CS_1.

Similarly, Bob generates the $n \times n$ adjacency matrix M_B based on the encoding rules. He then randomly generates the

corresponding even multiple $n \times n$ of the full-row matrix R_B, calculates:

$$M_B' = M_B + R_B \tag{10}$$

and encrypts the matrix M_B' element-wise using the public key pk_{CS_2}, obtaining the encrypted ciphertext matrix $C_B[i][j] = E_{pk_{CS_2}}\left(M_B'[i][j]\right)$. He sends the ciphertext matrix C_B to the cloud CS_1.

(2) Cloud CS_1 Data Processing Phase

The cloud CS_1 receives the ciphertext matrices C_A and C_B. It encrypts zeros and pads the smaller-dimensional matrix (assuming m<n) to obtain ciphertext matrices C_A' and C_B. It calculates the difference matrix between corresponding positions of C_A' and C_B, i.e.:

$$C[i][j] = C_A'[i][j] \cdot C_B[i][j] \tag{11}$$

yielding the difference ciphertext matrix C. This is then converted row-wise to obtain $C' = \left(C[1][1], \ldots, C[i][j], \ldots, C[n][n]\right)$. Subsequently, generate k_1 random odd number $O = \left(o_1, o_2, \ldots, o_{k_1}\right)$ and k_2 random even numbers $E = \left(e_1, e_2, \ldots, e_{k_1}\right)$ satisfying $k_1 + k_2 = k$, encrypt them to obtain ciphertexts C_O and C_E, construct the extended array $C'' = C' \| C_O \| C_E$, apply the random permutation algorithm π to obtain $C''' = \pi(C'')$, and send the obfuscated ciphertext data C''' to CS_2.

Finally, the cloud CS_1 encrypts the values k_1 using Alice's public key pk_A and Bob's public key pk_B respectively:

$$k_A = Enc_{pk_A}(k_1), k_B = Enc_{pk_B}(k_1) \tag{12}$$

then sends the ciphertexts k_A and k_B to the corresponding participants.

(3) Cloud CS_2 Data Processing Phase

Cloud CS_2 receives the obfuscated ciphertext C''' and decrypts it using its private key sk_{CS}:

$$M = Dec_{pk_{CS}}(C''') \tag{13}$$

obtaining the plaintext sequence $M = \left(m_1, \ldots, m_{n^2+k}\right)$. It then applies the odd number counting algorithm(Table 2) to count the number N_O of odd elements within the sequence. Subsequently, it encrypts the count result using Alice's public key pk_A and Bob's public key pk_B respectively:

$$N_O^{(A)} = Enc_{pk_A}(N_O), N_O^{(B)} = Enc_{pk_B}(N_O) \tag{14}$$

and sends the ciphertext $N_O^{(A)}, N_O^{(B)}$ to the corresponding participant.

Table 2 Odd Number Counting Algorithm

Property	Description
Input	$n^2 + k$ Dimensional data sequence $M = \left(m_1, \ldots, m_{n^2+k}\right)$
Output	Number of odd elements in the sequence N_O
Steps	Initialize $N_O = 0$; For indices $i = 1$ to $i = n^2 + k$, perform: If $m_i \bmod 2 \neq 0$, then $N_O \leftarrow N_O + 1$; Repeat step 2 for each index i until all elements are processed; The final value N_O obtained is the number of odd elements in the sequence.

(4) Result Recovery Phase

Alice receives ciphertext k_A from Cloud CS_1 and ciphertext $N_O^{(A)}$ from Cloud CS_2, decrypts them using private key sk_A to obtain:

$$k_1 = Dec_{sk_A}(k_A), N_O = Dec_{sk_A}\left(N_O^{(A)}\right) \tag{15}$$

yielding random value k_1 and statistical result N_O. She calculates the graph edit distance $GED = N_O - k_1$.

Similarly, Bob receives ciphertexts k_B and $N_O^{(B)}$, decrypts them with private key sk_B to obtain values k_1 and N_O, and calculates the graph edit distance $GED = N_O - k_1$.

4. Security Analysis and Proof

4.1. Correctness Analysis

Theorem 1 (Correctness): The correctness of the protocol means that under the semi-honest model, after all participants follow the protocol steps, they can accurately compute the graph edit distance between two graphs, i.e.:

$$GED(G_A, G_B) = \left|\left\{(i,j) \mid M_A[i][j] \neq M_B[i][j]\right\}\right| \tag{16}$$

Proof: The protocol ensures correctness through multiple steps:

During the graph encoding and noise addition phase, Alice and Bob generate random even matrices R_A and R_B, respectively, and compute:

$$M_A' = M_A + R_A, M_B' = M_B + R_B \tag{17}$$

Since each element in R_A and R_B is the sum of even numbers, for any position (i,j):

$$M_A'[i][j] = M_A[i][j] (\mathrm{mod}\, 2), M_B'[i][j] = M_B[i][j] (\mathrm{mod}\, 2) \tag{18}$$

meaning adding even numbers does not alter the parity of matrix elements, thus preserving the parity characteristics of the original matrix. Both parties encrypt M_A' and M_B' using the Paillier homomorphic encryption system, obtaining ciphertext matrices C_A and C_B, which are sent to cloud CS_1.

During the cloud CS_1 data processing phase, cloud CS_1 encrypts zeros and pads one of the matrices with the smaller dimension (assuming (m < n)), yielding ciphertext matrices C_A' and C_B. It then computes in ciphertext form:

$$C[i][j] = C_A'[i][j] \cdot C_B[i][j] \tag{19}$$

By Paillier's additive homomorphism property, decryption yields:

$$Dec(C[i][j]) = M_A'[i][j] + M_B'[i][j] (\mathrm{mod}\, N^2) \tag{20}$$

whose parity matches that of $M_A[i][j] + M_B[i][j]$. If the result is odd, it indicates $M_A[i][j] \neq M_B[i][j]$; if even, it indicates equality. Subsequently, the cloud CS_1 converts the ciphertext matrices into array form to obtain C', randomly adds k_1 odd and k_2 even noise values to it, and performs a random permutation to generate the obfuscated sequence C'''.

During the cloud CS_2 data processing phase, the cloud CS_2 decrypts the obfuscation sequence C''' and counts the total number N_O of odd numbers . Although the cloud CS_1 's random permutation operation destroys traceability of differing positions, the total number of odd numbers only includes the original data count and added odd noise. Clearly:

$$N_O = GED(G_A, G_B) + k_1 \tag{21}$$

During the result recovery phase, Alice and Bob respectively retrieve the encrypted k_1 from Cloud CS_1 and the encrypted N_O from Cloud CS_2 . After decryption, they compute:

$$GED(G_A, G_B) = N_O - k_1 \tag{22}$$

to obtain the accurate graph edit distance.

In summary, the protocol maintains parity through even-numbered noise, ensures computational correctness via homomorphic encryption, and achieves privacy protection through noise obfuscation. The final output aligns with the definition, guaranteeing the protocol's correctness.

4.2. Security Proof

Theorem 2 (Security): Under the semi-honest model, the protocol $\prod_{PGED}$ securely implements the ideal function F_{PGED} for graph edit distance computation.

Proof: The protocol's security requires that even if an adversary compromises one or more entities, they cannot infer the private data of non-compromised parties. By constructing simulators for each compromise scenario, the protocol ensures that in an ideal model where the simulator has access only to the compromised party's input and the ideal function's output, it can simulate a view $View_{\mathrm{F},S}$ that is indistinguishable from the view $View_{\prod_{PGED}}$ computed by the adversary in the actual protocol execution.

Through participation in the protocol, each entity observes data views

$$View_{Alice}^{Real} = \left(G_A, M_A, R_A, M_A', C_A, k_1, k_A, N_O, N_O^{(A)}\right) \tag{23}$$

$$View_{Bob}^{Real} = \left(G_B, R_B, M_B, M_B', C_B, k_1, k_B, N_O, N_O^{(B)}\right) \tag{24}$$

$$View_{CS_1}^{Real} = \left(C_A, C_B, C_A', C, C_O, C_E, C', C'', C''', k_1, k_A, k_B\right) \tag{25}$$

$$View_{CS_2}^{Real} = \left(C''', M, N_O, N_O^{(A)}, N_O^{(B)}\right) \tag{26}$$

Subsequently, simulators are constructed for each compromise scenario to prove the protocol's security.

(1) Constructing the simulator S_{Alice} for the corrupted participant Alice:

The simulator S_{Alice} receives Alice's input G_A and output $GED(A,B)$. It generates matrix M_A according to the encoding rules, randomly generates an even m×m full-rank matrix $\tilde{R}_A$, computes $\tilde{M}_A' = M_A + \tilde{R}_A$ and encrypts it to obtain the ciphertext matrix $\tilde{C}_A = E_{pk_{CS_2}}\left(\tilde{M}_A'[i][j]\right)$; selects a random value $\tilde{k}_1$, computes $\tilde{N}_O = GED(A,B) + \tilde{k}_1$, generates ciphertexts $\tilde{k}_A = Enc_{pk_A}\left(\tilde{k}_1\right)$ and $\tilde{N}_O^{(A)} = Enc_{pk_A}\left(N_O\right)$; and outputs the simulated view $View_{\mathrm{S}_{Alice}}^{Ideal} = \left(G_A, M_A, \tilde{R}_A, \tilde{M}_A', \tilde{C}_A, \tilde{k}_1, \tilde{k}_A, \tilde{N}_O, \tilde{N}_O^{(A)}\right)$.

Due to the IND-CPA security of the Paillier encryption scheme, an adversary cannot distinguish between the ciphertext $\tilde{C}_A, \tilde{k}_A, \tilde{N}_O^{(A)}$ in the simulated view and the ciphertext $C_A, k_A, N_O^{(A)}$ in the genuine view. Since the random square matrix $\tilde{R}_A$, the random value $\tilde{k}_1$, and the genuine view R_A、k_1 are generated identically and share the same distribution, the simulated view and genuine view are computationally indistinguishable, i.e.:

$$View_{Alice}^{Real}(G_A, GED) \triangleq View_{\mathrm{S}_{Alice}}^{Ideal}(G_A, GED) \tag{27}$$

(2) Construct the simulator S_{Bob} for corrupted participant Bob: The procedure is identical to (1).

(3) Construct the simulator S_{CS_1} for the corrupted cloud CS_1 :

The simulator S_{CS_1} has no input. Generate a random ciphertext matrix $\tilde{C}_A, \tilde{C}_B$. By padding and computing the ciphertext matrix $\tilde{C} = \tilde{C}_A' \cdot \tilde{C}_B$, unfold it row-wise to obtain the ciphertext sequence $\tilde{C}'$. Generate random values $\tilde{k}_1$ and

random ciphertext sequences $\tilde{C}_O, \tilde{C}_E$. Merge the ciphertext to obtain $\tilde{C}'' = \tilde{C}' \| \tilde{C}_O \| \tilde{C}_E$, then apply a random permutation to yield the ciphertext $\tilde{C}'''$. Use the public key pk_A, pk_B to encrypt the values $\tilde{k}_1$ to obtain $\tilde{k}_A, \tilde{k}_B$; output the simulated view

$$View_{S_{CS_1}}^{Ideal} = \left(\tilde{C}_A, \tilde{C}_B, \tilde{C}'_A, \tilde{C}, \tilde{C}_O, \tilde{C}_E, \tilde{C}', \tilde{C}'', \tilde{C}''', \tilde{k}_1, \tilde{k}_A, \tilde{k}_B\right).$$

Due to the IND-CPA security of the Paillier encryption scheme, an adversary cannot distinguish the random ciphertext matrix $\tilde{C}_A, \tilde{C}_B, \tilde{C}'_A, \tilde{C}, \tilde{C}_O, \tilde{C}_E$ in the simulated view from the ciphertext $C_A, C_B, C'_A, C, C_O, C_E$ in the real view. Furthermore, owing to the probabilistic nature of Paillier encryption, the padded matrix ciphertext $C_A{}'$ is indistinguishable from the real ciphertext. The random value $\tilde{k}_1$ follows the same distribution as the random value k_1 in the real view. The ciphertext sequence $\tilde{C}', \tilde{C}'', \tilde{C}'''$ is generated identically to the genuine view, rendering the simulated view computationally indistinguishable from the genuine view, i.e.:

$$View_{CS_1}^{Real} \triangleq View_{S_{CS_1}}^{Ideal} \tag{28}$$

(4) Constructing the simulator S_{CS_2} for the corrupted cloud CS_2:

The simulator S_{CS_2} has no input. Generate a random ciphertext sequence $\tilde{C}'''$. Randomly select values $\tilde{N}_O$ and generate a random plaintext sequence $\tilde{M}$, ensuring exactly $\tilde{N}_O$ odd values and the remainder being even. Encrypt the values $\tilde{N}_O$ to produce $\tilde{N}_O^{(A)} = Enc_{pk_A}\left(\tilde{N}_O\right), \tilde{N}_O^{(B)} = Enc_{pk_B}\left(\tilde{N}_O\right)$, then output the simulated view

$$View_{CS_2}^{Ideal} = \left(\tilde{C}''', \tilde{M}, \tilde{N}_O, \tilde{N}_O^{(A)}, \tilde{N}_O^{(B)}\right).$$

Due to the IND-CPA security of the Paillier cryptosystem, the computation of the random ciphertext sequence $\tilde{C}''', \tilde{N}_O^{(A)}, \tilde{N}_O^{(B)}$ in the simulated view is indistinguishable from the ciphertext $C''', N_O^{(A)}, N_O^{(B)}$ in the real view. For the real value $N_O = GED(A,B) + k_1$, since CS_1, CS_2 is not colluding, the adversary cannot obtain k_1, and thus cannot distinguish between N_O and $\tilde{N}_O$. The plaintext sequence M has its data positions scrambled by adding random values and random values, preventing the adversary from distinguishing between elements of the real sequence and the simulated sequence. Therefore, the computation of the simulated view is indistinguishable from the real view, i.e.:

$$View_{CS_2}^{Real} \triangleq View_{S_{CS_2}}^{Ideal} \tag{29}$$

(5) Constructing a simulator S_{Alice,CS_1} for collusion between Alice and the cloud CS_1:

The simulator S_{Alice,CS_1} obtains the input G_A and output $GED(A,B)$. Based on the encoding rules, it generates a matrix M_A, randomly generates an even m-order fully-pseudo-random matrix $\tilde{R}_A$, computes $\tilde{M}_A{}' = M_A + \tilde{R}_A$, encrypts it to obtain the ciphertext matrix $\tilde{C}_A = E_{pk_{CS_2}}\left(\tilde{M}_A{}'[i][j]\right)$;Generate a random ciphertext matrix $\tilde{C}_B$. By padding and computing the ciphertext matrix $\tilde{C} = \tilde{C}'_A \cdot \tilde{C}_B$, unfold it row-wise to obtain the ciphertext sequence $\tilde{C}'$. Generate random values and a random ciphertext sequence $\tilde{C}_O, \tilde{C}_E$. Merge the ciphertext to obtain $\tilde{C}'' = \tilde{C}' \| \tilde{C}_O \| \tilde{C}_E$, then apply a random permutation to yield the ciphertext $\tilde{C}'''$. Use the public key pk_A, pk_B to encrypt $\tilde{k}_1$ and obtain $\tilde{k}_A, \tilde{k}_B$. Compute $\tilde{N}_O = GED(A,B) + \tilde{k}_1$ from the output $GED(A,B)$ and the random value $\tilde{k}_1$, then encrypt it with the public key pk_A to obtain $\tilde{N}_O^{(A)}$. Output the simulated view

$$View_{S_{Alice,CS_1}}^{Ideal} = \left(\tilde{C}_A, \tilde{C}_B, \tilde{C}'_A, \tilde{C}, \tilde{C}_O, \tilde{C}_E, \tilde{C}', \tilde{C}'', \tilde{C}''', \tilde{k}_1, \tilde{k}_A, \tilde{k}_B, \tilde{N}_O, \tilde{N}_O^{(A)}\right) \tag{30}$$

Due to the IND-CPA security of the Paillier cryptosystem, the random ciphertext matrix $\tilde{C}_A, \tilde{C}_B, \tilde{C}'_A, \tilde{C}$ in the simulated view is indistinguishable from the computation of C_A, C_B, C'_A, C in the real view; the random ciphertext sequence $\tilde{C}, \tilde{C}_O, \tilde{C}_E$ is indistinguishable from the ciphertext sequence in the real view; the ciphertext $\tilde{C}', \tilde{C}'', \tilde{C}'''$ is constructed identically to the real view; the random value $\tilde{k}_1, \tilde{N}_O$ has the same distribution as the real value; and the random ciphertext $\tilde{N}_O^{(A)}$ is indistinguishable from the real ciphertext $N_O^{(A)}$. Therefore, the simulated view and real view are computationally indistinguishable, i.e.:

$$View_{CS_1}^{Real}\left(G_A, GED\right) \triangleq View_{S_{Alice,CS_1}}^{Ideal}\left(G_A, GED\right) \tag{31}$$

(6) Construct Bob and the cloud CS_1's collusive simulator S_{Bob,CS_1} : Its construction method is identical to (5).

5. Performance Analysis

5.1. Theoretical Analysis

This section compares the efficiency of the proposed scheme with that of related literature schemes.

(1) Computational Complexity Analysis: Since Reference[12]employs the BGN encryption scheme, while

References [13]and the proposed protocol both utilize the Paillier encryption scheme, the computational complexity is measured by the more costly modular exponentiation operation, with other factors disregarded.

Assuming both Alice and Bob in [13]hold graphs with m vertices each, they respectively encrypt each element in the extended one-dimensional data A^* and B^*, totaling $m(m+1)$ elements. Since the Paillier encryption scheme requires two modular exponentiations per encryption operation, it necessitates $4m(m+1)$ modular exponentiations. Subsequently, each corresponding value is retrieved via a single decryption operation. Given that Paillier decryption requires 2 modular exponentiations per operation, this necessitates 4 modular exponentiations. In summary, the graph edit distance between two graphs in [13] requires a total of $4m(m+1)+4$ modular exponentiations.

Assume that in [12], the query graph H contains m vertices. First, encrypt each element of the query graph's adjacency matrix, totaling m^2 elements. Since the BGN encryption scheme requires two modular exponentiations per encryption operation, this step requires $2m^2$ modular exponentiations. The original graph G contains $n,(m \le n)$ vertices. Encrypting n^2 elements of the original graph adjacency matrix requires $2n^2$ modular exponentiation operations. Next, the cloud server pairs encrypted edges in the lower triangular positions $\frac{m(m+1)}{2}$ of the query graph and original graph, requiring modular exponentiation operations. Finally, the query user decrypts the returned data to obtain the result, requiring 1 modular exponentiation operation. In summary, [12]calculates that computing the graph edit distance between two graphs requires a total of $2m^2+2n^2+\frac{m(m+1)}{2}+1$ modular exponentiation operations.

In this protocol, Alice holds a graph with m vertices, and Bob holds a graph with n vertices. First, Alice and Bob respectively encrypt each element of the blinded adjacency matrices M_A' and M_B', with the number of elements being m^2 and n^2, requiring $2m^2+2n^2$ modular exponentiation operations. Subsequently, the cloud CS_1 encrypts k random values and encrypts value k_1 using Alice's and Bob's public keys respectively, requiring $2(k+2)$ modular exponentiation operations. Next, the cloud CS_2 decrypts n^2+k ciphertexts and encrypts values N_O using Alice's and Bob's public keys respectively, requiring $2(n^2+k+2)$ modular exponentiation operations. Finally, Alice and Bob decrypt and compute results using their respective private keys, requiring 8 modular exponentiation operations. In summary, the protocol in this paper requires a total of $2m^2+4n^2+4k+16$ exponential operations to compute the graph edit distance between two graphs, with the two data users collectively requiring $2m^2+2n^2+8$ exponential operations.

(2) Communication Complexity Analysis: Communication complexity is typically measured by the number of communications or the amount of data transmitted. This paper analyzes and compares the literature scheme with the proposed protocol from both perspectives.

In [13], since Alice and Bob follow identical procedures, we use Alice as an example: First, Alice sends the ciphertext data $E_{pk_A}(A^*)$ to the other party, requiring one communication session and transmitting $m(m+1)$ ciphertexts; Next, Alice sends the hash value H_A and the ciphertext data C_A, requiring one communication session and transmitting one ciphertext; Subsequently, Alice decrypts and sends the value C_b, requiring one communication session and transmitting one data item; Finally, Alice computes and sends to G_A, requiring one communication session and transmitting one data item. In total, Alice communicates four times, transmitting $m(m+1)+4$ data items. Similarly, Bob communicates four times, transmitting $m(m+1)+4$ data items. Thus, the protocol in [13] requires eight communication sessions in total, transmitting $2m(m+1)+8$ data items.

In [12], first, the querying user sends data $\{ID_{QU} \| H_{QU} \| E(E^H) \| TS_{QU} \| MAC_{QU}\}$ to the server, requiring one communication session and transmitting $\frac{m(m+1)}{2}+m+3$ data item. Next, the data owner sends data $\{ID_{DO} \| H_{DO} \| E(E^G) \| TS_{DO} \| MAC_{DO}\}$ to the server, requiring one communication session and transmitting $\frac{n(n+1)}{2}+n+3$ data item. Finally, the server returns data to the querying user by sending $\{ID_{CS} \| C \| T_{CS} \| MAC_{CS}\}$, requiring one communication session and transmitting four data items. Thus, [12]involves three communication sessions in total, transmitting $\frac{m(m+1)}{2}+m+\frac{n(n+1)}{2}+n+10$ data items.

In the protocol described herein, since the processes for Alice and Bob are identical, the following description uses Alice as an example. First, Alice sends the ciphertext data C_A to cloud CS_1, involving one communication session and transmitting m^2 data units. Similarly, Bob also communicates once, transmitting n^2 data units. Subsequently, cloud CS_1 sends data C''' to cloud CS_2 and sends the ciphertext k_A to Alice, involving two communication sessions and transmitting n^2+k+1 data units. Finally, Cloud CS_2 sends data $N_O^{(A)}$ and $N_O^{(B)}$ to Alice and Bob respectively, requiring 2 communication rounds and transmitting 2 data units. Thus,

the protocol in this paper involves 6 communication rounds in total, transmitting m^2+2n^2+k+3 data units. Among these, two data units are transmitted without user interaction, and a total of m^2+n^2 data units are sent to the cloud.

Based on the above analysis, Table 3 presents a comparison between the protocol in this paper and those in [12]and [13].

Table 3 Comparison between Protocol in This Paper and Existing Protocols

Protocol	Computational Complexity	Communication Complexity	Number of Communications	Type
Reference[12]	$4m(m+1)+4$	$2m(m+1)+8$	8	Undirected Graph
Reference[13]	$2m^2+2n^2+\frac{m(m+1)}{2}+1$	$\frac{m(m+1)}{2}+m+\frac{n(n+1)}{2}+n+10$	3	Undirected graph
This paper	$2m^2+4n^2+4k+16$	m^2+2n^2+k+3	6	Directed graph

5.2. Simulation Experiment

This paper verifies and compares the efficiency of various protocols by simulating and testing the time consumption of the protocols in [12] and [13] alongside the protocol proposed herein.

The experimental environment is as follows: Windows 10 64-bit operating system, 12th Gen Intel® Core™ i5-12400F 2.50 GHz processor, 32 GB RAM. Implementation is in Python, using the phe library for the Paillier encryption scheme and random. shuffle for random permutations. Both the protocols in [12][13] and the present protocol employ homomorphic encryption schemes with a large prime number p set to 512 bits.

The experiments aim to securely compute the graph edit distance between two graphs. To test the overall runtime of the protocols in [12][13] under the same parameter set as the present protocol, we set . To ensure data accuracy, each parameter set undergoes 10 experimental runs. [12][13] and the proposed protocol under identical parameter sets. To ensure consistent vertex counts in all comparison protocols, we set. For data accuracy, each parameter set underwent 10 independent runs, with results averaged. Experimental outcomes are presented in Figure 2.

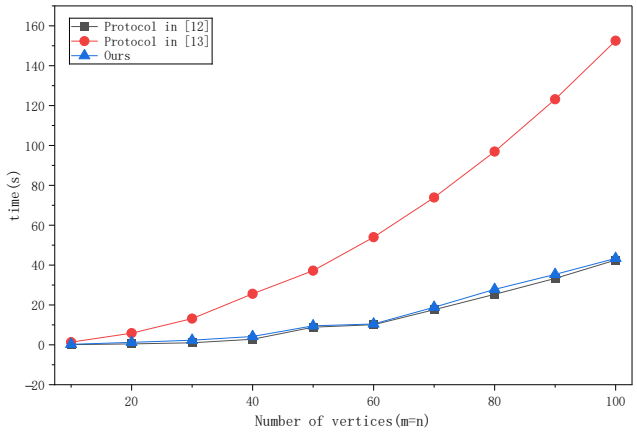


Figure 2 Comparison of running time with other protocols

As shown in Figure 2, the running time of the proposed protocol is slightly higher than that of [13] but lower than that of [12]. This is because the proposed protocol focuses on directed graphs, which incur significant computational overhead in vertex and edge processing. However, by delegating most computations to two cloud servers, Alice and Bob only perform encoding, encryption, and decryption operations, substantially reducing computational burden for the two data users. Therefore, although the overall running time of the proposed protocol is marginally higher than that of [13], the actual execution time for the two data users is significantly less than this total. Furthermore, the figure shows that as the number of vertices in the graph increases, the runtime of the protocol proposed in [12] grows substantially, consistently exceeding that of the proposed protocol. Moreover, its growth rate is noticeably faster than that of the proposed protocol. Although the computational overhead of the proposed protocol also increases when the number of vertices for the data users is large, it remains far lower than that of the protocol in [12].

6. Conclusion

This paper focuses on secure computation of directed graph edit distance under privacy protection, proposing a security protocol that integrates homomorphic encryption with dual-cloud collaboration. By transforming graph structures into matrices and incorporating random value obfuscation techniques, computations can be performed in encrypted form. Leveraging a dual-server architecture, core computational tasks are offloaded to the cloud, significantly reducing computational and communication burdens on the user side. Security analysis demonstrates that the protocol effectively resists collusive attacks under the semi-honest model, safeguarding all parties' data privacy. Experimental evaluations reveal superior performance in data user overhead compared to some existing methods. Future work may explore security-enhancing protocols under malicious models.

References

[1] Zhao, C., Zhao, S., Zhao, M., Chen, Z., Gao, C. Z., Li, H., & Tan, Y. A. (2019). Secure multi-party computation: theory, practice and applications. Information Sciences, 476, 357-372.

[2] Goldreich, O. (1998). Secure multi-party computation. Manuscript. Preliminary version, 78(110), 1-108.

[3] Pang, H., & Wang, B. (2020). Privacy-preserving association rule mining using homomorphic encryption in a multikey environment. IEEE Systems Journal, 15(2), 3131-3141.

[4] Abawajy, J. H., Ninggal, M. I. H., & Herawan, T. (2016). Privacy preserving social network data publication. IEEE communications surveys & tutorials, 18(3), 1974-1997.

[5] Prathik, A., Uma, K., & Anuradha, J. (2016). An overview of application of graph theory. International Journal of ChemTech Research, 9(2), 242-248.

[6] Ding, X., Wang, C., Choo, K. K. R., & Jin, H. (2019). A novel privacy preserving framework for large scale graph data publishing. IEEE transactions on knowledge and data engineering, 33(2), 331-343.

[7] Arshad, M. U., Kundu, A., Bertino, E., Ghafoor, A., & Kundu, C. (2017). Efficient and scalable integrity verification of data

and query results for graph databases. IEEE Transactions on Knowledge and Data Engineering, 30(5), 866-879.

[8] Liu, X., Tu, X. F., Luo, D., Xu, G., Xiong, N. N., & Chen, X. B. (2023). Secure multi-party computation of graphs' intersection and union under the malicious model. Electronics, 12(2), 258.

[9] Wang, S., Zheng, Y., Jia, X., Huang, H., & Wang, C. (2022). OblivGM: Oblivious attributed subgraph matching as a cloud service. IEEE Transactions on Information Forensics and Security, 17, 3582-3596.

[10] Sharmila, G., & Kavitha Devi, M. K. (2024). BTLA-LSDG: Blockchain-based triune layered architecture for authenticated subgraph query search in large-scale dynamic graphs. IETE Journal of Research, 70(2), 1495-1518.

[11] Ghosh, E., Kamara, S., & Tamassia, R. (2021, May). Efficient graph encryption scheme for shortest path queries. In Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (pp. 516-525).

[12] Zuo, X., Li, L., Peng, H., Luo, S., & Yang, Y. (2020). Privacy-preserving subgraph matching scheme with authentication in social networks. IEEE Transactions on Cloud Computing, 10(3), 2038-2049.

[13] Liu, X., Kong, J., Peng, L., Luo, D., Xu, G., Chen, X., & Liu, X. (2023). A Secure Multi-Party Computation Protocol for Graph Editing Distance against Malicious Attacks. Mathematics, 11(23), 4847.

[14] Sanfeliu, A., & Fu, K. S. (2012). A distance measure between attributed relational graphs for pattern recognition. IEEE transactions on systems, man, and cybernetics, (3), 353-362.

[15] Wei, Q., Li, S. D., Wang, W. L., & Du, R. M. (2020). Secure multi-party computation of graph intersection and union. J. Cryptologic Res, 7, 774-788.

[16] Paillier, P. (1999, April). Public-key cryptosystems based on composite degree residuosity classes. In International conference on the theory and applications of cryptographic techniques (pp. 223-238). Berlin, Heidelberg: Springer Berlin Heidelberg.

[17] Fisher, R. A., & Yates, F. (1953). Statistical tables for biological, agricultural and medical research. Hafner Publishing Company.

[18] Durstenfeld, R. (1964). Algorithm 235: random permutation. Communications of the ACM, 7(7), 420.

[19] Lindell, Y. (2017). How to simulate it–a tutorial on the simulation proof technique. Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich, 277-346.

[20] Feng, D., & Yang, K. (2022). Concretely efficient secure multi-party computation protocols: survey and more. Security and Safety, 1, 2021001.

[21] Gao, W., & Yu, J. (2025). Enabling Privacy-Preserving Top-k Hamming Distance Query on the Cloud. IEEE Transactions on Network and Service Management.

[22] Ge, X., Yu, J., & Hao, R. (2023). Privacy-preserving graph matching query supporting quick subgraph extraction. IEEE Transactions on Dependable and Secure computing, 21(3), 1286-1300.