Article

A Secure Multi-Party Computation Protocol for Graph Editing Distance against Malicious Attacks

Xin Liu [1,2], Jianwei Kong [1], Lu Peng [3], Dan Luo [4,*], Gang Xu [5], Xiubo Chen [2] and Xiaomeng Liu [1]

1 School of Information Engineering, Inner Mongolia University of Science and Technology, Baotou 014010, China; lx2001lx@imust.edu.cn (X.L.); 2022022346@stu.imust.edu.cn (J.K.); 2021023305@stu.imust.edu.cn (X.L.)
2 State Key Laboratory of Network and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China; xb_chen@bupt.edu.cn
3 Beijing Institute of Computer Technology and Applications, Beijing 100039, China; jialk@bupt.edu.cn
4 Department of Computer Science, Tianjin Renai University, Tianjin 301636, China
5 School of Information Science and Technology, North China University of Technology, Beijing 100144, China; gx@ncut.edu.cn
* Correspondence: rdjeans@gmail.com; Tel.: +86-136-1996-1021

Abstract: The secure computation of the graph structure is an important element in the field of secure calculation of graphs, which is important in querying data in graphs, since there are no algorithms for the graph edit distance problem that can resist attacks by malicious adversaries. In this paper, for the problem of secure computation of similarity edit distance of graphs, firstly, the encoding method applicable to the Paillier encryption algorithm is proposed, and the XOR operation scheme is proposed according to the Paillier homomorphic encryption algorithm. Then, the security algorithm under the semi-honest model is designed, which adopts the new encoding method and the XOR operation scheme. Finally, for the malicious behaviors that may be implemented by malicious participants in the semi-honest algorithm, using the hash function, a algorithm for secure computation of graph editing distance under the malicious model is designed, and the security of the algorithm is proved, and the computational complexity and the communication complexity of the algorithm are analyzed, which is more efficient compared with the existing schemes, and has practical value. The algorithm designed in this paper fills the research gap in the existing literature on the problem of graph edit distance and contributes to solving the problem.

Keywords: secure multi-party computation; graph editing distance; XOR; hash function; malicious model

MSC: 14-06

check for updates

Citation: Liu, X.; Kong, J.; Peng, L.; Luo, D.; Xu, G.; Chen, X.; Liu, X. A Secure Multi-Party Computation Protocol for Graph Editing Distance against Malicious Attacks. *Mathematics* **2023**, *11*, 4847. https://doi.org/10.3390/math11234847

Academic Editor: Shaomin Wu

Received: 28 October 2023
Revised: 17 November 2023
Accepted: 29 November 2023
Published: 1 December 2023

1. Introduction

The rapid growth of the Web not only made it easy for participants to jointly perform information search and computation, but also poses a great challenge to participants' data security, which can easily lead to data privacy leakage if not properly protected [1,2]. Using secure multi-party computation, which can fully utilize the data while protecting their privacy, has become a powerful tool for privacy protection [3–6], which makes secure multi-party computation gain more and more attention. Secure multi-party computation (MPC) is an important branch of cryptography [7], which aims to solve the problem of cooperative computing between a group of malicious parties on the premise of protecting private data. In the whole process of the implementation of the MPC algorithms, data calculation can be completed without relying on any third party, and the participants can only obtain the calculation results and cannot obtain the private data of other participants.

Graph theory is a scientific modeling and data analysis tool that is the most effective model for studying the interconnections and interactions between things [8]. Graphs can visually represent the connection between things, and make the messy information orderly, intuitive, and clear, so almost everything can be expressed by graphs. Secure multi-party computation for graph theory focuses on the secure computation of graph intersection and union [9–13], the secure query of subgraphs [14–18], the secure query of graph edit distances [19–24], and the secure query of graph paths [25–27].

A dynamic graph is a chart in which data changes over time or on a time axis, and these changes are displayed in real time through animation or interaction [28]. Dynamic graphs are often used to show trends, patterns, correlations, or other data characteristics over time. The graph editing distance (GED) can be used as a measure of similarity between two graphs. The GED between two graphs can be defined as $GED(G_A, G_B)$. If the two graphs are the same, the editing distance is 0; if the editing distance between the two graphs is large, the two graphs are less similar. The GED refers to the minimum number of operations required to convert one graph into another by adding, deleting, and replacing edges and vertices, but considering that the replacement operations in graphs involve both edges and vertices, which are more tedious, only deletion and addition operations are considered in this paper. Reference [10] designed a new encoding method that represents the vertices in a graph in a special matrix, and combined it with the Lifted-ElGamal threshold cryptosystem to design an MPC algorithm of graph intersection and union sets. Reference [21] applied the method of an adjacency matrix and angle vectors to transform the graph into an adjacency matrix, and used the equality of corresponding interior angles and proportionality of corresponding edges as well as hash functions to confidentially determine graph similarity. Reference [22] proposed a new similarity measure using a depth-first search combined with Levenshtein distance to transform graph matching into a string matching problem, and designed a graph similarity editing distance algorithm. Reference [23] computes the approximate GED and mentions the determination of a fault-tolerant graph matching scheme. Reference [26] studied the MPC problem of shortest paths in graphs.

In this paper, for the problem of secure computation of graph editing distance, we propose a coding method applicable to the Paillier encryption algorithm, which is simpler and more efficient. At the same time, this is for the problem that the algorithm under the semi-honest model cannot resist the attack of a malicious adversary, and there are problems such as privacy leakage, even possible errors of judgement, etc. We design a algorithm for graph editing distance under the malicious model that can resist the attack of a malicious adversary.

The contributions are as follows:

(1) First, an encoding method applicable to the Paillier encryption algorithm is proposed, which is simpler and more efficient.
(2) Using the Paillier additive homomorphism and XOR homomorphism, the Paillier encryption scheme used to realize the secure XOR operation scheme is proposed to facilitate the problem of confidential calculation of the graph editing distance for research.
(3) An MPC algorithm of graph editing distance under the semi-honest model is designed, and analyzed for correctness using the coding methods and XOR operation schemes, and the algorithm has been analyzed for correctness.
(4) With the help of hash functions, an MPC algorithm for GED that can resist malicious attacks is designed for the malicious behaviors that may be committed by malicious participants. The security of the algorithm is demonstrated using the real/ideal model paradigm, and the efficiency of the algorithm is verified with the efficiency analysis and experimental simulations.

The rest of the paper is organized as follows: Section 2 introduces some basic tools needed to construct secure algorithms, edit distance encoding rules, and security definitions of algorithms; Section 3 constructs edit distance secure algorithms for graphs under semi-honesty and analyzes their correctness; Section 4 constructs edit distance secure MPC algorithms for graphs under malicious models and analyzes and proves the security of the algorithms under malicious models; Section 5 analyzes the performance of the algorithms and presents their engineering applications; Section 6 summarizes the paper.

2. Related Work

2.1. Paillier Cryptosystem

The Paillier encryption system is a public key cryptosystem with the additive homomorphism and semantic security [29]. The description is as follows:

(1) Key generation: Set the security parameter k to generate large prime numbers p, q, satisfying $\gcd(pq,(p-1)(q-1)) = 1$. Calculate $N = pq$, $\lambda = \mathrm{lcm}(p-1, q-1)$, where lcm denotes the least common multiple. Choose $g \in Z_N^*$ randomly to satisfy $\gcd\left(L\left(g^\lambda \mathrm{mod} N^2\right), N\right) = 1$, where $L(x) = \frac{x-1}{N}$. The public key is (g, N) and the private key is λ.

(2) Encryption process: An arbitrary plaintext message $m \in Z_N$, and arbitrarily chosen random number $r \in Z_N^*$, computed to obtain the ciphertext $C = E(m) = g^m r^N \mathrm{mod} N^2$.

(3) Decryption process: For the ciphertext $C \in Z_{N^2}^*$, calculate $m = \frac{L\left(c^\lambda \mathrm{mod} N^2\right)}{L\left(g^\lambda \mathrm{mod} N^2\right)} \mathrm{mod} N$.

Additive homomorphism: For any plaintext $m_1, m_2 \in Z_N$ and any $r_1, r_2 \in Z_N^*$, corresponding to two ciphertexts $c_1 = E(m_1, r_1)$ and $c_2 = E(m_2, r_2)$, they satisfy $c_1 \cdot c_2 = E(m_1, r_1) \cdot E(m_2, r_2) = g^{m_1+m_2}(r_1 \cdot r_2)^N \mathrm{mod} N^2 = E(m_1 + m_2) \mathrm{mod} N^2$.

In addition, the Paillier additive homomorphism can implement the XOR operation between binary numbers.

Theorem 1. *The Paillier encryption algorithm allows for the implementation of an XOR operation on binary numbers:* $E(a \oplus b) = E(a \bullet \overline{b} + \overline{a} \bullet b) = E(a)^{\overline{b}} E(\overline{a})^b = E(\overline{b})^a E(b)^{\overline{a}}$.

Suppose a, b is a binary string, i.e., $a = a_1 a_2 \ldots a_n$, $b = b_1 b_2 \ldots b_n$. If the elements of the corresponding positions of a, b are XOR by position, and the summation operation is carried out for each bit $c_i = a_i \oplus b_i (i \in [1, n])$ of the resulting binary string $c = c_1 c_2 \ldots c_n$, the result $\sum_{i=1}^{n} c_i$ represents the number of different elements in the corresponding positions of a, b. The above process can be expressed as $E(\sum_{i=1}^{n} a_i \oplus b_i) = \prod_{i=1}^{n} E(a_i)^{\overline{b_i}} E(\overline{a_i})^{b_i}$ (or $\prod_{i=1}^{n} E(\overline{b}_i)^{a_i} E(b_i)^{\overline{a}_i}$).

2.2. Hash Function

The hash function $h = H(M)$ can take the data M of different lengths as input and produce a fixed-length hash value [30]. For larger sets of inputs, $M_1, M_2, \ldots, M_n$ are processed separately using the hash function, and the resulting output result $h_1, h_2, \ldots, h_n$ has uniformly distributed, seemingly random values for each element. If one or several bits of the data M are changed, a different hash value will be generated.

The hash function can be used for message authentication, a way to verify the integrity of a message. It ensures that the data received by the receiver do match the data sent by the sender and that there are no unauthorized modifications, insertions, deletions, and replay.

2.3. Coding Rules

Let an undirected complete graph $G = (V, E)$ have a vertex set $V = \{v_1, v_2, \ldots, v_m\}$ and edge set $E = (e_1, e_2, \cdots e_m)$. The subgraph G_0 can be encoded by a $m \times m$ matrix M_0 where the elements M_{ij} are the following values:

$$\mathrm{M_{ij}} = \begin{cases} 1, \text{ an edge exists between } \mathrm{v_i}, \mathrm{v_j} (\text{if } \mathrm{i} = \mathrm{j}, \text{ means the existence of vertex } \mathrm{v_i}); \\ 0, \text{ no edge exists between vertices } \mathrm{v_i}, \mathrm{v_j} (\text{if } \mathrm{i} = \mathrm{j}, \text{ means vertex } \mathrm{v_i} \text{ does not exist}). \end{cases}$$

G_0 is an undirected graph, and it is easy to know that M_0 is symmetric about the diagonal. The diagonal and following elements of M_0 are taken out by rows and arranged sequentially, which can form a one-dimensional array $X = (x_1, x_2, \ldots, x_k)$, $k = \frac{m(m+1)}{2}$. M_0 is shown as follows:

$$M_0 = \begin{bmatrix} 1 & 1 & \cdots & 0 & 1 \\ 1 & 1 & \cdots & 1 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 1 & \cdots & \ddots & \vdots \\ 1 & 0 & \cdots & \cdots & 1 \end{bmatrix}$$

Alice and Bob generate matrices M_A and M_B of size $m \times m$, respectively, according to the encoding method, and the elements of the diagonal and below are taken out by rows and arranged sequentially to form the arrays $A = (a_1, a_2, \ldots, a_k)$ and $B = (b_1, b_2, \ldots, b_k)$. To calculate the editing distance between two graphs, it is only necessary to calculate the number of unequal elements in the corresponding positions in the array A, B. The Paillier encryption algorithm is used to realize the secure XOR operation, and the basic idea is as described in Theorem 1. A, B is expanded to $A^* = \left(a_1^*, a_2^*, \ldots, a_{2k}^*\right) = (a_1, \ldots a_k, \overline{a_1}, \ldots, \overline{a_k})$ and $B^* = \left(b_1^*, b_2^*, \ldots, b_{2k}^*\right) = \left(\overline{b_1}, \ldots, \overline{b_k}, b_1, \ldots b_k\right)$, respectively. Alice encrypts A^* and sends it to Bob, who selects the ciphertext at the corresponding position according to the element with value '1' in B^*, and multiplies the selected ciphertext to obtain a new ciphertext to Alice, who decrypts it and obtains the result, which is the graph G_A and the editing distance of G_B.

Example 1. *Suppose the full set of vertices is $V = (v_1, v_2, \ldots, v_5)$, Alice has subgraph G_A, and Bob has subgraph G_B, as shown in Figures 1 and 2:*

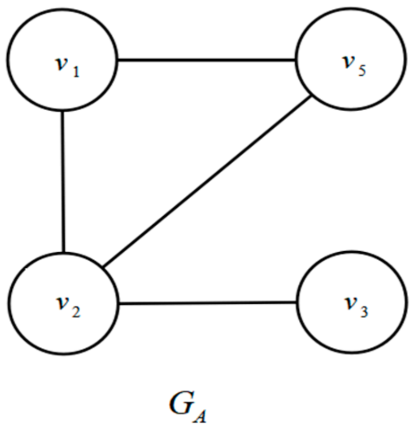


Figure 1. Alice's subgraph G_A.

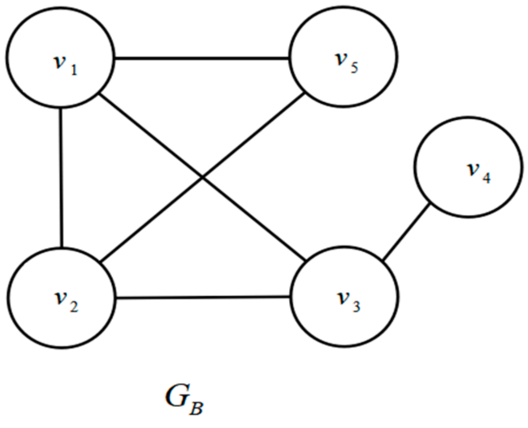


Figure 2. Bob's subgraph G_B.

According to the encoding method, the matrix M_A, M_B is generated.

$$M_A = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}, \; M_B = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Alice and Bob form arrays of diagonals and the following elements of matrices M_A and M_B, respectively:

$$A = (1,1,1,0,1,1,0,0,0,0,1,1,0,0,1),\; B = (1,1,1,1,1,1,0,0,1,1,1,1,0,0,1).$$

Alice will then transform A into A^*, i.e., $A^* = (1,1,1,0,1,1,0,0,0,0,1,1,0,0,1,0,0,0,1,0,0,1,1,1,1,0,0,1,1,0)$, and Bob transforms B into B^*, i.e., $B^* = (1,1,1,1,1,1,0,0,1,1,1,1,0,0,1,0,0,0,0,0,0,1,1,0,0,0,0,1,1,0)$.

Alice encrypts the array A^* using the Paillier algorithm and $E(A^*)$ is sent to Bob, who selects the ciphertext $E(a_i^*) = (1,1,1,0,1,1,0,0,1,1,1,1,1,1,1)$ from $E(A^*)$ at the corresponding position according to the element b_i^* with B^* value '1′ and computes $C = \prod_{b_i^*} E(a_i^*)$. Alice decrypts C to obtain $c = D(C) = 12$, i.e., the edit distance of graph G_A, G_B is 12. This is graphically represented as follows. As shown in Figure 3.

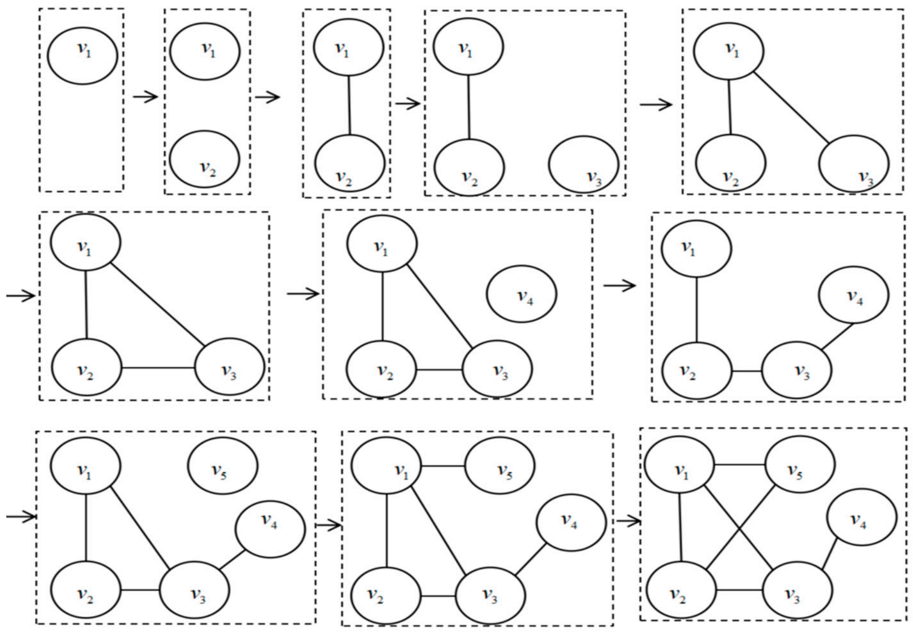


Figure 3. Graphics conversion process.

2.4. Security under the Malicious Model

The malicious model [31] is a pervasive model for secure multi-party computation, but it is difficult to design because it can prevent or detect malicious behaviors. To prove that a Algorithm is secure under the malicious model, it must be shown that it meets the security definition under the malicious model, i.e., a Algorithm is secure if the real Algorithm can reach the same level of security as the ideal Algorithm.

Ideal Algorithm: P_1 and P_2 have private data x and y, respectively, and P_1 and P_2 want to jointly compute the function $f(x,y) = (f_1(x,y), f_2(x,y))$. The computation process requires a trusted third party (TTP), and finally both parties obtain the results $f_1(x,y)$ and $f_2(x,y)$, respectively. The concrete implementation process is as follows:

(1) P_1 and P_2 send x and y to TTP, respectively. If $P_i(i = 1,2)$ is honest, the correct data are sent to TTP. If P_i is malicious, it may send false input x' or y' based on the private

data, or it may refuse to execute the Algorithm. However, such cases will affect the computation results, and should not be considered.

(2) If TTP receives x, y and calculates $f(x, y)$, send $f_1(x, y)$ to P_1 and send $f_2(x, y)$ to P_2.

In the ideal model Algorithm, P_1 and P_2 do not obtain any information from each other except for obtaining $f_i(x, y)(i = 1, 2)$. The ideal Algorithm is the safest. If the Algorithm designed under the malicious model can also achieve the same security as the ideal Algorithm, the real algorithm can be considered as secure. In addition, the malicious model requires at least one of the parties to be honest, and there does not exist a algorithm that is secure even if all participants are malicious adversaries.

Under the ideal model, participants have auxiliary information z. The process of calculating strategy $\overline{B}$ jointly with $F(x, y)$ is denoted as $IDEAL_{F,\overline{B}(z)}(x, y)$. Choose a random number r, let $IDEAL_{F,\overline{B}(z)}(x, y) = \gamma(x, y, z, r)$, where $\gamma(x, y, z, r)$ is defined as follows:

(1) If P_1 is honest, there is

$$\gamma(x, y, z, r) = (f_1(x, y'), B_2(y, z, r, f_2(x, y'))), \tag{1}$$

among them, $y' = B_2(y, z, r)$.

(2) If P_2 is honest, there is

$$\gamma(x, y, z, r) = \begin{cases} (B_1(x, z, r, f_1(x', y), \perp), \perp), & \text{if} \quad B_1(x, z, r, f_1(x', y)) = \perp \\ (B_1(x, z, r, f_1(x', y)), f_2(x', y)), & \text{otherwise} \end{cases}. \tag{2}$$

In both Formulas (1) and (2), there is $x' = B_1(x, z, r)$.

Definition 1. *Security of MPC Algorithms under the malicious model.*

Let F: $\{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ be a probability polynomial time function. The output sequence messages are generated by $A_1(y, z)$ and interaction in the process $A_2(y, z)$ of $REAL_{\Pi,\bar{A}(z)}(x, y)$ executing algorithm with strategy $\overline{A}$ in the case of auxiliary input z; $\bar{A} = (A_1, A_2)$ denotes the probabilistic polynomial time algorithm constructed in the real model algorithm. The privacy information owned by both parties are x and y, respectively.

If for any acceptable strategy $\bar{A} = (A_1, A_2)$ in the real model, an acceptable strategy $\bar{B} = (B_1, B_2)$ in the ideal model can be found that satisfies

$$\left\{IDEAL_{F,\bar{B}(z)}(x, y)\right\}_{x,y,z} \overset{c}{\equiv} \left\{REAL_{\Pi,\bar{A}(z)}(x, y)\right\}_{x,y,z}, \tag{3}$$

then the algorithm Π securely computes the function F.

3. Secure Computation Algorithm for Graph Editing Distances under the Semi-Honest Model

Problem description: An undirected complete graph $G = (V, E)$ is formed by the vertex set $V = \{v_1, v_2, \ldots, v_m\}$ and the edge set $E = \left\{l_1, l_2, \ldots, l_{\frac{m(m-1)}{2}}\right\}$. Clearly, G has $\frac{m(m-1)}{2}$ edges. Alice has a subgraph G_A of G, and Bob has a subgraph G_B of G. Both parties want to query the editing distance between G_A and G_B confidentially without revealing their respective subgraphs.

The solution idea: Alice and Bob generate the one-dimensional arrays $A = (a_1, a_2, \ldots, a_k)$ and $B = (b_1, b_2, \ldots, b_k)$ corresponding to the subgraph G_A, G_B, respectively, according to the encoding method, and the problem of computing the editing distance of G_A, G_B is transformed into a secure computation of the number of different elements at the corresponding position of A, B. If $a_i \neq b_i$, it means that the edges or vertices represented by a_i and b_i exist only in the graph of one of the parties and need to be removed from this graph (or, alternatively, added to the graph of the other party).

3.1. Specific Algorithm

See Algorithm 1.

```
Algorithm 1 The MPC Algorithm of graphs editing distance under the semi-honest model
Input: Alice has the graph $G_A$ and Bob has the graph $G_B$.
Output: Graphs editing distance $GED(G_A, G_B)$.
Algorithm start:
    (1) Alice generates the Paillier cryptosystem public key $g$, private key $\lambda$ and sends $g$ to Bob.
    (2) Alice encodes $G_A$ into matrix $M_A$ according to the encoding method, takes out the elements of $M_A$ diagonally, arranges them in rows to get a one-dimensional array $A = (a_1, a_2, \ldots, a_k)$, then expands $A$ into $A^* = (a_1{}^*, a_2{}^*, \ldots, a_{2k}{}^*) = (a_1, a_2, \ldots, a_k, \overline{a}_1, \overline{a}_2, \ldots, \overline{a}_k)$ and encrypts $E(A^*)$, and sends $E(A^*)$ to Bob.
    (3) Bob encodes $G_B$ into matrix $M_B$ according to the encoding method, takes out the elements diagonal to $M_B$, arranges them in order by rows to obtain a one-dimensional array $B = (b_1, b_2, \ldots, b_k)$, and then expands $B$ into $B^* = (b_1{}^*, b_2{}^*, \ldots, b_{2k}{}^*) = (\overline{b}_1, \overline{b}_2, \ldots, \overline{b}_k, b_1, b_2, \ldots, b_k)$.
    (4) Bob selects the ciphertext $E(a_i^*)$ from $E(A^*)$ at the corresponding position according to the element $b_i^*$ in $B^*$ with value '1', calculates $C = \prod_{b_i^*} E(a_i^*)$, and then sends $C$ to Alice.
    (5) Alice decrypt $C$, which is the editing distance $GED(G_A, G_B)$ of the graph $G_A$ and $G_B$, and Alice outputs $GED(G_A, G_B)$.
End.
```

3.2. Correctness Analysis

(1) Alice expands A into $A^* = (a_1{}^*, a_2{}^*, \ldots, a_{2k}{}^*) = (a_1, a_2, \ldots, a_k, \overline{a}_1, \overline{a}_2, \ldots, \overline{a}_k)$, where $a_i = 0, \overline{a}_i = 1$, using the XOR operation; similarly, Bob generates $B^* = (b_1{}^*, b_2{}^*, \ldots, b_{2k}{}^*) = (\overline{b}_1, \overline{b}_2, \ldots, \overline{b}_k, b_1, b_2, \ldots, b_k)$.

(2) Bob computes $C = \sum_{b_i^*=1} E(a_i^*)$ and sends it to Alice, who decrypts it to obtain the number of elements with value '1' among the elements selected by Bob, i.e., the editing distance of G_A and G_B.

Algorithm 1 under the semi-honest model is secure because the participants are able to follow the rules to execute the algorithm. However, in real life, it is necessary to design MPC algorithms under the malicious model as there may be some malicious behaviors of the participants.

4. Secure Computation Algorithm for Graphs' Editing Distances under the Malicious Model

Designing an MPC algorithm under the malicious model usually involves designing countermeasures based on the malicious behaviors that the malicious participant in the semi-honest model algorithm might carry out, so that the malicious participant cannot carry them out or can be detected in the malicious model algorithm.

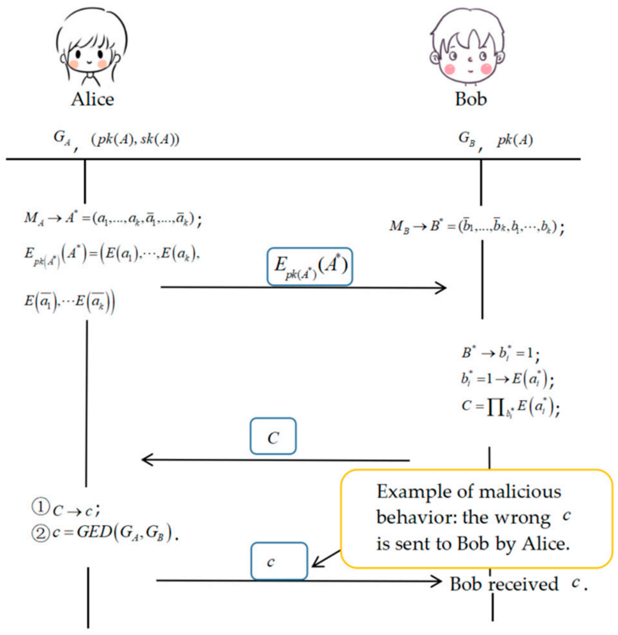
The following are possible malicious acts committed by a participant in Algorithm 1 (shown in Figure 4):

(1) In Algorithm 1, Alice has g and λ, while Bob only has g, and the final result is only decrypted unilaterally by Alice, which is unfair. It is also possible for Alice to tell Bob the wrong result. The countermeasure to solve this situation is that both parties can decrypt.

(2) In step (4) of Algorithm 1, Bob may provide a false ciphertext to Alice, and the solution is to use a hash function to avoid the situation.

(3) In step (5) of Algorithm 1, Alice told Bob the wrong result after decryption, causing him to draw the wrong conclusion. The solution is to request equal status and generate their respective public and private keys at the same time, and in the algorithm, Alice

and Bob decrypt the computation results separately to obtain the editing distance of the graphs, and finally both parties compute the correct result.



Alice
Bob
G_A, $(pk(A), sk(A))$
G_B, $pk(A)$
$M_A \to A^* = (a_1,\dots,a_k,\bar{a}_1,\dots,\bar{a}_k)$;
$E_{pk(A^*)}(A^*) = (E(a_1),\cdots,E(a_k), E(\bar{a}_1),\cdots E(\bar{a}_k))$
$E_{pk(A^*)}(A^*)$
$M_B \to B^* = (\bar{b}_1,\dots,\bar{b}_k,b_1,\cdots,b_k)$;
$B^* \to b_i^* = 1$;
$b_i^* = 1 \to E(a_i^*)$;
$C = \prod_{b_i^*} E(a_i^*)$;
C
①$C \to c$;
②$c = GED(G_A, G_B)$.
Example of malicious behavior: the wrong c is sent to Bob by Alice.
c
Bob received c.

Figure 4. Malicious behavior in Algorithm 1.

4.1. Specific Algorithm

To observe the encryption and decryption activities of the algorithm, we analyze the encoding and decoding of the algorithm. We refer to the public keys of the communicating parties as PK_A and PK_B, the private keys as sk_A and sk_B, and the roles of the recipients as Alice and Bob, respectively.

(1) $E_{pk}(A^*)$ $(PK_A$, Bob): Alice encodes G_A as M_A, takes the elements below the diagonal and arranges them in rows to obtain a one-dimensional array $A = (a_1, a_2, \dots, a_k)$, and then expands it to $A^* = (a_1{}^*, a_2{}^*, \dots, a_{2k}{}^*) = (a_1, a_2, \dots, a_k, \bar{a}_1, \bar{a}_2, \dots, \bar{a}_k)$ according to the XOR operation and sends it to Bob.

(2) $E_{pk}(B^*)$ $(PK_B$, Alice): Bob encodes G_B as M_B, takes the elements below the diagonal and arranges them in rows to obtain a one-dimensional array $B = (b_1, b_2, \dots, b_k)$, and then sends it to Alice as $B^* = (b_1{}^*, b_2{}^*, \dots, b_{2k}{}^*) = (\bar{b}_1, \bar{b}_2, \dots, \bar{b}_k, b_1, b_2, \dots, b_k)$ based on the XOR operation expansion.

(3) $D_{sk_A}(C_B)$ $(sk_A$, Bob): Alice decodes C_B to obtain C_b and sends C_b to Bob for verification.

(4) $D_{sk_B}(C_A)$ $(sk_B$, Alice): Bob decodes C_A to obtain C_a and sends C_a to Alice for verification.

4.2. Correctness Analysis

(1) In step (2), Alice and Bob use their public keys to encrypt A^*, B^* item by item to obtain $E_{pk_A}(A^*)$ and $E_{pk_B}(B^*)$, respectively, and then publish the ciphertext to the other party, which is secure because the other party does not have its own private key.

(2) In step (5), Alice and Bob decrypt C_b, C_a using their own λ_a and λ_b, respectively, and send them to each other.

(3) After each side receives the message from the other, Alice computes $G_A = C_a/r_a$ and sends it to Bob, Bob computes $G_B = C_b/r_b$ and sends it to Alice, and both parties obtain their respective ciphertexts, i.e., $\prod_{a_i*=1} E_{pk_B}(b_i*)$ and $\prod_{b_i*=1} E_{pk_A}(a_i*)$.

(4) In step (7), Alice verifies the equation $Hash(C_b/GED_B) \overset{?}{=} H_B$, and if it holds, outputs the editing distance $GED_B(G_A, G_B) = G_B$. In step (8), Bob verifies that equation $Hash(C_a/GED_A) \overset{?}{=} H_A$ holds, and if it does, outputs the editing distance

$GED_A(G_A, G_B) = G_A$. If $GED_A(G_A, G_B) = GED_B(G_A, G_B)$, the result is proven correct.

(5) No secure information is revealed throughout the process, and both parties are able to arrive at their respective results, avoiding the unfairness associated with one party telling the other the result.

4.3. Security Analysis

In the algorithm, both parties have exactly the same status and operations, and both parties have exactly the same security status, so only Alice's possible malicious behavior and its impact on the privacy of Bob's data and the correctness of the algorithm are analyzed.

(1) Alice fills her matrix expansion M_A with a one-dimensional array A^*. Alice makes an incorrect input for the elements in A^*. This is a case of Alice changing her own inputs, which is not considered because it cannot be avoided in an ideal algorithm.
(2) During the message passing in step (6), Bob leaks the result of C_B to Alice, but since Alice only has the public key but not the private key, Alice cannot obtain any information about C_B.
(3) In step (7), Alice has to prove that $G_A = C_a/r_a$ is correct using the hash function, which cannot be spoofed in this step, then after announcing G_A, Bob can compute $GED_A(G_A, G_B) = G_A$.
(4) Thus, all steps of the algorithm are secure, and we further prove that the algorithm is secure using the real/ideal model paradigm.

4.4. Security Proof

For MPC algorithms under the malicious model, the real/ideal model paradigm is widely accepted to be used to prove the security of the algorithms.

Theorem 2. *The graphs' editing distance MPC Algorithm (Algorithm 2) is secure under the malicious model.*

Proof. To prove that Algorithm 2 is secure, it is sufficient to show that the participants transform the acceptable policy pair $\bar{A} = (A_1, A_2)$ into the corresponding policy pair $\bar{B} = (B_1, B_2)$ in the ideal model algorithm during the execution of Algorithm 2, so that the output information of A_1 and A_2 is indistinguishable from that of B_1 and B_2 when Algorithm 2 is executed. Since the case of both parties being malicious participants is considered, it is assumed that one party is honest and the other dishonest. The discussion is divided into two cases (here, A_1, B_1 and A_2, B_2 represent Alice and Bob, respectively). □

Case 1. *A_1 is honest and A_2 is dishonest. A_1 executes Algorithm 2 honestly, then $REAL_{\bar{A}}(C_b, C_a) = \{F(C_b, A_2(C_a)), GED(G_A, G_B), H_A, S\}$, where the message sequence received by zero-knowledge proof A_2 is marked as S.*

A_1 is honest, in which case B_1 is determined, and B_1 executes the algorithm according to the algorithm steps. It is necessary to transform the real algorithm adversary A_2 into the ideal algorithm malicious adversary B_2. In other words, it is necessary to find an acceptable strategy for $\bar{B} = (B_1, B_2)$ in the ideal model, so that its output is indistinguishable from the calculation of $REAL_{\bar{A}(C_b, C_a)}$. (Moreover, B_2's decision depends on A_2's act).

Ideally, B_1 sends correct W to TTP (allowing TTP to send B_2 messages when B_1 receives the message). B_2 is dishonest, and its message to TTP depends on A_2's strategy. In summary, we know that B_2 sends $A_2(B)$ to TTP and TTP sends $F(C_b, A_2(C_a))$ to B_2 (B_1 will also obtain this result). B_2 has to use $F(C_b, A_2(C_a))$ to obtain an $view_{B_2}(C_b, A_2(C_a))$, which is indistinguishable from $view_{A_2}(C_b, A_2(C_a))$ obtained by A_2 in the real case and given to A_2 to obtain the output of A_2.

Algorithm 2 The MPC Algorithm of graphs editing distance under the malicious model

Input: Alice has graph G_A and Bob has graph G_B.
Output: Graphs editing distance $GED(G_A, G_B)$.
Preparation: Alice and Bob generate their own public keys (g_a, N_a), (g_b, N_b) and private keys λ_a, λ_b respectively. Exchange (g_a, N_a) and (g_b, N_b).
Algorithm start:

(1) Alice follows the encoding method and encodes G_A into a matrix M_A. She takes the elements of M_A diagonally and below and arranges them in order to obtain a one-dimensional array $A = (a_1, a_2, \ldots, a_k)$. She then expands A into
$A^* = (a_1{}^*, a_2{}^*, \ldots, a_{2k}{}^*) = (a_1, a_2, \ldots, a_k, \bar{a}_1, \bar{a}_2, \ldots, \bar{a}_k)$.
Bob operates on G_B in the same way to obtain the array
$B^* = (b_1{}^*, b_2{}^*, \ldots, b_{2k}{}^*) = (\bar{b}_1, \bar{b}_2, \ldots, \bar{b}_k, b_1, b_2, \ldots, b_k)$.

(2) Alice and Bob use their respective public keys to encrypt A^*, B^* item by item. Alice gets $E_{pk_A}(A^*) = [E_{pk_A}(a_1), E_{pk_A}(a_2), \ldots, E_{pk_A}(a_k), E_{pk_A}(\bar{a}_1), E_{pk_A}(\bar{a}_2), \ldots, E_{pk_A}(\bar{a}_k)]$ and Bob gets $E_{pk_B}(B^*) = [E_{pk_B}(\bar{b}_1), E_{pk_B}(\bar{b}_2), \ldots, E_{pk_B}(\bar{b}_k), E_{pk_B}(b_1), E_{pk_B}(b_2), \ldots, E_{pk_B}(b_k)]$. Alice and Bob publish $E_{pk_A}(A^*)$, $E_{pk_B}(B^*)$ to each other respectively.

(3) Alice selects the random number r_a, computes $H_A = Hash(r_a)$, and refers to the position of the element of $a_i{}^* = 1 (i \in [1, 2k])$ in A^*, selects the ciphertext $E_{pk_B}(b_i{}^*)$ in $E_{pk_B}(B^*)$ for the corresponding position to calculate $C_A = \left[\prod_{a_i*=1} E_{pk_B}(b_i*)\right] r_a$. Alice sends H_A and C_A to Bob.

(4) Bob selects the random number r_b, calculates $H_B = Hash(r_b)$, and refers to the position of the elements of $b_i{}^* = 1 (i \in [1, 2k])$ in B^*, selects the ciphertext $E_{pk_A}(a_i{}^*)$ in the corresponding position in $E_{pk_A}(A^*)$ to calculate $C_B = \left[\prod_{b_i*=1} E_{pk_A}(a_i*)\right] r_b$. Bob sends H_B and C_B to Alice.

(5) Alice decrypts C_B with her private key to get $C_b = D_{sk_A}(C_B)$. Bob decrypts C_A with his private key to get $C_a = D_{sk_B}(C_A)$. Alice and Bob send C_b and C_a to each other respectively

(6) Alice computes $G_A = C_a / r_a$ and sends it to Bob. Bob computes $G_B = C_b / r_b$ and sends it to Alice.

(7) Alice verifies $Hash(C_b / GED_B) \underset{=}{?} H_B$. If the equation holds, it means that Bob is not spoofing and Alice gets the editing distance $GED_B(G_A, G_B) = G_B$ between the two graphs and outputs it; otherwise the algorithm is terminated.

(8) Bob verifies $Hash(C_a / GED_A) \underset{=}{?} H_A$. If the equation holds, it shows that Alice did not deceive and Bob gets the editing distance $GED_A(G_A, G_B) = G_A$ between the two graphs and outputs it; otherwise terminate the algorithm.

(9) If $GED_A(G_A, G_B) = GED_B(G_A, G_B)$, prove that the result is correct; otherwise show that the result is wrong and not accepted.
End.

B_2 selecting C'_b satisfies $F(C'_b, A_2(C_a)) = F(C_b, A_2(C_a))$, i.e., Algorithm 2 is executed with C'_b assumed to be the input of A_1 with A_2. The corresponding sequence of messages S' is available during the execution of the algorithm B_2. The completion of the algorithm execution yields

$$IDEAL_{\bar{B}}(C_b, C_a) = \left\{F(C_b, A_2(C_a)), GED(G_A, G_B), H_A, S'\right\}. \tag{4}$$

Due to the ideal algorithm using the same encryption as the real algorithm, it is guaranteed that $C_b \overset{c}{\equiv} C'_b$. The hash function proves that k is guaranteed again, so $Hash(C_b / GED_B) \underset{=}{?} H_B$.

Case 2. *A_2 is honest, A_1 is dishonest. Two scenarios exist:*

(a) *A_1 does not disclose the result or ignores TTP (considered as A_1 aborting the algorithm), TTP sends $\perp$ to A_2. There is*

$$REAL_{\bar{A}}(C_b, C_a) = \left\{A_1(C'_b, C'_a), GED(G_A, G_B)S, \perp\right\}. \tag{5}$$

(b) *Conversely, the TTP sends $F(A_1(C_b), C_a)$ to A_2, there is*

$$REAL_{\bar{A}}(C_b, C_a) = \{A_1(C_b', C_a'), GED(G_A, G_B), S, F(A_1(C_b), C_a)\}, \tag{6}$$

In the process of zero-knowledge proof, the message sequence received by A_1 is marked as S.

A_2 honestly transforms the adversary A_1 under the real model into the ideal adversary B_1. That is, to prove that A_1 is indistinguishable from B_1, so find a set of strategy pairs $\bar{B} = (B_1, B_2)$ under the ideal model such that their output satisfies computational indistinguishability with $REAL_{\bar{A}(Q_1,Q_2)}$.

A_1 is dishonest, and B_1's strategy for treating the TTP depends on A_1's behavior, so the message it would send to TTP is $A_1(C_b)$, obtaining $F(A_1(C_b), C_a)$ from TTP. Ideally, B_1 utilizes $F(A_1(C_b), C_a)$ to manage to obtain a $view_{B_1}(A_1(C_b), C_a)$ that satisfies the computational indistinguishability of $view_{A_1}(A_1(C_b), C_a)$ obtained with A_1 in the real algorithm, and gives it to A_1 to obtain an A_1 output. Let B_1 execute Algorithm 2 with A_1 using B' satisfying $F(A_1(C_b), C_a') = F(A_1(C_b), C_a)$ as an input value.

During the execution of the algorithm, the corresponding message sequence S' is available to B_1 and corresponds to the existence of the following two cases above:

(a) Under the ideal model, when B_1 informs TTP not to send the result to B_2, it is obtained that

$$IDEAL_{\bar{B}}(C_b, C_a) = \{A_1(C_b', C_a'), GED(G_A, G_B), S', \perp\}. \tag{7}$$

(b) Conversely, it is

$$IDEAL_{\bar{B}}(C_b, C_a) = \{A_1(C_b', C_a'), GED(G_A, G_B), S', F(A_1(C_b), C_a)\}. \tag{8}$$

The outputs of A_2 and B_2 in the real and ideal algorithms in these two cases are the same, and the ideal and real algorithms use the same encryption algorithm, so $C_a \overset{c}{\equiv} C_a'$, and the zero-knowledge proof guarantees that $Hash(C_a/GED_A) \underline{?} H_A$, then

$$\left\{IDEAL_{\bar{B}}(C_b, C_a)\right\} \overset{c}{\equiv} \left\{REAL_{\bar{A}}(C_b, C_a)\right\}. \tag{9}$$

In summary, under the malicious model, Algorithm 2 is secure.

5. Performance Analysis

Through the analysis of computational complexity and communication complexity, Algorithms 1 and 2 are compared with the existing schemes, and the performance of the Algorithm is illustrated.

5.1. Computational Complexity

Reference [15] uses a depth-first search code as a canonical tagging system, and on the basis of this code, a new graph similarity measure algorithm is proposed in combination with Levenshtein distance (i.e., string editing distance), which has a complexity of n^3 modulo exponential operation. Reference [16] designed secure graph isomorphism and similarity determination algorithms based on hash functions, which measure computational complexity by comparing the number of hash operations, and it has a complexity of $6h$ (h is the number of operations to perform one hash operation).

The computational complexity of Algorithm 1 in this paper mainly consists of Alice encrypting an array of length $2k$, which requires $4k$ modulo exponential operations; Bob picks the element with the value of '1' at the corresponding position from Alice's ciphertext according to his own array and computes it, which requires 1 modulo exponential operation; and finally, Alice decrypts it once, which requires 2 modulo exponential operations.

Therefore, Algorithm 1 requires a total of $4k + 3$ modulo exponential operations. The computational complexity of Algorithm 2 consists mainly of Alice and Bob encrypting an array of length $2k$, which requires $8k$ modulo exponential operations, and using the hash function to execute the MPC algorithm, which has the computation complexity of $4h$. Therefore, the complexity of Algorithm 2 is $8k + 4h$.

5.2. Communication Complexity

Communication complexity is usually measured in terms of communication rounds. Reference [15] requires six rounds of interaction to complete the computation, Reference [16] requires four rounds, Algorithm 1 in this paper requires one round of communication, and Algorithm 2 requires three rounds of communication to complete the computation. Table 1 shows a comparison of the performance.

Table 1. Performance comparison.

Algorithm	Computational Complexity	Communication Complexity	Anti-Malicious Adversaries
Reference [15]	n^3 mode index	6	×
Reference [16]	$6h$	5	×
Algorithm 1	$4k + 3$ mode index	1	×
Algorithm 2	$8k$ mode index $+ 4h$	3	√

n is the number of vertices in the graph, h is the number of hash operations performed, k is the number of elements in the array.

As shown in Table 1, with a small difference in the number of communication rounds, algorithms 1 and 2 improve efficiency by choosing simple and fast encryption algorithms, and Algorithm 2 can resist malicious participants' attacks and has wider applications.

5.3. Experimental Simulation

In order to have a visual comparison of the computational complexity of each algorithm, the algorithms in References [15,16] and Algorithm 2 in this paper are experimentally simulated. The experimental environment is processor Intel(R) Core(TM) i5-8300 H @ 2.30 GHz, 12 GB of RAM, Windows 10 (64-bit) operating system, in PyCharm 2020.3.2 using the Python language. algorithms based on hash functions and using public keys do not take up a lot of memory. This is because they process one element at a time and are easily pipelined. It is also possible to run such algorithms on standard PCs on collections of millions. The algorithm for confidentially computing graph edit distances in this paper takes up about 2G of memory while running on a PyCharm system, so it is entirely possible to implement the algorithm on a standard PC.

The algorithms of both References [15,16] use a homomorphic encryption algorithm when calculating the graphs' editing distance, so the time used by References [15,16] and Algorithm 2 is tested through simulated experiments to compare the efficiency by comparing the time taken for the algorithm to execute. In the experiment, all other things being equal, the graph G_A is transformed into G_B by the add and delete operations, for example, by first randomly generating two undirected graphs with vertex number m. For the vertex m, the graphs are taken in order 5, 10, . . ., 40, and for each m, 1000 simulations are tested and the average of the algorithm execution times (ignore preprocessing time in the algorithm) is counted. Figure 4 shows a comparison of the execution times of References [15,16] and Algorithm 2 as the number of vertices increases, where the vertical coordinates indicate the execution time in milliseconds and the horizontal coordinates indicate the number of different vertices. From Figure 5, it can be seen that Algorithm 2 takes less time than the other algorithms as the number of vertices increases.

In this experiment, all other things being equal, the number of edges l in the graph takes 4, 6, . . , 18 for each l and is tested in 1000 simulation experiments; the average of the algorithm execution time is counted (ignore preprocessing time in the algorithm). Figure 5 shows a comparison of the execution times of the algorithms in References [15,16] and Algorithm 2 of this paper as the number of edges increases, where the vertical coordinates indicate the time taken (milliseconds) and the horizontal coordinates indicate the different number of edges. As shown in Figure 6, Algorithm 2 takes less time than the other algorithms as the number of edges of the graph increases.

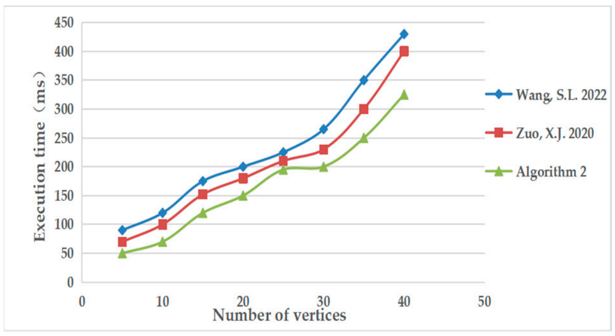


Figure 5. Execution time pattern with increasing number of vertices. (Reference [15]: Wang, S.L. 2022, Reference [16]: Zuo, X.J. 2020).

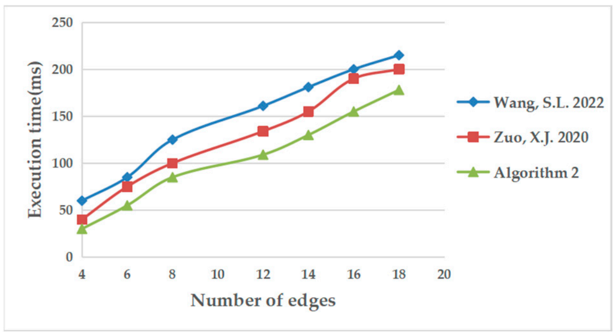


Figure 6. Execution time pattern with increasing number of edges. (Reference [15]: Wang, S.L. 2022, Reference [16]: Zuo, X.J. 2020).

To more fully evaluate the performance of Algorithm 2, communication interaction experiments are conducted based on the previous experiments. The number of vertices and edges of the simulated graph is increased to determine the possible delay time in the execution of Algorithm 2. The delay time affects the efficiency of the algorithm execution, but it is not considered in the previous experimental evaluation. The relationship between delay time and the number of graph vertices is shown in Figure 7, and the relationship with the number of graph edges is shown in Figure 8.

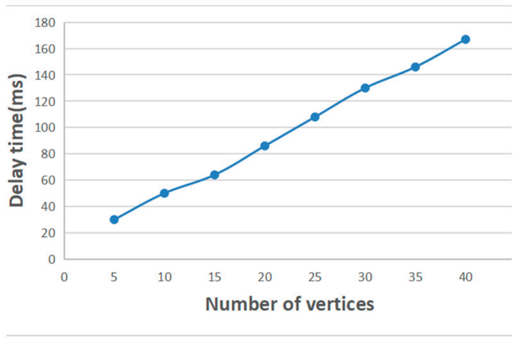


Figure 7. Delay time versus number of vertices.

In order to evaluate the performance of Algorithm 2 more comprehensively, we conducted experiments on processing scales based on the previous experiments. The performance of Algorithm 2 is evaluated more explicitly by looking at the number of scales comparing the literature and the number of processing graphs of Algorithm 2, while time remains constant. This is shown in Figure 9.

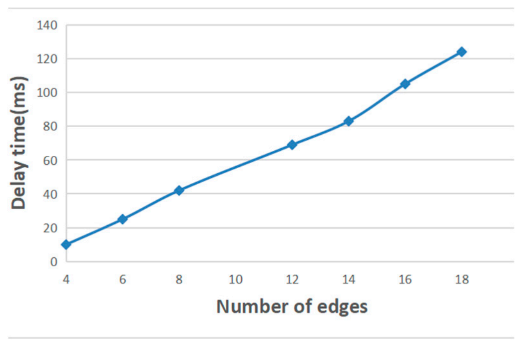


Figure 8. Delay time versus number of edges.

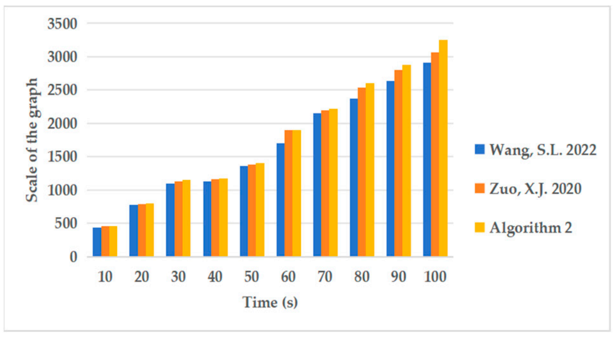


Figure 9. Scale of the graph versus time. (Reference [15]: Wang, S.L. 2022, Reference [16]: Zuo, X.J. 2020).

We also conducted experiments on detecting attacks with an increased number of graph sizes to detect how long it takes for the algorithm to detect the operation of a malicious adversary in the presence of spoofing. The experiments show that the algorithm is effective in detecting attacks when one party has a malicious attack. The experimental results are shown in Figure 10.

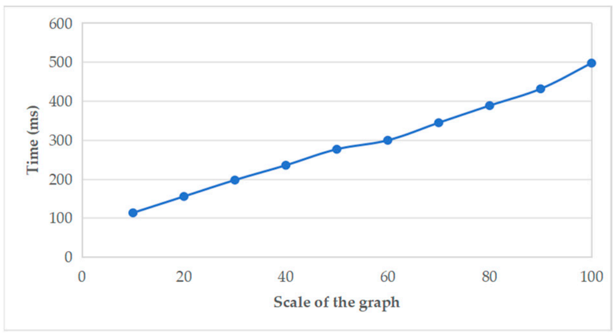


Figure 10. Detection of attack time.

5.4. Applications

(1) In bioinformatics, gene structure needs to be represented as a graph, and similarity between two genes can be measured using the graph editing distance. In practice, most genetic data are more private and solving such detection and query problems without compromising privacy requires secure computation of graph editing distances. For example, in association similarity studies concerning disease, crime, drugs, social aspects, etc., the data involved are highly private, and if the DNA of a suspect is highly similar to the DNA structure of the evidence information left by the criminal at

the crime scene, then the suspect may be directly related to the criminal, and thus the similarity of the DNA graph structure needs to be calculated confidentially, and the problem can be abstracted as graph editing distance secure computation, which can be solved using the method of this paper's algorithm.

(2) In artificial intelligence, computer vision is a simulation of biological vision using computers and related equipment. Its main task is to obtain the 3D information about the corresponding scene by processing the collected pictures or videos. Using computer vision, a search function can be realized with a picture, and similar or identical pictures can be found quickly. For example, in a game with a terrain map, computer vision can find similarities between the virtual game and reality.

6. Summary Outlook

This paper solves the problem of secure computation of graph edit distance. Firstly, a new encoding method is proposed, which can transform the correspondence between vertices and edges in a graph into a matrix, and the Paillier encryption algorithm is applied to design an MPC algorithm under the semi-honest model. A algorithm for secure computation of graph editing distances under the malicious model is designed using the hash function for the attack behaviors that may be implemented by malicious participants in the semi-honest algorithm. The security of the algorithm is demonstrated using the real/ideal model paradigm and reduces the computational complexity and communication complexity compared to existing algorithms, which can be applied to computer vision to find out similar or identical photos using computer vision implementation of graph search function. It can also be applied to confidentially calculate the similarity of DNA graph structures, which is important in relationships related to disease, crime, socialization, and so on. The application scenarios are wider.

In addition to the above advantages, the algorithm has some limitations, which can be addressed in future research work. The proposed method is applicable to two parties for information interaction; therefore, as a future work, the proposed method can be extended to multiple parties for graph edit distance computation, so that it can have wider applications in areas such as machine learning. Meanwhile, the graph theory problem is one of the important problems in secure computational geometry, and there are many meaningful graph theory problems that deserve further research in the next work, such as confidential determination of graph isomorphism, confidential computation of the shortest paths in graphs, and so on.

Author Contributions: Conceptualization, X.L. (Xin Liu) and J.K.; methodology, J.K.; investigation, X.L. (Xin Liu); writing—original draft preparation, X.L. (Xin Liu) and L.P.; software, D.L. and G.X.; funding acquisition, X.L. (Xin Liu); validation, X.L. (Xiaomeng Liu) and X.C.; writing—original draft, J.K.; writing—review and editing, X.L. (Xin Liu), X.L. (Xiaomeng Liu) and X.C. All authors have read and agreed to the published version of the manuscript.

Funding: National Natural Science Foundation of China (92046001, 61962009), Inner Mongolia Natural Science Foundation (2021MS06006), 2023 Inner Mongolia Young Science and Technology Talents Support Project (NJYT23106), 2022 Basic Scientific Research Project of Direct Universities of Inner Mongolia (2022-101), 2022 Fund Project of Central Government Guiding Local Science and Tech-nology Development (2022ZY0024), 2022 Chinese Academy of Sciences "Western Light" Talent Training Program "Western Young Scholars" Project (22040601), Open Foundation of State key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications) (SKLNST-2023-1-08), 2023 Inner Mongolia Archives Technology Project (2023016), the 14th Five Year Plan of Education and Science of Inner Mongolia (NGJGH2021167), Inner Mongolia Science and Technology Major Project (2019ZD025), 2022 Inner Mongolia Postgraduate Education and Teaching Reform Project (JGSZ2022037), Inner Mongolia Postgraduate Scientific Research Innovation Project(S20231164Z), Research and Application Project of Big Data Privacy Security Computing System (2023).

Data Availability Statement: The authors approve that data used to support the findings of this study are included in the article.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

N	$N = pq$, Both p and q are large primes
m	Plaintext
c	Ciphertext
(g_a, N_a)	The public key of Alice's Paillier encryption system
(g_b, N_b)	The public key of Bob's Paillier encryption system
λ_a	The private key of Alice's Paillier encryption system
λ_b	The private key of Bob's Paillier encryption system
$E(\ )$	The process of converting encrypted plaintext into ciphertext
$D(\ )$	The process of decrypting ciphertext into plaintext
r_i	Random numbers
$E_{pk(A)}$	Encrypted calculation with $A's$ public key
$E_{pk(B)}$	Encrypted calculation with $B's$ public key
$IDEAL_{\bar{B}}(C_b, C_a)$	The function calculation results of C_b and C_a in the ideal case
$REAL_{\bar{A}}(C_b, C_a)$	The function calculation results of C_b and C_a in the practical case
$F(\ )$	Function calculation results

References

1. Zhao, C.; Zhao, S.N.; Zhao, M.H.; Chen, Z.X.; Gao, C.Z.; Li, H.W.; Tan, Y.A. Secure multi-party computation: Theory, practice and applications. *Inf. Sci.* **2019**, *476*, 357–372. [CrossRef]
2. Knott, B.; Venkataraman, S.; Hannun, A.; Sengupta, S.; Ibrahim, M. Crypten: Secure multi-party computation meets machine learning. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 4961–4973.
3. Volgushev, N.; Schwarzkopf, M.; Getchell, B.; Varia, M.; Lapets, A.; Bestavros, A. Conclave: Secure multi-party computation on big data. In Proceedings of the Fourteenth EuroSys Conference, Dresden Germany, 25–28 March 2019.
4. Feng, Q.; He, D.B.; Zeadally, S.; Khan, M.K.; Kumar, N. A survey on privacy protection in blockchain system. *J. Netw. Comput. Appl.* **2019**, *126*, 45–58. [CrossRef]
5. Pang, H.P.; Wang, B.C. Privacy-preserving association rule mining using homomorphic encryption in a multikey environment. *IEEE Syst. J.* **2020**, *15*, 3131–3141. [CrossRef]
6. Dong, C.Y.; Loukide, G. Approximating private set union/intersection cardinality with logarithmic complexity. *IEEE Trans. Inf. Forensics Secur.* **2017**, *12*, 2792–2806. [CrossRef]
7. Goldreich, O. Secure multi-party computation. Manuscript. *Prelim. Version* **1998**, *78*, 110.
8. Prathik, A.; Uma, K.; Anuradha, J. An Overview of application of Graph theory. *Int. J. ChemTech Res.* **2016**, *9*, 242–248.
9. Dou, J.W.; Liu, X.H.; Zhou, S.F.; Li, S.D. Efficient pooled secure multi-party computing protocols and applications. *Chin. J. Comput.* **2018**, *41*, 1844–1860.
10. Wei, Q.; Li, S.D.; Wang, W.L.; Du, R.M. Safe multiparty computation of graph intersections and mergers. *J. Cryptologic Res.* **2020**, *7*, 774–788.
11. He, J.; Erfani, S.; Ma, X.; Bailey, J.; Chi, Y.; Hua, X.S. α-IoU: A Family of Power Intersection over Union Losses for Bounding Box Regression. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 20230–20242.
12. Zhao, X.L.; Jia, Z.L.; Li, S.D. Safe computation of set intersection problems. *J. Cryptologic Res.* **2022**, *9*, 294–307.
13. Tang, C.M.; Lin, X.H. Privacy Protection Set Intersection Computing protocol. *Netinfo Secur.* **2020**, *20*, 9–15.
14. Gao, A.; Liang, Y.; Xie, X.J.; Wang, Z.S.; Li, J.T. Social network information dissemination methods that support privacy protection. *J. Front. Comput. Sci. Technol.* **2021**, *15*, 233–248.
15. Wang, S.L.; Zheng, Y.F.; Jia, X.H.; Wang, C. OblivGM: Oblivious Attributed Subgraph Matching as a Cloud Service. *IEEE Trans. Inf. Forensics Secur.* **2022**, *17*, 3582–3596. [CrossRef]
16. Zuo, X.J.; Li, L.X.; Peng, H.P.; Luo, S.S.; Yang, Y.X. Privacy-preserving subgraph matching scheme with authentication in social networks. *IEEE Trans. Cloud Comput.* **2020**, *10*, 2038–2049. [CrossRef]
17. Sharmila, G.; Devi, M.K. BTLA-LSDG: Blockchain-Based Triune Layered Architecture for Authenticated Subgraph Query Search in Large-Scale Dynamic Graphs. *IETE J. Res.* **2023**, 1–24. [CrossRef]
18. Xu, C.; Chen, Q.; Hu, H.B.; Hei, X.J. Authenticating aggregate queries over set-valued data with confidentiality. *IEEE Trans. Knowl. Data Eng.* **2017**, *30*, 630–644. [CrossRef]
19. Bringmann, K.; Gawrychowski, P.; Mozes, S.; Weimann, O. Tree edit distance cannot be computed in strongly subcubic time (unless APSP can). *TALG* **2020**, *16*, 1–22. [CrossRef]
20. Garcia-Hernandez, C.; Fernandez, A.; Serratosa, F. Ligand-based virtual screening using graph edit distance as molecular similarity measure. *J. Chem. Inf. Model.* **2019**, *59*, 1410–1421. [CrossRef]

21. Li, S.D.; Yang, X.L.; Zuo, X.J.; Zhou, S.F.; Kang, j.; Liu, X. Graphical similarity determination for the protection of private information. *Acta Electron. Sin.* **2017**, *45*, 2184–2189.
22. Blumenthal, D.B.; Gamper, J. On the exact computation of the graph edit distance. *Pattern Recogn. Lett.* **2020**, *134*, 46–57. [CrossRef]
23. Yuan, Y.; Lian, X.; Wang, G.R.; Ma, Y.L.; Wang, Y.S. Constrained shortest path query in a large time-dependent graph. *Proc. Vldb Endow.* **2019**, *12*, 1058–1070. [CrossRef]
24. Dey, R.; Balabantaray, R.C.; Mohanty, S.H. Sliding window based off-line handwritten text recognition using edit distance. *Multimed. Tools Appl.* **2022**, *81*, 22761–22788. [CrossRef]
25. Ma, J.C.; Zheng, H.B.; Zhao, J.H.; Chen, X.; Zhai, J.Q.; Zhang, C.H. An islanding detection and prevention method based on path query of distribution network topology graph. *IEEE Trans. Sustain. Energy* **2021**, *13*, 81–90. [CrossRef]
26. Ghosh, E.; Kamara, S.; Tamassia, R. Efficient graph encryption scheme for shortest path queries. In Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security, Hong Kong, China, 7–11 June 2021.
27. Zhang, M.W.; Chen, Y.; Susilo, W. PPO-CPQ: A privacy-preserving optimization of clinical pathway query for e-healthcare systems. *IEEE Internet Things* **2020**, *7*, 10660–10672. [CrossRef]
28. Zhou, J.; Qin, X.; Ding, Y.; Ma, H. Spatial–Temporal Dynamic Graph Differential Equation Network for Traffic Flow Forecasting. *Mathematics* **2023**, *11*, 2867. [CrossRef]
29. Fang, W.T.; Mohsen, Z.; Chen, Z.Y. Secure and privacy preserving consensus for second-order systems based on paillier encryption. *Syst. Control Lett.* **2021**, *148*, 104869. [CrossRef]
30. Sobti, R.; Geetha, G. Cryptographic hash functions: A review. *IJCSI* **2012**, *9*, 461.
31. Kociumaka, T.; Pissis, S.P.; Radoszewski, J. Pattern matching and consensus problems on weighted sequences and profiles. *Theor. Comput. Syst.* **2019**, *63*, 506–542. [CrossRef]