Hindawi
Wireless Communications and Mobile Computing
Volume 2020, Article ID 8859213, 13 pages
https://doi.org/10.1155/2020/8859213

Research Article

Privacy-Preserving Graph Operations for Mobile Authentication

Peng Li,[1,2] **Fucai Zhou**,[1] **Zifeng Xu**,[1] **Yuxi Li**,[1] **and Jian Xu**[1]

[1]*Software College, Northeastern University, Shenyang, China*
[2]*School of Information Engineering, Eastern Liaoning University, China*

Correspondence should be addressed to Fucai Zhou; fczhou@mail.neu.edu.cn

Received 7 July 2020; Revised 13 October 2020; Accepted 11 November 2020; Published 23 November 2020

Academic Editor: Ding Wang

Along with the fast development of wireless technologies, smart devices have become an integral part of our daily life. Authentication is one of the most common and effective methods for these smart devices to prevent unauthorized access. Moreover, smart devices tend to have limited computing power, and they may possess sensitive data. In this paper, we investigate performing graph operations in a privacy-preserving manner, which can be used for anonymous authentication for smart devices. We propose two protocols that allow two parties to jointly compute the intersection and union of their private graphs. Our protocols utilize homomorphic encryption to prevent information leakage during the process, and we provide security proofs of the protocols in the semihonest setting. At last, we implement and evaluate the efficiency of our protocols through experiments on real-world graph data.

1. Introduction

With the rapid development of IoT technology, we are surrounded by various types of smart devices in our daily life, such as sensors, wearable devices, and smart vehicles [1]. Authentication is one of the most important mechanisms to provide security protection for these smart devices [2], and authentication for light-weighted devices has become a hot research topic in the past years [3, 4].

In recent years, researchers have proposed several mobile authentication schemes based on graph data structure and graph algorithms [5–7]. Graph data and graph processing are well studied for the last decades [8, 9], since they can help to solve many practical problems in different application areas, such as web data processing [10], data mining [11], social networking [12], biological networking [13], and communication networking [14].

1.1. Motivation. In this paper, we consider the problem of computing graph operations between two parties while preventing information leakage, which has great potential in smart device authentication. For example, when the mobile devices communicate with cloud servers, they need to first jointly perform identity authentication for security protection. Since the mobile devices may contain sensitive information of the users and the cloud servers cannot be fully trusted in general, the privacy leakage problem for mobile authentication has become a security threat [15]. In order to protect the privacies of the mobile devices, the devices can model their identities and properties as graph-structured data, and the cloud servers can model their authentication policies as graph-structured data as well. After that, the identity authentication process can be converted into performing graph operations in a privacy-preserving manner.

1.2. Our Contributions. We study the problem of performing graph intersection and union while protecting the privacies of the input graphs. Suppose that for two parties, Alice and Bob, each has a private graph, denoted as G_A and G_B, respectively. Alice wishes to learn the intersection and union of these two graphs. In other words, Alice wishes to learn $G_I = G_A \bigcap G_B$ and $G_U = G_A \bigcup G_B$. In addition, both Alice and Bob do not wish to reveal any information about their graphs to the other party. The contributions of this paper can be summarized as below:

(i) We present two graph operation protocols between two parties, a server and a client. The first protocol

allows the server and the client to jointly compute the intersection of their input graphs, and the second protocol computes the union of the input graphs. Our constructions first use the Paillier cryptosystem and oblivious polynomial evaluation to compute the intersection and the union of the vertices. After that, we use the homomorphic property of the Paillier cryptosystem to compute the edge intersection and union

(ii) We provide the security models of the protocols, and we prove that the protocols are secure in the semi-honest setting. Furthermore, we analyze the information leakage and propose methods to minimize the leakages

(iii) We discuss the efficiencies of the protocols in terms of computation costs and communication costs. At last, we implement our constructions and perform experiments on real-world graph data

An earlier version of this paper was presented at the 22nd Australasian Conference on Information Security and Privacy, 2017 [16]. The previous work presented a private graph intersection protocol with rough analysis. This paper extends the previous work by presenting a private graph union protocol with detailed analysis and experimental results.

2. Related Work

There are many different approaches to construct authentication schemes for smart devices. Among them, graph-based authentication schemes are widely used in IoT [5, 17, 18]. In 2002, Micali and Rivest [19] first introduced the transitive signature based on graph theory, which provides an unforgeable signature for undirected graphs. After that, various graph-based signature and authentication schemes were proposed [5–7]. In 2017, Chuang et al. [5] proposed an authentication system in Internet of Things based on multigraph zero-knowledge. The system provides suitable security protection for IoT authentication services. The proposed multigraph zero-knowledge procedure is faster than traditional zero-knowledge methods and ECC-based solutions. The experiment results indicate that the system is lightweighted and highly adaptive. Lin et al. [6] proposed a transitively graph authentication scheme for blockchain-based identity management systems in 2018. The system is used to bind a digital identity object to its real-world entity, therefore achieving identity authentication. The system is constructed based on transitively closed undirected graphs and vertex signatures. According to the evaluation results, the system is efficient, even when the graph dynamically adds or deletes vertices and edges. In 2019, Shao et al. [7] proposed a multifactor authentication scheme using a fuzzy graph domination model. The scheme is adaptive choosing one or multiple privacy-preserving identities to authenticate the users. The authors designed a weighted vertex-edge dominating set to solve the weighted domination problem on fuzzy graphs. Compared to existing solutions, the scheme is more efficient for solving instances with moderate orders.

In this work, we consider the problem of performing graph intersection and union in the privacy-preserving manner and proposed two secure multiparty computation protocols. Secure Multiparty computation (MPC) has been extensively studied over the past decades. Generally speaking, MPC allows multiple participants to jointly perform certain computations without losing the privacy of their input data, even when some players cheat during the process. MPC was first formally introduced by Yao in 1982 [20] and extended by Goldreich et al. [21]. Their works convert certain computation problems into a combinatorial circuit, then the parties perform computations over the gates in the circuit. After that, a large number of MPC protocols have been proposed to solve various problems, such as privacy-preserving set operations [22] and private information retrieval [23].

3. Preliminary

In this section, we present the preliminaries related to our proposed protocols. First, we present the relevant notations that we used in this paper in Table 1.

3.1. Additive Homomorphic Encryption. Homomorphic encryption schemes allow the users to perform certain computation operations on the ciphertext space, such as addition and multiplication. In our private graph operation protocols, we utilize an additive homomorphic encryption scheme called the Paillier cryptosystem, proposed by Paillier in 1999 [24]. The Paillier cryptosystem contains three algorithms, described as follows:

$(pk, sk) \leftarrow \mathrm{KeyGen}(1^k)$ is the key generation algorithm. The input is a security parameter k. The outputs are a public key pk and a secret key sk. The public key contains a large number N which specifies the message space, the ciphertext space, and the random space to be $\mathbb{Z}_N$, $\mathbb{Z}_{N^2}^*$, and $\mathbb{Z}_N^*$, respectively.

$t^{\oplus} \leftarrow \mathrm{Enc}(pk, t; r)$ is the encryption algorithm. The input is the public key pk, a plaintext $t \in \mathbb{Z}_N$, and a random number $r \in \mathbb{Z}_N^*$. The output is the ciphertext $t^{\oplus} \in \mathbb{Z}_{N^2}^*$. For simplicity, we use the notion $t^{\oplus} = \mathrm{Enc}(t)$.

$t \leftarrow \mathrm{Dec}(sk, t^{\oplus})$ is the decryption algorithm. The input is the secret key sk and a ciphertext $t^{\oplus} \in \mathbb{Z}_{N^2}^*$. The output is the plaintext $t \in \mathbb{Z}_N$. For simplicity, we use the notion $t = \mathrm{Dec}(t^{\oplus})$.

The Paillier cryptosystem has the following properties:

3.1.1. Correctness. For any key pairs $(pk, sk) \leftarrow \mathrm{KeyGen}(1^k)$ and any plaintext $t \in \mathbb{Z}_N$, $\mathrm{Dec}(\mathrm{Enc}(t)) = t$ always holds.

3.1.2. IND-CPA Security. Two ciphertexts $t_0^{\oplus}$ and $t_1^{\oplus}$ are indistinguishable for probabilistic polynomial-time adversaries that only have access to the public parameters.

3.1.3. Homomorphic Property. For any two plaintexts $t_0, t_1 \in \mathbb{Z}_N$, there exists an operation $\oplus$ in the ciphertext space, such that $\mathrm{Dec}(\mathrm{Enc}(t_0) \oplus \mathrm{Enc}(t_1)) = t_0 + t_1$. Furthermore, there exists another operation $\otimes$ in the ciphertext space, such that $\mathrm{Dec}(\mathrm{Enc}(t_0) \otimes t_1) = t_0 \cdot t_1$.

Table 1: Table of notations.

Symbol	Description
PGI	Private graph intersection protocol
PGU	Private graph union protocol
S, C	The server, the client
G_S, G_C	The server's graph, the client's graph
G_I	The intersection of $G_S$and G_C
G_U	The union of G_S and G_C
V_S, V_C, V_I, V_U	The vertices of G_S, G_C, G_I, and G_U
E_S, E_C, E_I, E_U	The edges of G_S, G_C, G_I, and G_U
m, n, p, q	The number of vertices in G_S, G_C, G_I, and G_U

3.2. Private Set Intersection. Private Set Intersection (PSI) is a cryptographic protocol that allows two parties, each holding a private set, to jointly compute the intersection of their sets without leaking any additional information. The first secure two-party private set intersection protocol is introduced by Freedman, Nissim, and Pinkas (FNP) in 2004 [25]. The protocol utilizes homomorphic encryption and oblivious polynomial evaluation to ensure each party learns no information about the other party's private input during the computation. Later, several other protocols have been proposed with different features and security levels [26–28].

3.3. Graph Representation. In our protocol, we represent a graph as $G = (V, E)$, where V is the vertex collection and E is the edge collection. We represent the vertex collection as a sorted set with ascending order, $V = \{v_1, v_2, \cdots, v_z\}$, where z is the number of vertices in G, $v_i \in \mathbb{Z}$, and $v_i < v_{i+1}$ for $1 \leq i \leq z - 1$. We represent the edge collection as an adjacency matrix,

$$E = \begin{pmatrix} e_{1,1} & \cdots & e_{1,z} \\ \vdots & \ddots & \vdots \\ e_{z,1} & \cdots & e_{z,z} \end{pmatrix}, \quad (1)$$

where $e_{i,j}$ is the adjacency relation between the vertices v_i and v_j, and $e_{i,j} \in \{0, 1\}$. If vertices v_i and v_j are adjacent, i.e., there is at least one edge that connects them, $e_{i,j} = 1$; otherwise, $e_{i,j} = 0$. Note that E is a square matrix with z rows and z columns. For an undirected graph, E is a symmetric matrix, since the edges are two-way.

For example, we represent the directed graph illustrated in Figure 1 as $G = (V, E)$, where $V = \{1, 5, 23, 50, 74\}$ and

$$E = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}. \quad (2)$$

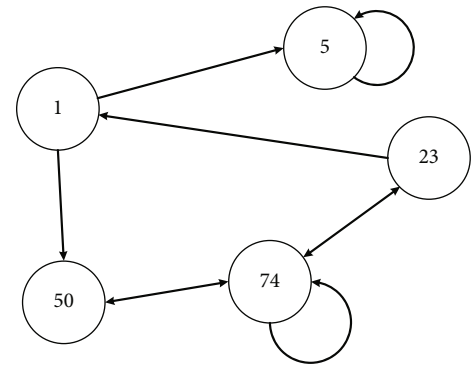


Figure 1: Example graph.

4. Definitions and Security Models

4.1. Protocol Definitions. We formally describe the private graph intersection (PGI) protocol and the private graph union (PGU) protocol. The protocols involve two participants, a server and a client, denoted as S and C, respectively. Each of the participants holds a private graph, which is intended to be kept secret from the other participant.

We denote the graphs of the server and client as $G_S = (V_S, E_S)$ and $G_C = (V_C, E_C)$, respectively, where V and E are the sets of vertices and edges of the graphs. The intersection of G_S and G_C is defined as $G_I = G_S \bigcap G_C = (V_I, E_I)$, where $V_I = V_S \bigcap V_C$ and $E_I = E_S \bigcap E_C$. The union of G_S and G_C is defined as $G_U = G_S \bigcup G_C = (V_U, E_U)$, where $V_U = V_S \bigcup V_C$ and $E_U = E_S \bigcup E_C$.

PGI and PGU allow the participants to jointly compute G_I and G_U, respectively, in a privacy-preserving manner. At the end of the protocols, only the server learns the result. The formal definitions of PGI and PGU are described as follows:

Definition 1 (private graph intersection protocol). If both participants are honest, for any $G_S = (V_S, E_S)$ and any $G_C = (V_C, E_C)$, the private graph intersection protocol computes $G_I = G_S \bigcap G_C$. At the end of the protocol, only S learns G_I.

Definition 2 (private graph union protocol). If both participants are honest, for any $G_S = (V_S, E_S)$ and any $G_C = (V_C, E_C)$, the private graph union protocol computes $G_U = G_S \bigcup G_C$. At the end of the protocol, only S learns G_U.

4.2. Security Models. When considering privacy protecting in authentication, the term privacy may have different definitions and properties, such as user identity and untraceability [29, 30]. In this work, the privacies of the server and the client refer to any information about their graphs. Therefore, any information about the vertices and edges of the graphs is considered as private, such as the number of vertices, the number of edges, the values of the vertices, and whether two vertices are connected by an edge.

The security goals of both PGI and PGU protocols are protecting the privacies of both the server and the client during the computation. In other words, both the server and the

client should learn no information about the graph of the other party.

We use the semihonest security model for both PGI and PGU, which means both the server and the client perform the protocols faithfully, but they may try to learn any information about the graph of the other participant. The security models are adopted from the work of [31–33].

While achieving no information leakage is the ideal goal, our protocols leak partial information during the process. The information leakages for PGI are defined as leakage functions $\mathscr{L}_1$ and $\mathscr{L}_2$, and the information leakages for PGU are defined as $\mathscr{L}_3$ and $\mathscr{L}_4$. The detailed information about the leakage functions are as follows: $\mathscr{L}_1$ is the number of vertices in G_C, $\mathscr{L}_2$ is the vertex intersection V_I and the number of vertices in G_S, $\mathscr{L}_3$ is the number of vertices in G_C and the number of common vertices between G_S and G_C, and $\mathscr{L}_4$ is the vertex union V_U and the number of vertices in G_S.

The formal definitions of security models are described as follows:

Definition 3 (PGI security). A semihonest server learns nothing about the client's graph, beyond what can be deduced from G_I and the leakage function $\mathscr{L}_1$, and a semihonest client learns nothing about the server's graph, beyond the leakage function $\mathscr{L}_2$.

Definition 4 (PGU security). A semihonest server learns nothing about the client's graph, beyond what can be deduced from G_U and the leakage function $\mathscr{L}_3$, and a semihonest client learns nothing about the server's graph, beyond the leakage function $\mathscr{L}_4$.

5. Protocol Construction

In this section, we propose the constructions of PGI and PGU. The graphs of the server and the client are represented as $G_S = (V_S, E_S)$ and $G_C = (V_C, E_C)$, respectively, where $V_S = \{s_1, s_2, \cdots, s_m\}$, $V_C = \{c_1, c_2, \cdots, c_n\}$,

$$E_S = \begin{pmatrix} s_{1,1} & \cdots & s_{1,m} \\ \vdots & \ddots & \vdots \\ s_{m,1} & \cdots & s_{m,m} \end{pmatrix}, \quad E_C = \begin{pmatrix} c_{1,1} & \cdots & c_{1,n} \\ \vdots & \ddots & \vdots \\ c_{n,1} & \cdots & c_{n,n} \end{pmatrix}. \tag{3}$$

5.1. PGI Construction. We use the FNP protocol [25] as a building block for computing the vertex intersection. The private graph intersection protocol is described below:

Input: S and C hold the graphs $G_S = (V_S, E_S)$ and $G_C = (V_C, E_C)$, respectively.

Output: S learns $G_I = (V_I, E_I)$.

Protocol:

Step 1. S runs the key generation algorithm of the Paillier cryptosystem, $(pk, sk) \leftarrow \mathrm{KeyGen}(1^k)$, and obtains the public key and the secret key. Then, S publishes pk.

Step 2.

(a) S constructs a polynomial $P(x) = (x - s_1)(x - s_2) \cdots (x - s_m) = \sum_{u=0}^{m} \alpha_u x^u$, such that all the roots of $P(x)$ are exactly the elements in V_S. In other words, $P(x) = 0$ if and only if $x \in V_S$

(b) S encrypts each α_i, for $0 \le i \le m$, under the Paillier cryptosystem, and sends the set of ciphertexts $\alpha^{\oplus} = \{\alpha_i^{\oplus}\}_{0 \le i \le m}$ to C

Step 3.

(a) By using the homomorphic properties of the Paillier cryptosystem, C evaluates the polynomial P using each element in V_C as input. In other words, C computes $\mathrm{Enc}(P(c_i))$, for $1 \le i \le n$

(b) For each polynomial evaluation, C chooses a random value r and computes $\beta_i^{\oplus} = \mathrm{Enc}(rP(c_i) + c_i)$. Then, C sends $\beta^{\oplus} = \{\beta_i^{\oplus}\}_{1 \le i \le n}$ to S

Step 4. S decrypts all the ciphertexts received and compares the decrypted values with his vertex set V_S. If a decrypted value $\beta_i = \mathrm{Dec}(\beta_i^{\oplus})$ has a corresponding element in V_S, it is an element of the intersection of V_S and V_C. In other words, if $\beta_i \in V_S$, $\beta_i \in V_I$. After decrypting all the received ciphertexts, the server obtains V_I.

Step 5.

(a) S uses V_I to construct an adjacency matrix A of size $p \times p$, where p is the number of the vertex in V_I:

$$A = \begin{pmatrix} a_{1,1} & \cdots & a_{1,p} \\ \vdots & \ddots & \vdots \\ a_{p,1} & \cdots & a_{p,p} \end{pmatrix}. \tag{4}$$

A has the property that, for each vertex pair $v_x \in V_I$ and $v_y \in V_I$, if an edge exists in G_S between vertices v_x and v_y, $a_{x,y} = 1$; otherwise, $a_{x,y} = 0$.

(b) S encrypts each element in A under the Paillier cryptosystem and obtains an encrypted matrix $A^{\oplus} = \mathrm{Enc}(A)$

(c) S sends $A^{\oplus}$ and V_I to C

Step 6.

(a) By using V_I, C constructs an adjacency matrix B using the same method in the last step:

$$B = \begin{pmatrix} b_{1,1} & \cdots & b_{1,p} \\ \vdots & \ddots & \vdots \\ b_{p,1} & \cdots & b_{p,p} \end{pmatrix} \tag{5}$$

(b) C computes

$$E_I^{\oplus} = A^{\oplus} \otimes B = \begin{pmatrix} a_{1,1}^{\oplus} & \cdots & a_{1,p}^{\oplus} \\ \vdots & \ddots & \vdots \\ a_{p,1}^{\oplus} & \cdots & a_{p,p}^{\oplus} \end{pmatrix} \otimes \begin{pmatrix} b_{1,1} & \cdots & b_{1,p} \\ \vdots & \ddots & \vdots \\ b_{p,1} & \cdots & b_{p,p} \end{pmatrix} = \begin{pmatrix} a_{1,1}^{\oplus} \otimes b_{1,1} & \cdots & a_{1,p}^{\oplus} \otimes b_{1,p} \\ \vdots & \ddots & \vdots \\ a_{p,1}^{\oplus} \otimes b_{p,1} & \cdots & a_{p,p}^{\oplus} \otimes b_{p,p} \end{pmatrix} \tag{6}$$

(c) C sends $E_I^{\oplus}$ to S

Step 7. S decrypts each element in $E_I^{\oplus}$ and obtains $E_I = \mathrm{Dec}(E_I^{\oplus})$. At last, S obtains $G_I = (V_I, E_I)$.

5.2. PGU Construction. The private graph union protocol is described below:

Input: S and C hold the graphs $G_S = (V_S, E_S)$ and $G_C = (V_C, E_C)$, respectively.

Output: S learns $G_U = (V_U, E_U)$.

Protocol:

Step 1. Same as Step 1 of PGI.

Step 2. Same as Step 2 of PGI.

Step 3.

(a) By using the homomorphic properties of the Paillier cryptosystem, C evaluates the polynomial P using each element in V_C as input. In other words, C computes $\mathrm{Enc}(P(c_i))$, for $1 \le i \le n$

(b) For each polynomial evaluation, C choose a random value r and computes $\beta_i^{\oplus} = \mathrm{Enc}(P(c_i)) \otimes r$. Then, C sends the set of all resulting ciphertexts $\beta^{\oplus} = \{\beta_i^{\oplus}\}_{1 \le i \le n}$ to S

Step 4. S decrypts each ciphertext received as $\beta_i = \mathrm{Dec}(\beta_i^{\oplus})$ and checks the decrypted value. If $\beta_i = 0$, S computes $\gamma_i^{\oplus} = \mathrm{Enc}(0)$; otherwise, S computes $\gamma_i^{\oplus} = \mathrm{Enc}(1)$. Then, S sends $\gamma^{\oplus} = \{\gamma_i^{\oplus}\}_{1 \le i \le n}$ to C.

Step 5. After receiving $\gamma^{\oplus}$, C computes $\delta_i^{\oplus} = c_i \otimes \gamma_i^{\oplus}$, for $1 \le i \le n$. Then, C sends $\delta^{\oplus} = \{\delta_i^{\oplus}\}_{1 \le i \le n}$ to S.

Step 6.

(a) S decryptes each value in $\delta^{\oplus}$ and checks if the decrypted value $\delta_i = \mathrm{Dec}(\delta_i^{\oplus})$ is zero

(b) By combining the server's vertex set V_S and the set of nonzero decrypted values $\{\delta_i\}_{\delta_i \ne 0}$, S obtains V_U. V_U is then sorted in ascending order and is represented as $V_U = \{u_1, u_2, \cdots, u_q\}$

Step 7.

(a) S uses V_U to construct an adjacency matrix A of size $q \times q$, where q is the number of vertex in V_U:

$$A = \begin{pmatrix} a_{1,1} & \cdots & a_{1,q} \\ \vdots & \ddots & \vdots \\ a_{q,1} & \cdots & a_{q,q} \end{pmatrix}. \tag{7}$$

A has the property that, for each vertex pair $u_x \in V_U$ and $u_y \in V_U$, if an edge exists in G_S between vertices u_x and u_y, $a_{x,y} = 1$; otherwise, $a_{x,y} = 0$

(b) S encrypts each element in A under the Paillier cryptosystem and sends the encrypted matrix $A^{\oplus}$ and V_U to C

Step 8.

(a) C uses V_U to construct an adjacency matrix B in the same manner as S in the last step:

$$B = \begin{pmatrix} b_{1,1} & \cdots & b_{1,q} \\ \vdots & \ddots & \vdots \\ b_{q,1} & \cdots & b_{q,q} \end{pmatrix} \tag{8}$$

(b) C encrypts each element in B using the Paillier cryptosystem and obtains $B^{\oplus}$

(c) C generates a matrix R with $q \times q$ random values:

$$R = \begin{pmatrix} r_{1,1} & \cdots & r_{1,\mathrm{q}} \\ \vdots & \ddots & \vdots \\ r_{q,1} & \cdots & r_{q,q} \end{pmatrix} \tag{9}$$

(d) C computes:

$$E_U^{\oplus} = \left(A^{\oplus} \oplus B^{\oplus}\right) \otimes R = \begin{pmatrix} \left(a_{1,1}^{\oplus} \oplus b_{1,1}^{\oplus}\right) \otimes r_{1,1} & \cdots & \left(a_{1,q}^{\oplus} \oplus b_{1,q}^{\oplus}\right) \otimes r_{1,q} \\ \vdots & \ddots & \vdots \\ \left(a_{q,1}^{\oplus} \oplus b_{q,1}^{\oplus}\right) \otimes r_{q,1} & \cdots & \left(a_{q,q}^{\oplus} \oplus b_{q,q}^{\oplus}\right) \otimes r_{q,q} \end{pmatrix} = \begin{pmatrix} e_{1,1}^{\oplus} & \cdots & e_{1,q}^{\oplus} \\ \vdots & \ddots & \vdots \\ e_{q,1}^{\oplus} & \cdots & e_{q,q}^{\oplus} \end{pmatrix} \tag{10}$$

(e) C cends $E_U^{\oplus}$ to S

Step 9. S decrypts the matrix $E_U^{\oplus}$. For each decrypted element $e_{i,j}$, if $e_{i,j} \neq 0$, set $e_{i,j} = 1$. At last, S obtains E_U.

6. Analysis

6.1. Security Analysis. In this section, we prove the correctness and security of both PGI and PGU. When analyzing the security of the proposed protocols, we assume both the server and the client evaluate the protocols faithfully, but they may try to obtain as much information about the graph of the other party as possible. The security analysis for the protocols is divided into two cases, where one of the server and the client acts as the adversary in each case. Then, we prove the zero-knowledge properties of the server and the client in each case, using the methods and techniques introduced in [15, 34].

Lemma 5 (PGI correctness). *If both participants are honest, for any $G_S = (V_S, E_S)$ and any $G_C = (V_C, E_C)$, the private graph intersection protocol computes $G_I = (V_I, E_I) = G_S \bigcap G_C$.*

Proof. The correctness of PGI is ensured by the correctness of the FNP protocol and the homomorphic property of the Paillier cryptosystem.

During Steps 2 to 4 of the protocol, the client and the server jointly perform a FNP protocol using their vertex collections as inputs. At the end of Step 4, the server learns the vertex intersection V_I, and the client receives V_I from the server in Step 5.

In Steps 5 and 6, the server and the client construct two adjacency matrices by using V_I, denoted as A and B, respectively. Note that A and B contain the adjacency relations between the vertices in V_I for graphs G_S and G_C, respectively. In other words, if an edge exists between two vertices in V_I, it leads to a value of 1 in the corresponding position of the constructed adjacency matrix; otherwise, it leads to a value of 0 instead. Therefore, the dot product of A and B will produce an adjacency matrix that represents the edge intersection. If an edge exists in both A and B, i.e., it is a common edge between G_S and G_C, the dot product of its adjacency relations will result a value of 1. If an edge only exists in one of G_S and G_C, or the edge does not exist at all, the dot product will result in a value of 0.

In Step 6, the client receives the encryption of A under the Paillier cryptosystem from the server. If the Paillier cryptosystem has the homomorphic property, i.e., it supports multiplication between a ciphertext and a constant, the client can homomorphically compute the dot product of the $A^{\oplus}$ and B, and the result is the encryption of the edge intersection. Finally, in Step 7, the server obtains the edge intersection after decryption.

As a result, if the FNP protocol is correct and the Paillier cryptosystem has the homomorphic property, the private graph intersection protocol computes $G_I = (V_I, E_I) = G_S \bigcap G_C$.

Lemma 6 (PGI server zero-knowledge). *A semihonest server learns nothing about the client's graph, beyond what can be deduced from G_I and the leakage function $\mathscr{L}_1$.*

Proof. The proof of PGI server zero-knowledge is trivial. During PGI, there are two parts where the server receives information about the client's graph. The first part is during the FNP protocol in Step 3, and the second part is at the end of Step 6.

For the first part, in Step 3, the server receives a set of ciphertexts from the client. The server can learn the number of vertices in the client's graph by counting the number of ciphertexts, which is the predefined leakage function $\mathscr{L}_1$. By decrypting the ciphertexts, the server obtains a set of values. If a value exists in V_S, it is a common vertex between G_S and G_C, which is a part of the final result of the protocol. Otherwise, if the value does not exist in V_S, it will be a random value, which has no relation to the client's graph.

For the second part, the server receives $E_I^{\oplus}$ from the client, which is the ciphertext of the edge intersection. Upon decryption, the server only learns the edge intersection. As a result, the PGI server zero-knowledge holds.

Lemma 7 (PGI client zero-knowledge). *A semihonest client learns nothing about the server's graph, beyond the leakage function $\mathscr{L}_2$.*

Proof. There are two parts where the client receives information about the server's graph. The first part is during the FNP protocol in Step 2, and the second part is at the end of Step 5.

For the first part, the client receives a set of encrypted coefficients $\alpha^{\oplus}$ of the polynomial P from the server. The client can learn the number of vertices of the server's graph by counting the number of encrypted coefficients received, which is a part of the predefined leakage function $\mathscr{L}_2$.

For the second part, the client receives an encrypted matrix $A^{\oplus}$ and the vertex intersection V_I. Since V_I is also a part of the predefined leakage function $\mathscr{L}_2$, we need to show that $A^{\oplus}$ does not reveal any information about the server's graph. According to the protocol construction, $A^{\oplus}$ contains the encryptions of adjacency relations between the vertices in V_I for the server's graph. Therefore, if the client cannot distinguish between the cases where the server has different input graphs, given the knowledge of $A^{\oplus}$ and V_I, the PGI client zero-knowledge holds. Consider the following experiment:

$$\begin{aligned}
&\mathrm{EXP}_{\mathscr{A}}\left(1^k\right),\\
&(G_0, G_1) \leftarrow \mathscr{A},\\
&b \leftarrow {}^{\$}\{0, 1\},\\
&(pk, sk) \leftarrow \mathrm{Step}\,1\left(1^k\right),\\
&\alpha^{\oplus} \leftarrow \mathrm{Step}\,2(G_b, pk),
\end{aligned}$$

$$\beta^{\oplus} \leftarrow \text{Step 3}\left(\alpha^{\oplus}, G_C\right),$$

$$V_I \leftarrow \text{Step 4}\left(\beta^{\oplus}, sk\right),$$

$$A^{\oplus} \leftarrow \text{Step 5}(G_b, V_I, pk),$$

$$\widehat{b} \leftarrow \mathscr{A}\left(\alpha^{\oplus}, V_I, A^{\oplus}\right) \text{if } \widehat{b} = b, \text{ output 1},$$

$$\text{otherwise, output 0}. \tag{11}$$

In the above experiment, $\mathscr{A}$ is a probabilistic polynomial-time adversarial client with a private graph $G_C = (E_C, V_C)$. The adversary first chooses two graphs, denoted as $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$, respectively. The two graphs have the property that $V_0 \bigcap V_C = V_1 \bigcap V_C$ and $|V_0| = |V_1|$. $\mathscr{A}$ then sends the graphs to the server. The server randomly picks a bit $b = \{0, 1\}$, and chooses G_b as the private graph. After that, the server and $\mathscr{A}$ jointly perform the private graph intersection protocol from Steps 1 to 5.

At the end of Step 5, $\mathscr{A}$ needs to output a bit $\widehat{b}$, using the information he recevied during the protocol. If $\widehat{b} = b$, the experiment outputs 1; otherwise, it outputs 0. The advantage of the above experiment for $\mathscr{A}$ is defined as $\text{Adv}_{\mathscr{A}} = |\Pr[\text{EXP}_{\mathscr{A}}(1^k) = 1] - 1/2|$.

During PGI, the information that $\mathscr{A}$ receives contains $\alpha^{\oplus}$, V_I, and $A^{\oplus}$. $\alpha^{\oplus}$ contains a set of ciphertexts under the Paillier cryptosystem, V_I is the vertex intersection, and $A^{\oplus}$ is an encrypted adjacency matrix under the Paillier cryptosystem.

Due to the condition $V_0 \bigcap V_C = V_1 \bigcap V_C$, the vertex intersection V_I gives no useful information since V_I will be the same for both G_0 and G_1. Since the Paillier cryptosystem is IND-CPA secure and $\mathscr{A}$ cannot decrypt the ciphertexts without the private key, $\alpha^{\oplus}$ and $A^{\oplus}$ cannot help $\mathscr{A}$ to distinguish which graph the server has chosen. As a result, if the Paillier cryptosystem is IND-CPA secure, the advantage of the above experiment for $\mathscr{A}$ is negligible, i.e., $\text{Adv}_{\mathscr{A}} = |\Pr[\text{EXP}_{\mathscr{A}}(1^k) = 1] - 1/2| = \varepsilon$, where ε is negligible.

At last, we construct a simulator Sim_S to simulate the view of the client in the ideal model. Sim_S is given the knowledge of the vertex intersection V_I and the vertex number m of the server's graph. In the above experiment, Sim_S sends a set of $m + 1$ random values to the client in Step 2 and sends V_I and a matrix with $p \times p$ random values to the client in Step 5. Since the client cannot distinguish between the ciphertexts under the Paillier cryptosystem and random values, the view of the client in the ideal model is computationally indistinguishable from the view in the real model, i.e., $\text{View}_C^{\text{real}}[S(G_S), C] \approx \text{View}_C^{\text{ideal}}[\text{Sim}_S(V_I, m), C]$. As a result, the PGI client zero-knowledge holds.

Lemma 8 (PGU correctness). *If both participants are honest, for any $G_S = (V_S, E_S)$ and any $G_C = (V_C, E_C)$, the private graph union protocol computes $G_U = (V_U, E_U) = G_S \bigcup G_C$.*

Proof. The correctness of PGU is ensured by the homomorphic property of the Paillier cryptosystem. Steps 2– of PGU compute the vertex union, and Steps 7–9 compute the edge union.

In order to compute the vertex union between G_S and G_C, the server needs to obtain the vertices in G_C that are not in G_S.

In Step 2, the server constructs a polynomial, such that all the roots are exactly the vertices in G_S. After that, the client homomorphically evaluates the polynomial using all the vertices in G_C, and each polynomial evaluation is homomorphically multiplied by a random value. Therefore, the common vertices between G_S and G_C will result in encryptions of zero, and other vertices will result in encryptions of random values. In Step 4, the server decrypts all the polynomial evaluations. If the decryption is zero, the server generates an encryption of 0; otherwise, the server generates an encryption of 1. In the next step, the client homomorphically multiplies the received encryptions with the vertices in V_C. For an encryption of 0, i.e., the vertex is a common vertex, the client will result in an encryption of 0; for an encryption of 1, i.e., the vertex is not a common vertex, the client will result in an encryption of the vertex. As a result, in Step 6, the server learns the set of vertices that only exists in G_C. By combing the above set and V_S, the server obtains the vertex union V_U.

In order to compute the edge union, the server needs to obtain an adjacency matrix, such that if an edge does not exist in either G_S and G_C, it will have a corresponding value of 0 in the matrix; otherwise, it will have a corresponding value of 1.

In Steps 7 and 8, each of the server and the client constructs an adjacency matrix using the vertex union and his own graph and encrypts each element under the Paillier cryptosystem. The client then homomorphically adds the encrypted values at the same locations in the two matrices. There are three circumstances for the addition results. If an edge does not exist in either of the graphs, the addition will result in an encryption of 0; if an edge only exists in one of the graphs, the addition will result in an encryption of 1; if an edge exists in both of the graphs, the addition will result in an encryption of 2. Then, the client homomorphically multiplies each result by a random value. Therefore, for the edges that do not exist in either of the graphs, the final result will still be an encryption of 0; for the edges that only exist in one of the graphs and the edges that exist in both of the graphs, the final result will be encryptions of random values. Finally, in Step 9, the server decrypts the encrypted matrix and replaces all the nonzero values to 1, which is the edge union of G_S and G_C.

As a result, if the Paillier cryptosystem has the homomorphic property, the private graph union protocol computes $G_U = (V_U, E_U) = G_S \bigcup G_C$.

Lemma 9 (PGU server zero-knowledge). *A semihonest server learns nothing about the client's graph, beyond what can be deduced from G_U and the leakage function $\mathscr{L}_3$.*

Proof. There are three parts where the server receives information from the client, which are Steps 3, 5, and 8.

In Step 3, the server receives a set of ciphertexts, $\beta^{\oplus}$, from the client. Each vertex in V_C has a corresponding ciphertext in $\beta^{\oplus}$. If a vertex in V_C also exists in V_S, i.e., it is a common vertex in both graphs, it will result in an encryption of 0; otherwise, it will result in an encryption of a random value. By counting the number of ciphertexts in $\beta^{\oplus}$, the server can learn the number of vertices in the client's graph, and by decrypting and counting the number of 0 s, the server can learn the number of common vertices. The above information is defined as leakage function $\mathscr{L}_3$.

In Step 5, the server receives another set of ciphertexts, $\gamma^{\oplus}$, from the client. Similar as above, each vertex in V_C has a corresponding ciphertext in $\gamma^{\oplus}$. If a vertex exists in both V_S and V_C, it will result in an encryption of 0; otherwise, it will result in an encryption of the vertex itself. Therefore, upon decryption, the server learns of the vertices in V_C that do not exist in V_S, which are a part of the vertex union.

In Step 8, the server receives an encrypted matrix, $E_U^{\oplus}$, from the client. Each element of the matrix represents the adjacency relation between two vertices in the graph union. If an edge exists in at least one of the input graphs, the corresponding adjacency value will be a random number; if an edge does not exist in either of the input graphs, it will result in an adjacency value of 0. By decrypting the matrix and replacing the random values to 1, the server obtains the edge union. As a result, the PGU server zero-knowledge holds.

Lemma 10 (PGU client zero-knowledge). *A semihonest client learns nothing about the server's graph, beyond what can be deduced from V_U and the leakage function $\mathscr{L}_4$.*

Proof. There are three parts where the client receives information from the server, which are Steps 2, 4, and 7. In Step 2, the client receives a set $\alpha^{\oplus}$ that contains $m+1$ ciphertexts under the Paillier cryptosystem, which are encryptions of the coefficients of the server's polynomial. The client can learn the vertex number of the server's graph by counting the ciphertexts in $\alpha^{\oplus}$, which is the leakage function $\mathscr{L}_4$. In Step 4, the client receives another set of ciphertexts $\gamma^{\oplus}$, which contains n encryptions of 1 s and 0 s. In Step 7, the client receives an encrypted matrix of size $q \times q$, which contains encryptions of 1 s and 0 s. In order to prove that the above information does not reveal anything about the server's graph beyond what can be deduced from V_U and the leakage function $\mathscr{L}_4$, consider the following experiment:

$$\begin{gathered}
\mathrm{EXP}_{\mathscr{A}}\left(1^k\right),\\
(G_0, G_1) \leftarrow \mathscr{A},\\
b \leftarrow {}^{\$}\{0,1\},\\
(pk, sk) \leftarrow \text{Step } 1\left(1^k\right),\\
\alpha^{\oplus} \leftarrow \text{Step } 2(G_b, pk),\\
\beta^{\oplus} \leftarrow \text{Step } 3\left(\alpha^{\oplus}, G_C\right),\\
\gamma^{\oplus} \leftarrow \text{Step } 4\left(\beta^{\oplus}, pk, sk\right),\\
\delta^{\oplus} \leftarrow \text{Step } 5\left(\gamma^{\oplus}, G_C\right),\\
V_U \leftarrow \text{Step } 6\left(\delta^{\oplus}, sk, G_b\right),\\
A^{\oplus} \leftarrow \text{Step } 7(G_b, V_U, pk),\\
\widehat{b} \leftarrow \mathscr{A}\left(\alpha^{\oplus}, \gamma^{\oplus}, A^{\oplus}, V_U\right), \text{if } \widehat{b} = b, \text{ output } 1,\\
\text{otherwise, output } 0.
\end{gathered} \tag{12}$$

In the above experiment, $\mathscr{A}$ is a probabilistic polynomial-time adversarial client with a private graph $G_C = (E_C, V_C)$. The adversary first chooses two graphs, denoted as $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$, respectively. The two graphs have the property that $V_0 \bigcup V_C = V_1 \bigcup V_C$ and $|V_0| = |V_1|$. $\mathscr{A}$ then sends the graphs to the server. The server randomly picks a bit $b = \{0, 1\}$ and chooses G_b as the private graph. After that, the server and $\mathscr{A}$ jointly perform the private graph union protocol from Steps 1 to 7.

At the end of Step 7, $\mathscr{A}$ needs to output a bit $\widehat{b}$, using the information he received during the protocol. If $\widehat{b} = b$, the experiment outputs 1; otherwise, it outputs 0. The advantage of the above experiment for $\mathscr{A}$ is defined as $\mathrm{Adv}_{\mathscr{A}} = |\Pr[\mathrm{EXP}_{\mathscr{A}}(1^k) = 1] - 1/2|$.

During PGU, the information that $\mathscr{A}$ receives contains $\alpha^{\oplus}, \gamma^{\oplus}, A^{\oplus}$, and V_U. $\alpha^{\oplus}$ and $\gamma^{\oplus}$ are both sets of ciphertexts under the Paillier cryptosystem. Since G_0 and G_1 satisfied the condition $|V_0| = |V_1|$, the numbers of ciphertexts in $\alpha^{\oplus}$ will be the same for both G_0 and G_1. $A^{\oplus}$ is a matrix filled with $q \times q$ ciphertexts. Since the Paillier cryptosystem is IND-CPA secure and $\mathscr{A}$ cannot decrypt the ciphertexts without the private key, $\alpha^{\oplus}$, $\gamma^{\oplus}$, and $A^{\oplus}$ cannot help $\mathscr{A}$ to distinguish which graph the server has chosen. Furthermore, since G_0 and G_1 satisfied the condition $V_0 \bigcup V_C = V_1 \bigcup V_C$, V_U will be the same for both G_0 and G_1. As a result, if the Paillier cryptosystem is IND-CPA secure, the advantage of the above experiment for $\mathscr{A}$ is negligible, i.e., $\mathrm{Adv}_{\mathscr{A}} = |\Pr[\mathrm{EXP}_{\mathscr{A}}(1^k) = 1] - 1/2| = \varepsilon$, where ε is negligible.

At last, we construct a simulator Sim_S to simulate the view of the client in the ideal model. Sim_S is given the knowledge of the vertex union V_U and the vertex number m of the server's graph. In the ideal model, Sim_S genereates a set of $m+1$ random values in Step 2, a set of n random values in Step 4, and a matrix of size $q \times q$ filled with random values in Step 7. Since the Paillier cryptosystem is IND-CPA secure, the client cannot distinguish the ciphertexts and random values. Therefore, the view of the client in the ideal model is computationally indistinguishable from the view in the real model, i.e., $\mathrm{View}_C^{\mathrm{real}}[S(G_S), C] \approx \mathrm{View}_C^{\mathrm{ideal}}[\mathrm{Sim}_S(V_U, m), C]$. As a result, the PGU client zero-knowledge holds.

6.2. Efficiency Analysis. In this section, we analyze the efficiencies of PGI and PGU in terms of communication cost and computation cost. The communication cost is measured in terms of the amount of ciphertexts that has been transferred between the server and the client, and the computation

cost is measured in terms of modular exponentiations and multiplications.

We denote m as the number of vertices in G_S, n as the number of vertices in G_C, p as the number of vertices in the intersection of G_S and G_C, and q as the number of vertices in the union of G_S and G_C.

6.2.1. PGI Communication Cost. The construction of PGI is simple and only requires $O(1)$ rounds of communication. In Step 2, the server sends $m+1$ ciphertexts to the client. In Step 3, the client sends n ciphertexts to the server. In Step 5, the server sends p^2 ciphertexts to the client. At last, in Step 6, the client sends p^2 ciphertexts to the server. As a result, the total communication cost of our protocol is $O(m+n+p^2)$ ciphertexts.

6.2.2. PGI Server Computation Cost. In Step 2, constructing the polynomial requires $O(m^2)$ modular multiplication, and encrypting the coefficients requires $O(m)$ modular exponentiations. In Step 4, decrypting the received ciphertexts requires $O(n)$ modular exponentiations. In Step 5, encrypting each element in A requires $O(p^2)$ modular exponentiations. In Step 7, decrypting each element in $E_I^{\oplus}$ requires $O(p^2)$ exponentiations. As a result, the total computation cost for the server is $O(m+n+p^2)$ modular exponentiations and $O(m^2)$ modular multiplications.

6.2.3. PGI Client Computation Cost. In Step 3, obliviously evaluating the polynomial requires $O(mn)$ modular exponentiations. In Step 6, computing $E_I^{\oplus}$ requires $O(p^2)$ modular exponentiations. As a result, the total computation cost for the client is $O(mn+p^2)$ modular exponentiations.

6.2.4. PGU Communication Cost. The construction of PGU also only requires $O(1)$ rounds of communication. During Steps 2–5, the server sends $m+1+n$ ciphertexts to the client, and the client sends $2n$ ciphertexts to the server. During Steps 7 and 8, the server sends q^2 ciphertexts to the client, and the client sends q^2 ciphertexts to the server. As a result, the total communication cost is $O(m+n+q^2)$ ciphertexts.

6.2.5. PGU Server Computation Cost. In Step 2, constructing the polynomial requires $O(m^2)$ modular multiplication, and encrypting the coefficients requires $O(m)$ modular exponentiations. In Step 4, decrypting n ciphertexts requires $O(n)$ modular exponentiations, and encrypting n ciphertexts requires $O(n)$ modular exponentiations. In Step 6, decrypting n ciphertexts requires $O(n)$ modular exponentiations. In Step 7, encrypting each element in A requires $O(q^2)$ modular exponentiations. In Step 9, decrypting each element in $E_U^{\oplus}$ requires $O(q^2)$ modular exponentiations. As a result, the total computation cost for the server is $O(m+n+q^2)$ modular exponentiations and $O(m^2)$ modular multiplications.

6.2.6. PGU Client Computation Cost. In Step 3, obliviously evaluating the polynomial requires $O(mn)$ modular exponentiations. Computing n homomorphic multiplication requires $O(n)$ modular exponentiations. In Step 5, computing n homomorphic multiplication requires $O(n)$ modular exponentiations. In Step 8, encrypting the each element in B requires $O(q^2)$ modular exponentiations. Computing q^2 homomorphic addition and multiplication requires $O(q^2)$ modular exponentiations and $O(q^2)$ modular multiplication. As a result, the total computation cost for the client is $O(m n+q^2)$ modular exponentiations and $O(q^2)$ modular multiplications.

6.3. Leakage Analysis

6.3.1. PGI Leakage. As stated before, the proposed PGI leaks certain information about the private graphs, which is modeled as the leakage functions $\mathscr{L}_1$ and $\mathscr{L}_2$. There are several techniques that can be used to reduce the amount of information leakage; however, it cannot be completely avoided.

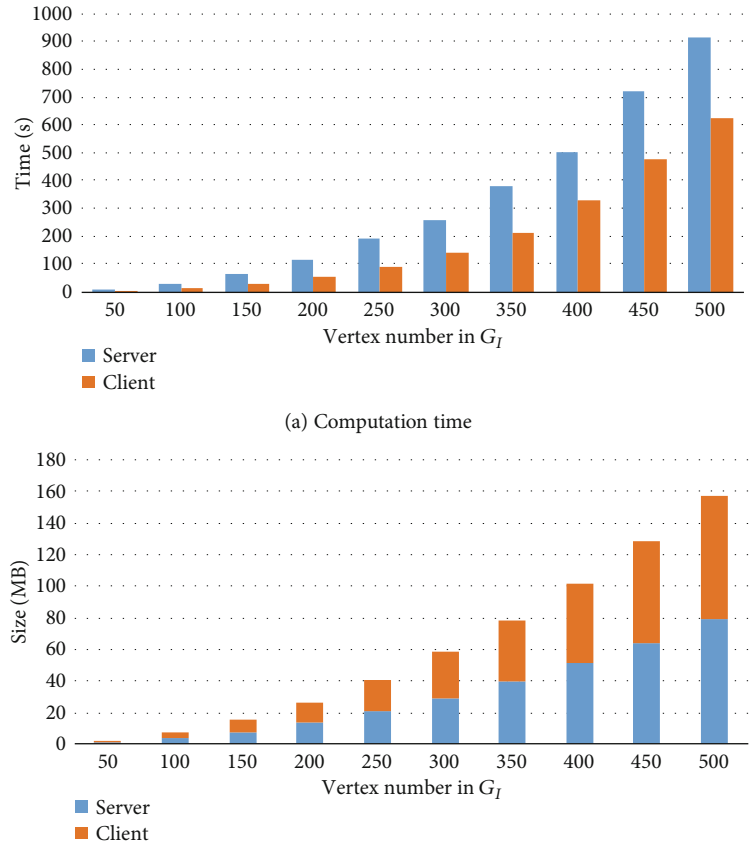
In Step 2, the server constructs a polynomial P, such that all the roots of $P(x)$ are exactly the elements in V_S. After that, the server sends the encryptions of the coefficients of P to the client. In order to prevent the client from learning the exact vertex number of the server's graph, the server first randomly constructs an irreducible polynomial $R(x)$ with degree d. The server then computes $P'(x)=P(x)R(x)$ and uses $P'(x)$ instead of $P(x)$ in Step 2. The polynomial $P'(x)$ has the same property as $P(x)$; therefore, it will not affect the result of the protocol. As a result, by counting the number of ciphertexts received, the client can only learn the upper bound of the vertex number of the server's graph, i.e., $m+d$.

In order to hide the exact vertex number of the client's graph, the client can randomly generate a set of h values from the message space of the Paillier cryptosystem in Step 3. After that, the client encrypts the random values and sends the encrypted random set to the server along with $\beta^{\oplus}$. Since the message space of the Paillier cryptosystem is large enough, the probability that a random value equals to an element in V_S can be assumed as negligible. Therefore, the random values will not affect the result of the protocol, since they are not in the vertex intersection. As a result, by counting the number ciphertexts received in Step 3, the server can only learn the upper bound of the vertex number of the client's graph, i.e., $n+h$.

6.3.2. PGU Leakage. Similar as PGI, PGU also leaks partial information about the input graphs during the process, which is modeled as $\mathscr{L}_3$ and $\mathscr{L}_4$.

In Step 2, the server can utilize the same technique, as introduced above, to hide the exact vertex number of his graph, and the client can only learn the upper bound instead, i.e., $m+d$.

In Step 3, in order to hide the exact vertex number of the client's graph, the client generates k encryptions of zero and sends the ciphertexts along with $\beta^{\oplus}$. An encryption of zero in Step 3 indicates that a vertex in the client's graph also exists in the server's graph. In later steps, extra encryptions of zero will not affect the final result, since the vertex union between the two input graphs will remain the same. As a result, the server can only learn the upper bound of the vertex number of the client's graph, i.e., $n+k$, and the upper bound of the common vertex number, i.e., $p+k$.



(a) Computation time

(b) Communication cost

Figure 2: Evaluation of private graph intersection protocol.

In addition, we consider the case where the server sends a graph with small size to the client in Step 2. If the server's graph is small enough, i.e., only 1 vertex and no edge, the union of the graphs will be almost the graph of the client. To prevent the server from learning the client's graph in such a method, there are two points where the client can choose to end the protocol.

The first point is at Step 3. If the client receives a very small polynomial, the client can choose to end the protocol, and at this point, the server has not learned anything yet. However, if the server uses the technique stated above, the polynomial that the client receives will not give the exact size of the server's graph. In this case, the client can check if the vertex union received in Step 8 is almost the same as his vertex set V_C. If $V_U \approx V_C$, it means either the server has a very small graph or the vertices in both graphs are highly overlapping. At this point, the client can choose to end the protocol; however, the server has already learned the vertex set of the client.

7. Experiments

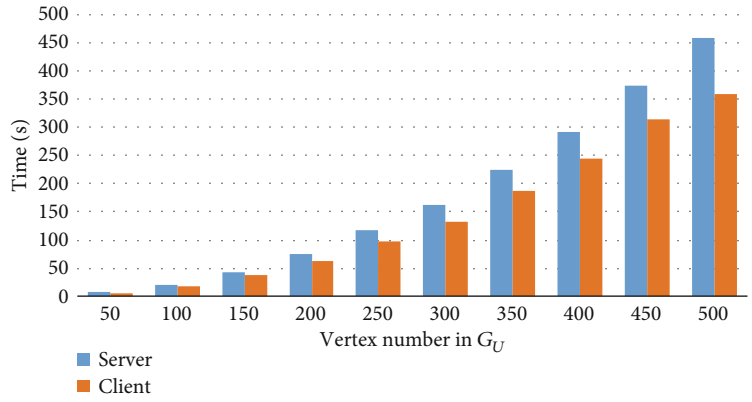
In order to evaluate the performances of the proposed PGI and PGU protocols, we implement the protocols and perform experiments over the Enron email dataset. All the experiments were conducted on two PCs with Intel Core i7-2600 4.2 GHz CPU, 16 GB RAM, and Windows 10 operating system. (Due to the COVID-19 crisis, we cannot access the lab in the university at the moment, which contains the environment and equipment to perform the experiments on real mobile devices. As a result, the experiments are performed on a PC in this paper, and we will improve the experiments on mobile devices in later works.). The protocols are implemented in Python 3.6, and we used the phe library for the Paillier cryptosystem with a 1024-bit key length.

7.1. Dataset. The Enron email dataset is publicly available from the Stanford SNAP website (https://snap.stanford.edu/data/). The dataset contains email communications of around half a million emails. In order to convert the dataset to a graph, the senders and the receivers of the emails are represented as vertices, and if vertex i sends at least one email to vertex j, there exists an undirected edge between i and j. The resulting graph has 36,692 vertices and 183,831 edges. In addition, each vertex of the graph is represented as a unique integer.

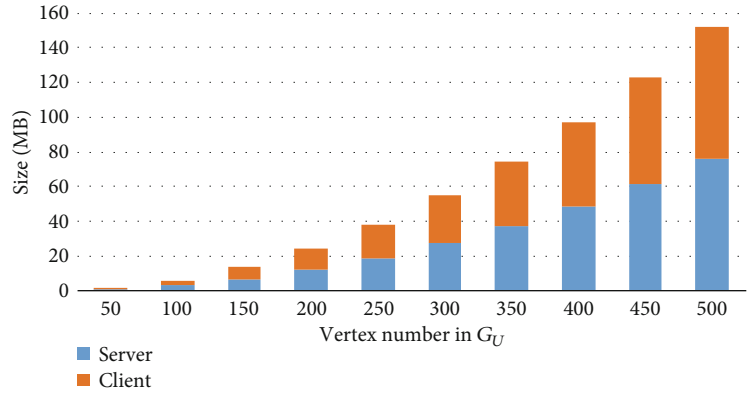
7.2. Evaluation of PGI. When evaluating the performance of PGI, we randomly generate two subgraphs from the Enron email graph dataset and assign them to the server and the client, respectively. For each experiment, we set m and n to have the same value, and they increase from 1,000 to 10,000. Furthermore, the graphs of the server and the client are generated following the rule that 5% of the vertices are the same

Table 2: The computation time for each step of private graph intersection protocol.

m/n	p	Step 2 time (s)	Step 3 time (s)	Step 4 time (s)	Step 5 time (s)	Step 6 time (s)	Step 7 time (s)
1,000	50	2.07	3.26	1.05	4.72	1.04	1.22
2,000	100	3.58	6.17	1.90	20.26	5.01	2.94
3,000	150	4.90	9.79	2.12	48.16	16.57	6.15
4,000	200	6.57	15.96	2.64	95.95	38.40	10.49
5,000	250	7.93	24.42	3.38	164.47	62.32	14.35
6,000	300	8.68	32.85	3.74	225.50	108.21	20.23
7,000	350	10.07	46.65	4.70	336.07	165.52	26.87
8,000	400	11.14	65.60	5.54	449.29	262.08	36.90
9,000	450	13.35	96.32	6.63	654.43	382.28	46.88
10,000	500	15.48	130.69	8.19	832.97	493.01	58.28



(a) Computation Time



(b) Communication Cost

Figure 3: Evaluation of private graph union protocol.

between the two graphs. Figure 2(a) shows the computation time for the server and the client.

As analyzed before, the computation costs for the server and the client are $O(m+n+p^2)$ and $O(mn+p^2)$, respectively, where p is the number of vertices in the intersection of G_S and G_C. Therefore, the most dominant part of the computation costs for both the server and the client is most likely to be the number of common vertices between G_S and G_C. As shown in Figure 2(a), the computation time for both the server and the client grows quadratically as the number of common vertices increases. The detailed computation time for each step is shown in Table 2.

As shown in Table 2, the most time-consuming parts of PGI are Steps 5 and 6. In Step 5, the server performs p^2 Paillier encryptions, and in Step 6, the client performs p^2 homomorphic multiplications. Since the computations for both Steps 5 and 6 are highly parallelizable, the computation time can be greatly reduced if cluster computing is deployed.

The communication costs of PGI for both the server and the client are shown in Figure 2(b). As analyzed before, the

Table 3: The computation time for each step of private graph union protocol.

m/n	q	Step 2 time (s)	Step 3 time (s)	Step 4 time (s)	Step 5 time (s)	Step 6 time (s)	Step 7 time (s)	Step 8 time (s)	Step 9 time (s)
30	50	0.54	1.13	0.52	0.45	0.46	4.10	5.12	1.74
60	100	0.59	1.23	0.50	0.50	0.52	14.32	16.62	5.43
90	150	0.66	1.32	0.47	0.47	0.51	31.17	35.66	10.94
120	200	0.68	1.26	0.45	0.45	0.53	54.90	62.11	19.25
150	250	0.77	1.30	0.48	0.48	0.52	84.50	96.92	30.05
180	300	0.80	1.32	0.52	0.52	0.65	117.55	130.72	43.26
210	350	0.82	1.38	0.48	0.48	0.63	162.46	185.12	59.09
240	400	0.80	1.48	0.49	0.49	0.65	212.23	242.75	77.69
270	450	0.95	1.59	0.58	0.58	0.65	275.12	311.50	94.98
300	500	0.87	1.52	0.53	0.53	0.63	336.23	355.94	118.34

total communication cost is $O(m + n + p^2)$. As a result, the communication costs have a quadratic growth in the figure. In addition, the communication costs are nearly the same for both the server and the client, and the overall communication cost for PGI is practical for the experimental dataset.

7.3. Evaluation of PGU. When evaluating the performance of PGU, we first randomly generate a subgraph from the Enron email graph dataset as the graph union G_U. Then, we randomly choose two subgraphs of G_U and assign them to the server and the client, respectively. The numbers of the vertices in the subgraphs are 60% of the vertex number in G_U; therefore, both m and n will have the same value. For each experiment, the number of vertices in G_U increases from 50 to 500. Figure 3(a) shows the computation time for the server and the client, and the detailed computation time for each step is shown in Table 3.

As analyzed before, the computation costs for PGU are $O(m + n + q^2)$ and $O(mn + q^2)$ for the server and the client, respectively, where q is the number of vertices in the union of G_S and G_C. Therefore, similar as PGI, the most dominant part of the computation costs for both the server and the client is most likely to be the number of vertices in G_U.

As shown in Table 3, most of the computation time is spent in Steps 7 and 8. In Step 7, the server performs q^2 Paillier encryptions, and in Step 8, the client performs q^2 Paillier encryptions and q^2 homomorphic additions and multiplications. Similar as before, the above computations are highly parallelizable, and cluster computing will greatly optimize the computation time.

As shown in Figure 3(b), the communication cost of PGU is similar to PGI, and the communication costs for both the server and the client have a quadratic growth as the number of vertices in G_U increases. For our experimental dataset, the overall communication cost for the PGU protocol is also practical.

8. Conclusion

In this work, we proposed two privacy-preserving graph operation protocols between two parties, which can be used for secure authentication for smart devices. The first protocol, PGI, allows a server and a client to jointly compute the intersection between their private graphs, while the second protocol, PGU, computes the union of the graphs. The protocols first use polynomial representation and oblivious polynomial evaluation to compute the intersection and union of the vertices. The intersection and union of the edges are then computed by using an additive homomorphic cryptosystem.

We proved that the proposed protocols are secure in the semihonest security model. In other words, a semihonest client learns nothing about the server's graph and a semihonest server learns nothing about the client's graph. We analyzed the leakages during the protocols for both the server and the client and modeled the leakages as leakage functions. At last, we implemented the constructions of the protocols and evaluated the efficiencies over real-word graph data.

Data Availability

The graph data used to support the findings of this study can be found at https://snap.stanford.edu/data/.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (61872069) and the Fundamental Research Funds for the Central Universities (N2017012).

References

[1] M. A. Khan and K. Salah, "IoT security: review, blockchain solutions, and open challenges," *Future Generation Computer Systems*, vol. 82, pp. 395–411, 2018.

[2] Z. K. Zhang, M. C. Y. Cho, C. W. Wang, C. W. Hsu, C. K. Chen, and S. Shieh, "IoT security: ongoing challenges and research opportunities," in *2014 IEEE 7th international conference on service-oriented computing and applications*, pp. 230–234, Matsue, Japan, 2014.

[3] M. El-hajj, A. Fadlallah, M. Chamoun, and A. Serhrouchni, "A survey of internet of things (IoT) authentication schemes," *Sensors*, vol. 19, no. 5, p. 1141, 2019.

[4] M. Wazid, A. K. Das, R. Hussain, G. Succi, and J. J. Rodrigues, "Authentication in cloud-driven IoT-based big data environment: survey and outlook," *Journal of Systems Architecture*, vol. 97, pp. 185–196, 2019.

[5] I. H. Chuang, B. J. Guo, J. S. Tsai, and Y. H. Kuo, "Multi-graph zero-knowledge-based authentication system in internet of things," in *2017 IEEE International Conference on Communications (ICC)*, pp. 1–6, Paris, France, 2017.

[6] C. Lin, D. He, X. Huang, M. K. Khan, and K. K. R. Choo, "A new transitively closed undirected graph authentication scheme for blockchain-based identity management systems," *IEEE Access*, vol. 6, pp. 28203–28212, 2018.

[7] Z. Shao, Z. Li, P. Wu, L. Chen, and X. Zhang, "Multi-factor combination authentication using fuzzy graph domination model," *Journal of Intelligent & Fuzzy Systems*, vol. 37, no. 4, pp. 4979–4985, 2019.

[8] S. S. Sonawane and P. A. Kulkarni, "Graph based representation and analysis of text document: a survey of techniques," *International Journal of Computer Applications*, vol. 96, no. 19, pp. 1–8, 2014.

[9] T. Washio and H. Motoda, "State of the art of graph-based data mining," *Acm Sigkdd Explorations Newsletter*, vol. 5, no. 1, pp. 59–68, 2003.

[10] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 974–983, New York, USA, 2018.

[11] S. Aridhi and E. M. Nguifo, "Big graph mining: frameworks and techniques," *Big Data Research*, vol. 6, pp. 1–10, 2016.

[12] H. Rong, T. Ma, M. Tang, and J. Cao, "A novel subgraph k^+-isomorphism method in social network based on graph similarity detection," *Soft Computing*, vol. 22, no. 8, pp. 2583–2601, 2018.

[13] E. Ko, M. Kang, H. J. Chang, and D. Kim, "Graph-theory based simplification techniques for efficient biological network analysis," in *2017 IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService)*, pp. 277–280, San Francisco, CA, USA, 2017.

[14] V. Mukhin, Y. Romanenkov, J. Bilokin et al., "The method of variant synthesis of information and communication network structures on the basis of the graph and set-theoretical models," *International Journal of Intelligent Systems and Applications*, vol. 9, no. 11, pp. 42–51, 2017.

[15] Q. Jiang, N. Zhang, J. Ni, J. Ma, X. Ma, and K. K. R. Choo, "Unified biometric privacy preserving three-factor authentication and key agreement for cloud-assisted autonomous vehicles," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 9, pp. 9390–9401, 2020.

[16] F. Zhou, Z. Xu, Y. Li, J. Xu, and S. Peng, "Private graph intersection protocol," in *Australasian Conference on Information Security and Privacy*, pp. 235–248, Springer, 2017.

[17] Y. Jia, Y. Xiao, J. Yu, X. Cheng, Z. Liang, and Z. Wan, "A novel graph-based mechanism for identifying Trac vulnerabilities in smart home IoT," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 1493–1501, Honolulu, HI, USA, 2018.

[18] F. Zhu, W. Wu, Y. Zhang, and X. Chen, "Privacy-preserving authentication for general directed graphs in industrial IoT," *Information Sciences*, vol. 502, pp. 218–228, 2019.

[19] S. Micali and R. L. Rivest, "Transitive signature schemes," in *Cryptographers' Track at the RSA Conference*, Privacy-preserving Graph Operations for Mobile Authentication, pp. 236–243, Springer, 2002.

[20] A. C. C. Yao, "Protocols for secure computations," in *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, vol. 82, pp. 160–164, Chicago, IL, USA, 1982.

[21] O. Goldreich, S. Micali, and A. Wigderson, *How to play any mental game or a completeness theorem for protocols with honest majority*, STOC, 1987.

[22] M. Blanton and E. Aguiar, "Private and oblivious set and multiset operations," *International Journal of Information Security*, vol. 15, no. 5, pp. 493–518, 2016.

[23] K. Banawan and S. Ulukus, "The capacity of private information retrieval from coded databases," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1945–1956, 2018.

[24] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 223–238, Springer, 1999.

[25] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *International conference on the theory and applications of cryptographic techniques*, pp. 1–19, Springer, 2004.

[26] H. Chen, K. Laine, and P. Rindal, "Fast private set intersection from homomorphic encryption," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1243–1255, New York, USA, 2017.

[27] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, "Spot-light: lightweight private set intersection from sparse OT extension," in *Annual International Cryptology Conference*, pp. 401–431, Springer, 2019.

[28] B. Pinkas, T. Schneider, and M. Zohner, "Scalable private set intersection based on OT extension," *ACM Transactions on Privacy and Security (TOPS)*, vol. 21, no. 2, pp. 1–35, 2018.

[29] D. Wang and P. Wang, "On the anonymity of two-factor authentication schemes for wireless sensor networks: attacks, principle and solutions," *Computer Networks*, vol. 73, pp. 41–57, 2014.

[30] X. Zhang, X. Chen, J. K. Liu, and Y. Xiang, "DeepPAR and DeepDPA: privacy preserving and asynchronous deep learning for industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 2081–2090, 2019.

[31] Q. Jiang, Y. Qian, J. Ma, X. Ma, Q. Cheng, and F. Wei, "User centric three-factor authentication protocol for cloud-assisted wearable devices," *International Journal of Communication Systems*, vol. 32, no. 6, article e3900, 2019.

[32] C. Wang, D. Wang, Y. Tu, G. Xu, and H. Wang, "Understanding node capture attacks in user authentication schemes for wireless sensor networks," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[33] D. Wang, W. Li, and P. Wang, "Measuring two-factor authentication schemes for real-time data access in industrial wireless sensor networks," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 9, pp. 4081–4092, 2018.

[34] D. Wang, D. He, P. Wang, and C. H. Chu, "Anonymous two-factor authentication in distributed systems: certain goals are beyond attainment," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 4, pp. 428–442, 2014.