Achieving Efficient and Privacy-Preserving (α,β)-Core Query over Bipartite Graphs in Cloud

Yunguo Guan, Rongxing Lu, *Fellow, IEEE,* Yandong Zheng, Songnian Zhang, Jun Shao, *Member, IEEE,* and Guiyi Wei

Abstract—Bipartite graphs have been widely adopted in applications such as e-healthcare and recommendation systems thanks to their ability to model various real-world relationships. Meanwhile, (α, β)-core query services over bipartite graphs are generally recognized as a promising approach for finding communities, i.e., closely related sets of vertices, in a bipartite graph. As the bipartite graph grows, the service providers tend to outsource the services to the cloud for flexible and highly reliable computational resources. However, since the cloud is not fully trustable, there are privacy concerns related to the dataset, query requests, and results. Although many schemes have been proposed for privacy-preserving queries over graphs, they cannot be directly adopted to handle accurate (α, β)-core queries over bipartite graphs. Aiming at the challenges, under the two-server setting, this paper constructs two privacy-preserving schemes with different levels of security to handle (α, β)-core queries over the bipartite graph. Specifically, in the proposed schemes, a graph is represented as an index containing an edge table and a node table and further encrypted by a symmetric homomorphic encryption scheme, and then the two servers securely traverse the index. Detailed security analysis shows that both schemes can achieve access pattern privacy, while the security-enhanced one can further protect the structure privacy of the query requests and results. In addition, extensive performance evaluations are conducted, and the results also indicate our proposed schemes are computationally efficient.

Index Terms—Community detection, (α, β)-core query, bipartite graph, privacy-preserving

1 Introduction

Bipartite graphs have been widely adopted in various applications, as they can model real-world relationships among different types of entities, e.g., author-paper [1], patient-disease [2], customer-product [3]. Meanwhile, many techniques have been proposed for mining insights from bipartite graphs, and among them, (α, β)-core query is a promising one that can achieve fault-tolerant group recommendation [4] and community discovery [5]. For instance, given a bipartite graph representing the relationships between patients and medical services, a doctor can query with several vertices (including patients and/or medical services) to obtain a maximal connected subgraph of the bipartite satisfying that, i) each patient in the subgraph links to at least α medical services in the subgraph; ii) each medical service in the subgraph links to at least β patients in the subgraph; and iii) the query vertices are included in the subgraph. The vertices in the obtained subgraph form a community that are closely related to the query vertices and may provide some useful information (e.g., advice on medical services) to the doctor [6]. Furthermore, as the bipartite graph grows, the service provider tends to outsource the query services to a powerful cloud, similar to other outsourced services in the cloud [7]–[9].

- *Y. Guan, R. Lu, Y. Zheng, and S. Zhang are with the Faculty of Computer Science, University of New Brunswick, Fredericton, Canada E3B5A3. E-mail: yguan4@unb.ca, rlu1@unb.ca, yzheng8@unb.ca, szhang17@unb.ca.*
- *J. Shao and G. Wei are with Zhejiang Gongshang University, Hangzhou, China 310018. E-mail: chn.junshao@gmail.com, weigy@zjgsu.edu.cn.*

However, directly outsourcing the services to the not-fully-trustable cloud raises privacy concerns for both the service provider and query users. On the one hand, the dataset should only be available to authorized query users, as it is a private asset of the service provider. On the other hand, the query requests and results should also be protected against other participants in the system, since they may reveal some private information related to the query users. Therefore, the original bipartite graph and query requests must be encrypted before being outsourced to the cloud. Nevertheless, it is commonly acknowledged that encryption techniques will hinder the usability of the dataset. As detailed in Section 7, many schemes [10]–[26] have been proposed to conduct queries over graph data while preserving data privacy. However, some of them [10]–[17] preserve data privacy of the original graph through k-anonymity, but they can neither be adapted to handle accurate (α, β)-core query, nor preserve the privacy of query requests. Some others [18]–[28] are built upon homomorphic encryption and some customized secure computing protocols. They cannot be trivially adapted to our (α, β)-core query scenario over bipartite graphs. Therefore, it is still challenging to design an efficient and privacy-preserving scheme that supports (α, β)-core queries over bipartite graphs.

In this paper, aiming at the above challenges, we propose our efficient and privacy-preserving (α, β)-core query scheme over bipartite graphs in cloud. Specifically, the contributions of our paper are three-fold.

• First, we build an index containing two tables obtained from the bipartite graph to support (α, β)-core queries, where each row in the two tables respectively represents a vertex and an edge in the original graph. Then, we propose

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3169386, IEEE Transactions on Dependable and Secure Computing

our basic (α, β)-core query scheme with two non-collusive cloud servers. In the basic scheme, the index and query requests are encrypted by a symmetric homomorphic encryption (SHE) scheme [29], thus the privacy of the original bipartite graph and the query requests are preserved.

• Second, to further protect the privacy of query results' structures, we revise the two tables in the index and build an encrypted queue based on the SHE scheme. Then, we construct our security-enhanced version, where the cloud servers cannot obtain any useful information related to the bipartite graph, query requests, and query results.

• Finally, we analyze the security of the two proposed schemes and compare their performance on two real datasets. The security analysis shows that the basic scheme can well preserve the privacy of the bipartite graph and query requests, and the security-enhanced one can further preserve the privacy of query results' structures. The experimental result also shows that our proposed schemes are indeed efficient.

The remainder of this paper is organized as follows. In Section 2, we recall the definition of (α, β)-cores and an SHE scheme as our preliminaries. Then, we formalize the system model and security model, and identify our design goal in Section 3. In Section 4, we present our basic scheme and analyze its security. After that, we propose a scheme with enhanced security in Section 5 followed by its security analysis. In Section 6, we demonstrate the performance of the two proposed schemes. We also review some related works in Section 7. Finally, we conclude this work in Section 8.

2 Preliminaries

2.1 Bipartite Graph and (α, β)-Core

A bipartite graph is a graph whose vertices can be divided into two disjoint and independent sets $\mathcal{U}$ and $\mathcal{V}$ such that each edge connects one vertex in $\mathcal{U}$ and the other in $\mathcal{V}$. By denoting a bipartite graph as a triple $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$, we have the edge set $\mathcal{E} \subseteq (\mathcal{U} \times \mathcal{V})$.

An (α, β)-core of $\mathcal{G}$, denoted by $\mathcal{G}' = (\mathcal{U}', \mathcal{V}', \mathcal{E}')$, is a subgraph of $\mathcal{G}$ satisfying that i) it is a connected subgraph; ii) the degree of each vertex in $\mathcal{U}'$ is not smaller than α, i.e., $deg(u_i) \geq \alpha$, for each $u_i \in \mathcal{U}'$; iii) the degree of each vertex in $\mathcal{V}'$ is not smaller than β, i.e., $deg(v_j) \geq \beta$, for each $v_j \in \mathcal{V}'$; and iv) it is maximal, i.e., the resulting subgraph obtained by adding one or more edges from $\mathcal{E} \setminus \mathcal{E}'$ and the corresponding vertices to $\mathcal{G}'$ will not satisfy the previous three conditions. An example of a bipartite and its (α, β)-core is shown in Fig. 1, where $\alpha = 2$ and $\beta = 3$.

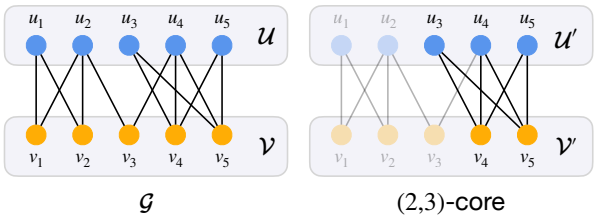


Fig. 1. An example of a bipartite and its (α, β)-core, where $\alpha = 2$ and $\beta = 3$. As shown in the figure, all $u_i \in \mathcal{U}' = \{u_3, u_4, u_5\}$ satisfy $deg(u_i) = 2 \geq \alpha$, and each $v_j \in \mathcal{V}' = \{v_4, v_5\}$ satisfy $deg(v_j) = 3 \geq \beta$.

Then, we formally define (α, β)-core queries as follows.

Definition 1 ((α, β)-Core Queries). *Given a bipartite graph $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$, an (α, β)-core query, denoted by a 3-tuple $\mathsf{req} = \{\mathcal{R} \subseteq (\mathcal{U} \cup \mathcal{V}), \alpha, \beta\}$, is to obtain a maximal subgraph $\mathcal{G}_{\mathsf{req}} = (\mathcal{U}_{\mathsf{req}}, \mathcal{V}_{\mathsf{req}}, \mathcal{E}_{\mathsf{req}}) \subseteq \mathcal{G}$ satisfying that i) it consists of one or multiple disjoint (α, β)-cores; and ii) each connected component in $\mathcal{G}_{\mathsf{req}}$ contains at least one vertex in $\mathcal{R}$.*

2.2 SHE Scheme

In this subsection, we recall a symmetric homomorphic encryption (SHE) scheme [29], which is IND-CPA secure [8], and will be used in our proposed scheme. It comprises three algorithms, namely, *Key Generation*, *Encryption*, and *Decryption*.

• **Key Generation:** Given security parameters k_0, k_1, k_2 satisfying $k_1 \ll k_2 < k_0/2$, the key generation algorithm randomly selects an integer $\mathcal{L}$ and two prime numbers p and q such that $|\mathcal{L}| = k_2$ and $|p| = |q| = k_0$. Then, it outputs the public parameter $PP = (k_0, k_1, k_2, \mathcal{N} = pq)$ and a secret key $SK = (p, \mathcal{L})$. Furthermore, the basic plaintext space is defined as $\mathcal{M} = [-2^{k_1-1}, 2^{k_1-1})$.

• **Encryption:** Given PP and SK, the algorithm first randomly selects two random numbers $r \in \{0,1\}^{k_2}$ and $r' \in \{0,1\}^{k_0}$. Then, a message $m \in \mathcal{M}$ can be encrypted as $c = E(m) = (m + r\mathcal{L})(1 + r'p) \bmod \mathcal{N}$.

• **Decryption:** With PP and SK, a ciphertext c can be decrypted in two steps. First, the algorithm computes $\widetilde{m} = (c \bmod p) \bmod \mathcal{L}$. Then, it outputs $m = \widetilde{m}$ if $\widetilde{m} < \mathcal{L}/2$; otherwise, it outputs $m = \widetilde{m} - \mathcal{L}$. As $k_1 \ll k_2 < k_0/2$, we have $m + r\mathcal{L} < p$ and $|m| < \mathcal{L}/2$, and thereby, the correctness of the decryption holds.

Homomorphic Properties. The SHE scheme enjoys the following homomorphic addition (Homo-Add) and homomorphic multiplication (Homo-Mul) properties. Specifically, given two ciphertexts $c_1 = E(m_1)$ and $c_2 = E(m_2)$, we have i) Homo-Add-I: $(c_1 + c_2) \bmod \mathcal{N} \rightarrow E(m_1 + m_2)$; ii) Homo-Mul-I: $(c_1 \times c_2) \bmod \mathcal{N} \rightarrow E(m_1 \times m_2)$; iii) Homo-Add-II: $(c_1 + m_2) \bmod \mathcal{N} \rightarrow E(m_1 + m_2)$; and iv) Homo-Mul-II: $(c_1 \times m_2) \bmod \mathcal{N} \rightarrow E(m_1 \times m_2)$, when $m_2 > 0$.

Note that, the SHE scheme is a leveled homomorphic encryption. Given security parameters k_0 and k_2, its multiplicative depth $\delta = \frac{k_0}{2k_2} - 1$.

3 Models and Design Goal

In this section, we formalize our system model, security model, and identify our design goal.

3.1 System Model

In this paper, we consider a privacy-preserving (α, β)-core query scenario, which contains three types of entities, namely, a data owner DO, a cloud containing two cloud servers $\mathcal{CS} = \{\mathsf{CS}_1, \mathsf{CS}_2\}$, and a set of query users, as shown in Fig. 2.

• **Data Owner:** In our system model, the data owner DO has a bipartite graph $\mathcal{G} = \{\mathcal{U}, \mathcal{V}, \mathcal{E}\}$. Note that, since we aim to design a privacy-preserving (α, β)-core query scheme, we assume that the vertices in $\mathcal{U}$ and $\mathcal{V}$ are labeled by continuous integers. To make better use of the graph,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3169386, IEEE Transactions on Dependable and Secure Computing

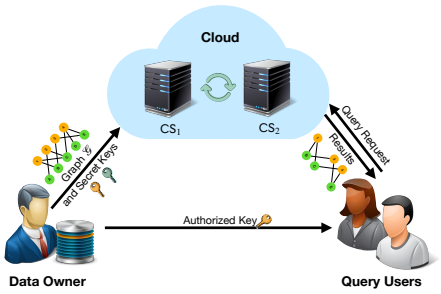


Fig. 2. The system model under consideration.

he/she wants to offer (α, β)-core query services to query users, which may exceed the computation capacity of DO. Therefore, DO is willing to outsource the dataset and the services to the cloud $\mathcal{CS}$. Furthermore, to preserve the privacy of the dataset, DO will encrypt the dataset before outsourcing it to the cloud.

• **Cloud:** The cloud servers CS_1 and CS_2 in our system model are equipped with powerful computation resource and plentiful storage space, and they are employed to provide the (α, β)-core query services. That is, upon receiving a query request from a query user, they jointly conduct the query on the outsourced dataset $\mathcal{G}$ and return the query result to the query user.

• **Query Users:** In our system model, after being authorized by the data owner DO, a query user can launch (α, β)-core queries to the cloud. That is, it can launch a (α, β)-core query request $\mathsf{req} = \{\mathcal{R}, \alpha, \beta\}$ to the cloud $\mathcal{CS}$ to obtain all (α, β)-cores related to the vertex set $\mathcal{R}$. Similar to DO, the query user needs to encrypt req before submitting it to $\mathcal{CS}$.

3.2 Security Model

In our security model, we consider the data owner is *trusted*, as he/she owns the dataset and has no motivation to deviate from the services. For the query users, they are considered to be *honest*, i.e., they will faithfully encrypt their query requests. Furthermore, the two cloud servers CS_1 and CS_2 are considered to be *honest-but-curious*. That is, they will faithfully conduct user queries but might be curious about the structure (including connectivity between vertices and the degrees of vertices) or the plaintext of not only the dataset, query requests and the corresponding results but also the access pattern during executing query. However, they will not collude with each other. This no-collusion assumption is reasonable as the collusion of the two cloud servers may damage the reputation of the corresponding cloud service providers, and it has already been adopted in many research works in the security community [30]–[33].

Note that, adversaries may also launch other active attacks, e.g., Denial-of-Service (DoS) attack. As this paper focuses on privacy preservation in (α, β)-core queries, those attacks are out of the scope and will be discussed in our future work.

3.3 Design Goal

The design goal of this paper is to propose an efficient and privacy-preserving (α, β)-core query scheme which should have the following two properties.

• *The proposed scheme should be privacy-preserving.* As the outsourced dataset, the query requests and the corresponding results are private to the data owner and the query users, the proposed scheme should be privacy-preserving. Specifically, the cloud servers should not be able to obtain not only the plaintext and structure of the outsourced dataset, query requests, and query results but also access patterns during executing queries.

• *The proposed scheme should be efficient.* To preserve data privacy, the proposed scheme will employ some cryptographic techniques that might be computationally expensive. However, to make the proposed scheme practical, its efficiency should also be taken into consideration.

4 Our Proposed Basic Scheme

In this section, we present our basic efficient and privacy-preserving (α, β)-core query scheme. We first give a general idea of the proposed scheme. Then, we describe our basic scheme followed by its security analysis.

4.1 Main Idea of Our Basic Scheme

As shown in Fig. 3, given a bipartite graph $\mathcal{G}$, we can easily observe that vertices in $\mathcal{G}$ will be pruned as α and β increase. That is, if a vertex $n \in (\mathcal{U} \cup \mathcal{V})$ is not included in an (α, β)-core of $\mathcal{G}$, it will not be included in any (α', β')-core, where $\alpha' \geq \alpha$, $\beta' \geq \beta$, and $\alpha' + \beta' > \alpha + \beta$. Based on this observation, we can compute a table $\mathcal{T} = \{\widehat{\beta}_{n,\alpha} \mid n \in (\mathcal{U} \cup \mathcal{V}), 1 \leq \alpha \leq \max_{u \in \mathcal{U}} deg(u)\}$ representing the maximum β such that the vertex $n \in (\mathcal{U} \cup \mathcal{V})$ is included in an (α, β)-core. For example, $\widehat{\beta}_{u_2,3} = 1$ means that the vertex u_2 is an element of a $(3, 1)$-core but is not an element of a $(3, 2)$-core, as shown in Fig. 3. In addition, if two connected vertices n and n' satisfy $\widehat{\beta}_{n,\alpha} \geq \beta$ and $\widehat{\beta}_{n',\alpha} \geq \beta$, then they are in the same (α, β)-core, otherwise these (α, β)-cores are not maximal and can be merged. Therefore, based on $\mathcal{T}$, we can efficiently compute an (α, β)-core for a specific query set $\mathcal{R} \subseteq (\mathcal{U} \cup \mathcal{V})$ by obtaining the maximal connected subgraph for each vertex in $\mathcal{R}$ where each vertex n in the subgraph satisfies $\widehat{\beta}_{n,\alpha} \geq \beta$, as shown in Alg. 1. To improve the efficiency, for each vertex $n \in (\mathcal{U} \cup \mathcal{V})$ and each α, we can further build a linked list of the edges connecting n and its neighbors n', in which the edges are sorted by $\widehat{\beta}_{n',\alpha}$. For instance, the vertex u_5 has three neighbors $\{v_2, v_4, v_5\}$, and as shown in Fig. 3(b), for $n' = v_2, v_4, v_5$, $\widehat{\beta}_{n',\alpha}$ are respectively 2, 2, and 1 when $\alpha = 1$. Then, we can build the following linked list.

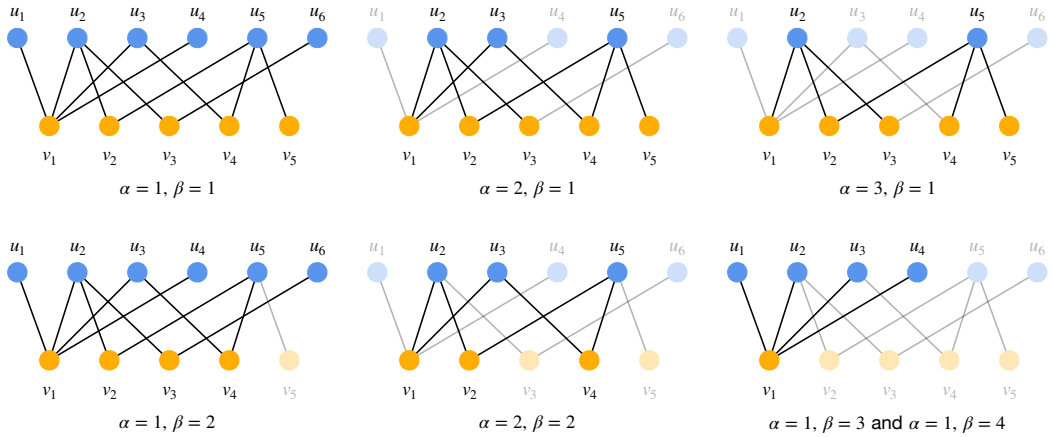
$$(u_5 \to v_2, 2) \to (u_5 \to v_4, 2) \to (u_5 \to v_5, 1) \to \perp$$

While extracting an (α, β)-core, we can traverse the neighbors of n through the linked list until encountering a neighbor n' whose $\widehat{\beta}_{n',\alpha} < \beta$. In this way, we can reduce the number of traversed neighbors and hide the number of vertices linked to n, i.e., $deg(n)$.

4.2 Description of Our Basic Scheme

Based on the idea presented in Section 4.1, we propose our basic privacy-preserving (α, β)-core query scheme, which comprises four algorithms, namely, *System Initialization*, *Graph Outsourcing*, *Query Encryption*, and *Query Conducting*.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3169386, IEEE Transactions on Dependable and Secure Computing



(a) Vertices are increasingly pruned as α and β increase.

	$\alpha = 1$	$\alpha = 2$	$\alpha = 3$
u_1	4	0	0
u_2	4	2	1
u_3	4	2	0
u_4	4	0	0
u_5	2	2	1
u_6	2	0	0
v_1	4	2	1
v_2	2	2	1
v_3	2	1	1
v_4	2	2	1
v_5	1	1	1

(b) $\widehat{\beta}_{n,\alpha}$ for all $n \in (\mathcal{U} \cup \mathcal{V})$.

Fig. 3. Given a graph $\mathcal{G} = \{\mathcal{U}, \mathcal{V}, \mathcal{E}\}$, we compute the maximum β for each vertex $n \in (\mathcal{U} \cup \mathcal{V})$ with different α, such that n is an element of an (α, β)-core. For example, when $\alpha = 3$, u_2 is not pruned until $\beta = 2$. Hence, $\widehat{\beta}_{u_2,3} = 1$, as shown in the table.

Algorithm 1 Conducting (α, β)-core query with $\mathcal{T}$

Input: the query request $\mathsf{req} = \{\mathcal{R}, \alpha, \beta\}$, the table $\mathcal{T}$
Output: the (α, β)-core query result $\mathcal{G}_{\mathsf{req}}$

Initialize an empty queue $\mathtt{Q}$
Initialize a mapping $\mathtt{visited}$ maps $n \in (\mathcal{U} \cup \mathcal{V})$ to a boolean **false**
for all $n \in \mathcal{R}$ **do**
  Enqueue n into $\mathtt{Q}$ if $\widehat{\beta}_{n,\alpha} \geq \beta$
Set $\mathtt{visited}[n] =$ **true** for each $n \in Q$
while $\mathtt{Q} \neq \varnothing$ **do**
  Take a vertex n from $\mathtt{Q}$
  Add n to the $\mathcal{U}_{\mathsf{req}}$ or $\mathcal{V}_{\mathsf{req}}$
  for all $n' \in \mathtt{neighbor}(n)$ **do**
    Obtain $\widehat{\beta}_{n',\alpha}$ from $\mathcal{T}$
    if $\widehat{\beta}_{n',\alpha} \geq \beta$ **and** $\mathtt{visited}[n'] ==$ **false then**
      Enqueue n' into $\mathtt{Q}$
      Set $\mathtt{visited}[n'] =$ **true**
Compute $\mathcal{E}_{\mathsf{req}} = \mathcal{E} \cap (\mathcal{U}_{\mathsf{req}} \times \mathcal{V}_{\mathsf{req}})$
return $\mathcal{G}_{\mathsf{req}} = (\mathcal{U}_{\mathsf{req}}, \mathcal{V}_{\mathsf{req}}, \mathcal{E}_{\mathsf{req}})$

4.2.1 System Initialization

In this algorithm, the data owner DO generates the secrets for all entities in the system and securely distributes the secrets. Specifically, DO first runs the *Key Generation* algorithm of the SHE scheme to obtain a public parameter PP and the corresponding secret key SK. Then, DO computes two SHE ciphertexts of 0, i.e., $\{E(0)_1, E(0)_2\}$. Furthermore, DO generates two secret keys $\mathtt{K}_1$ and $\mathtt{K}_2$ for a symmetric-key encryption scheme $\mathtt{SE}(\cdot)$, e.g., Advanced Encryption Standard (AES). After that, DO publishes $(PP, E(0)_1, E(0)_2)$, and respectively distributes $\mathtt{K}_1$ and $(SK, \mathtt{K}_2)$ to CS_1 and CS_2. Finally, it sends $(\mathtt{K}_1, \mathtt{K}_2)$ to authorized query users.

4.2.2 Graph Outsourcing

With the SK and PP generated during the system initialization, DO encrypts the bipartite graph $\mathcal{G}$ by running the following steps.

• First, DO computes the maximum degree of vertices in the graph $\mathcal{G}$, i.e., $\alpha_{max} = \max_{u \in \mathcal{U}} deg(u)$. Then, for $\alpha_k = 1, 2, \cdots, \alpha_{max}$, DO calculates $\widehat{\beta}_{n,\alpha_k}$ for each vertex $n \in (\mathcal{U} \cup \mathcal{V})$ and builds the table $\mathcal{T}$.

• Then, as shown in Fig. 4, DO builds two tables NodeTable and EdgeTable in the following steps.

1) For each α_k and each vertex $n \in (\mathcal{U} \cup \mathcal{V})$, DO builds a linked list L_{n,α_k} containing all edges connecting n and its neighbors n', and the elements in L_{n,α_k} are sorted by the corresponding $\widehat{\beta}_{n',\alpha_k}$ in descending order.

2) DO builds a table NodeTable where each row represents a vertex $n \in (\mathcal{U} \cup \mathcal{V})$, and it has $2 \times \alpha_{max}$ columns. For each $\alpha_k = 1, \cdots, \alpha_{max}$, there are two columns representing the head of the linked list L_{n,α_k} and the maximum beta value, denoted by $n.\mathtt{head}_{\alpha_k}$ and $n.\widehat{\beta}_{\alpha_k}$, respectively.

3) DO builds a table EdgeTable where each row represents an edge $e_{i,j} = (u_i, v_j) \in \mathcal{E}$, and there are $2 + 2 \times \alpha_{max}$ columns in EdgeTable. In addition to two pointers linked to u_i and v_j in NodeTable, there are α_{max} groups of columns. For the α_k-th group, there are two columns, namely, $e_{i,j}.\mathtt{next}_{\mathtt{u},\alpha_k}$ and $e_{i,j}.\mathtt{next}_{\mathtt{v},\alpha_k}$ representing the next edges in $L_{\mathtt{u},\alpha_k}$ and $L_{\mathtt{v},\alpha_k}$, respectively.

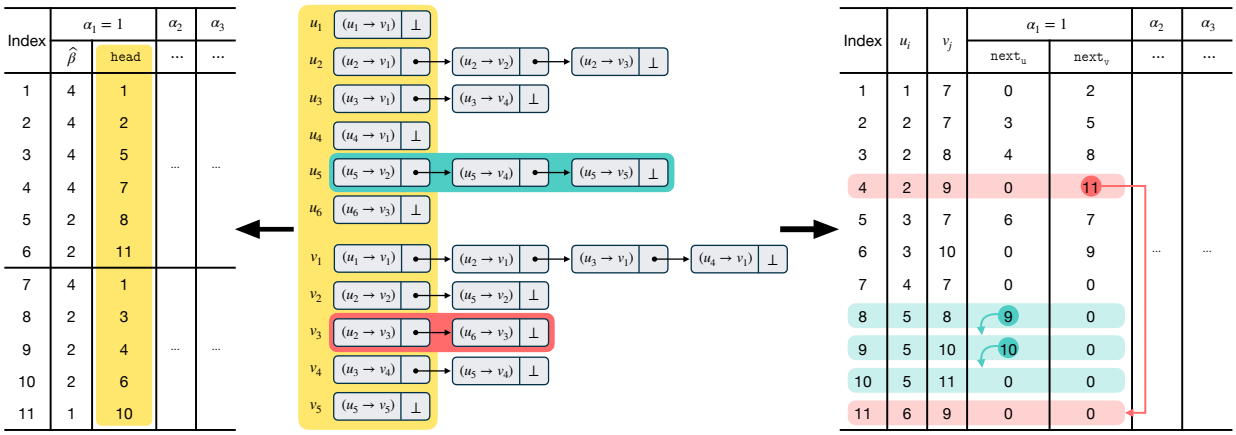
• After that, the data owner DO encrypts the two tables with PP and SK. For each pointer $\mathtt{ptr}(e_{i,j})$ (resp., $\mathtt{ptr}(u_i)$ or $\mathtt{ptr}(v_j)$) to a row in EdgeTable (resp., NodeTable), to support **OP1** (row/column retrieving) in Section 4.2.4, we encrypt it in a little-endian bit-wise manner and get an array of $\ell_{\mathtt{e}} = \lceil \log_2(|\mathcal{E}| + 1) \rceil$ (resp., $\ell_{\mathtt{n}} = \lceil \log_2(|\mathcal{U} \cup \mathcal{V}| + 1) \rceil$) SHE ciphertexts. For instance, given $|\mathcal{U} \cup \mathcal{V}| = 11$, the pointer $\mathtt{ptr}(u_3) = 3$ of the edge $(u_3 \rightarrow v_1)$ in EdgeTable will be encrypted into an array of $\lceil \log_2 12 \rceil = 4$ SHE ciphertexts, denoted as $[\![\mathtt{ptr}(u_3)]\!] = \{E(1), E(1), E(0), E(0)\}$. Note that, as SHE is IND-CPA secure, the two $E(1)$s here denote two different ciphertexts of 1, and the same case for the two $E(0)$s. As for the non-pointer field, i.e., $n.\widehat{\beta}_{\alpha_k}$, it will be encrypted into an SHE ciphertext $n.E(\widehat{\beta}_{\alpha_k})$.

At the end of the algorithm, the data owner DO uploads the two encrypted tables, denoted by $[\![\mathsf{NodeTable}]\!]$ and $[\![\mathsf{EdgeTable}]\!]$, to the cloud server CS_1, and the latter stores them for answering (α, β)-core queries.

4.2.3 Query Encryption

As (α, β)-core queries may contain some sensitive information, query users need to encrypt them before submitting them to the cloud. Note that, without knowing the secret key SK, the query users can generate SHE ciphertexts through

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3169386, IEEE Transactions on Dependable and Secure Computing



(a) NodeTable (b) Linked Lists of edges when $\alpha = 1$ (c) EdgeTable

Fig. 4. The linked lists of neighbors n for each vertex sorted by their $\widehat{\beta}_{n,\alpha}$ when $\alpha = 1$, and NodeTable and NodeTable derived from the linked lists. As shown in the figure, the heads of the linked lists are organized in NodeTable and linked lists are saved in EdgeTable.

Eq. (1), where $r_1, r_2 \in \{0,1\}^{k_2}$ are randomly chosen, and the obtained SHE ciphertexts are still IND-CPA secure [34].

$$E(m) = m + r_1 \cdot E(0)_1 + r_2 \cdot E(0)_2 \bmod \mathcal{N} \quad (1)$$

Then, given an (α, β)-core query request $\mathsf{req} = \{\mathcal{R}, \alpha, \beta\}$, a query user runs the following steps to build a query token.

• For each $n \in \mathcal{R}$, the query user will convert it into a pointer $\mathtt{ptr}(n)$ and encrypts it into $\ell_{\mathtt{n}} = \lceil \log_2(|\mathcal{U} \cup \mathcal{V}| + 1) \rceil$ SHE ciphertexts following the same approach used in Section 4.2.2. In addition, the query user attaches a bit $\mathtt{flag}_n$ to the encrypted pointer, where $\mathtt{flag}_n = 1$ if $n \in \mathcal{U}$, and $\mathtt{flag}_n = 0$ if $n \in \mathcal{V}$. Thereby, the query user obtains $[\![\mathcal{R}]\!] = \{([\![\mathtt{ptr}(n)]\!], \mathtt{flag}_n) \mid n \in \mathcal{R}\}$.

• Similarly, the query user encrypts α into an array of $\ell_\alpha = \lceil \log_2(\alpha_{max} + 1) \rceil$ SHE ciphertexts, denoted by $[\![\mathtt{ptr}(\alpha)]\!]$. Then, it encrypts β as an SHE ciphertext $E(\beta)$.

• Furthermore, the query user randomly chooses two keys $\mathtt{TK}_1$ and $\mathtt{TK}_2$ for the symmetric-key encryption scheme. Then, it generates two ciphertexts, namely, $\mathtt{SE}(\mathtt{K}_1, \mathtt{TK}_1 \| ts \| uid)$ and $\mathtt{SE}(\mathtt{K}_2, \mathtt{TK}_2 \| ts \| uid)$, where ts is the current timestamp and uid is the identity of the query user.

• Finally, the query user submits the encrypted query token $[\![\mathsf{req}]\!] = \{[\![\mathcal{R}]\!], [\![\mathtt{ptr}(\alpha)]\!], E(\beta), \mathtt{SE}(\mathtt{K}_1, \mathtt{TK}_1 \| ts \| uid), \mathtt{SE}(\mathtt{K}_2, \mathtt{TK}_2 \| ts \| uid)\}$ to CS_1.

4.2.4 Query Conducting

Upon receiving an encrypted query request $[\![\mathsf{req}]\!]$, the two cloud servers conduct the (α, β)-core query over the encrypted index. Before detailing this phase, we first introduce three basic operators of the two cloud servers.

OP1: Row/Column Retrieving. In this scheme, there are three types of encrypted pointers $[\![\mathtt{ptr}(n)]\!]$, $[\![\mathtt{ptr}(e_{i,j})]\!]$, and $[\![\mathtt{ptr}(\alpha)]\!]$ pointing to a row in NodeTable, a row in EdgeTable, and a group of columns in both tables, respectively. We take $[\![\mathtt{ptr}(\alpha)]\!]$ and the columns $n.E(\widehat{\beta}_{\alpha_k})$ in NodeTable as an example and show how CS_1 retrieves the column $n.E(\widehat{\beta}_{\alpha_k})$ with $\alpha_k = \alpha$ from all α_{max} columns with the following two steps.

• First, by running Alg. 2, it obtains α_{max} SHE ciphertexts $E(\alpha_k = \alpha)$, which will be $E(1)$ if $\alpha_k = \alpha$, and $E(0)$ otherwise. In the algorithm, CS_1 compares each α_k and α in a bit-wise manner. If a pair of corresponding bits of α_k and α are not equal, the intermediate comparison output will be $E(0)$; otherwise, it will be $E(1)$ or $E(-1)$ as shown in Alg. 2. After that, CS_1 combines the comparison results through multiplication and multiplies an extra $E(-1)$ if needed, such that the overall result is $E(0)$ if any of the intermediate comparison results is $E(0)$, and $E(1)$ otherwise.

Algorithm 2 Computing $E(x = y)$

Input: a plaintext x and an array of ciphertexts $[\![y]\!] = \{[\![y]\!][i]\}_{i=0}^{[\![y]\!].\mathtt{len}}$
Output: an encrypted flag $E(x = y)$

```
acc = 1
ctr = 0                                  ▷ A counter for the number of '0' bits in x
for i = 0; i < [[y]].len; i++ do
    if x & (1 ≪ i) > 0 then              ▷ Determine whether the i-th bit is '0'
        val = [[y]][i]                   ▷ [[y]][i]= 1 if the bits are equal
    else
        val = [[y]][i] −1                ▷ [[y]][i]−1 = 1 if the bits are equal
        ctr ++
    acc = acc * val mod N
if ctr.isOdd() then                      ▷ If x has an odd number of '0' bits
    acc *= E(−1)
return acc
```

• After that, CS_1 computes Eq. (2) on each row. In the equation, each cell $n.E(\widehat{\beta}_{\alpha_k})$ will be multiplied with the corresponding $E(\alpha_k = \alpha)$ and added together. Thereby, those $n.E(\widehat{\beta}_{\alpha_k})$ linked to $\alpha_k \neq \alpha$ are eliminated.

$$n.E(\widehat{\beta}) = \sum_1^{\alpha_{max}} n.E(\widehat{\beta}_{\alpha_k}) \times E(\alpha_k = \alpha) \bmod \mathcal{N} \quad (2)$$

OP2: Visited Vector Updating. As shown in Alg. 1, during traversing, CS_1 needs to test whether a vertex has been visited and whether an edge has been added to the result set, and proceed if not. To this end, it needs to maintain two encrypted vectors $\overline{\mathsf{V}}_{\mathtt{n}}$ and $\overline{\mathsf{V}}_{\mathtt{e}}$, where each encrypted bit 0 represents a visited edge or vertex. After each **OP1** operation on rows in NodeTable and EdgeTable, CS_1 correspondingly updates the vectors $\overline{\mathsf{V}}_{\mathtt{n}}$ and $\overline{\mathsf{V}}_{\mathtt{e}}$.

We take NodeTable as an example. Initially, CS_1 allocates $\overline{\mathsf{V}}_{\mathtt{n}}$ as an array of 1's and $|\overline{\mathsf{V}}_{\mathtt{n}}| = |\mathcal{U} \cup \mathcal{V}|$, indicating all vertices have not been visited. Then, after running an **OP1** operation to retrieve the row pointed by $\mathtt{ptr}(n)$, CS_1 obtains

$E(\texttt{ptr}(n) = j)$ for the j-th row in the table, CS_1 updates $\overline{\mathsf{V}}_\mathtt{n}[j]$ by computing

$$\overline{\mathsf{V}}_\mathtt{n}[i] \leftarrow \overline{\mathsf{V}}_\mathtt{n}[i] \cdot (1 + E(\texttt{ptr}(n) = i) \cdot E(-1)) \bmod \mathcal{N}.$$

Then, if a vertex $n \in (\mathcal{U} \cup \mathcal{V})$ is visited, $\overline{\mathsf{V}}_\mathtt{n}[\texttt{ptr}(n)]$ will be an SHE ciphertext $E(0)$; otherwise, it will be $E(1)$.

OP3: Bootstrapping. As analyzed in Section 2.2, the SHE scheme supports a limited number of homomorphic multiplications between ciphertexts. Therefore, once an SHE ciphertext $c = E(m)$ is about to exceed the limitation δ, CS_1 runs this operation with CS_2 to obtain $c^* = E(m)$. Specifically, CS_1 and CS_2 run the following steps.

• First, CS_1 homomorphically computes $\widetilde{c} = E(m + r)$, where $r \in \mathcal{M}$ is a random number. Then, it sends $\widetilde{c}$ to CS_2.

• On receiving $\widetilde{c}$, CS_2 decrypts it and gets $\widetilde{m} = m + r$. With SK, CS_2 generates another ciphertext $\widetilde{c}^* = E(\widetilde{m})$ and sends it to CS_1.

• Finally, CS_1 computes $c^* = \widetilde{c} - r$ as a refreshed ciphertext of the message m.

Based on **OP1**, **OP2** and **OP3**, the two cloud servers conduct the query in the following phases.

Phase 1: Token Verification. The two cloud servers verify the query token and prepare for conducting the (α, β)-core query. Specifically, they run the following steps.

1) CS_1 decrypts $\texttt{SE}(\texttt{K}_1, \texttt{TK}_1 \| ts \| uid)$ with $\texttt{K}_1$ and gets $\texttt{TK}_1$, ts, and uid. If the timestamp ts is fresh, CS_1 keeps $\texttt{TK}_1$ and sends $\texttt{SE}(\texttt{K}_2, \texttt{TK}_2 \| ts \| uid)$ to CS_2. Otherwise, it stops the query process.

2) On receiving $\texttt{SE}(\texttt{K}_2, \texttt{TK}_2 \| ts \| uid)$, CS_2 decrypts it and gets $\texttt{TK}_2$, ts, and uid. Similarly, it verifies the freshness of ts and keeps $\texttt{TK}_2$.

3) CS_1 excludes the columns in NodeTable and EdgeTable that link to $\alpha_k \neq \alpha$ by running **OP1**, and we denote the resulting tables as $[\![\mathsf{NodeTable}]\!][\alpha]$ and $[\![\mathsf{EdgeTable}]\!][\alpha]$.

Phase 2: Queue Initialization. CS_1 and CS_2 run the following steps to initialize a queue $\texttt{Q}$ based on the encrypted query request.

1) CS_1 initializes an empty queue $\texttt{Q}$. Then, it constructs two visited vectors for **OP2**, namely, a vector $\overline{\mathsf{V}}_\mathtt{n}$ of length $|\mathcal{U} \cup \mathcal{V}|$ and a vector $\overline{\mathsf{V}}_\mathtt{e}$ of length $|\mathcal{E}|$. Elements in both vectors are initialized to be 1.

2) For each $([\![\texttt{ptr}(n)]\!], \texttt{flag}_n) \in [\![\mathcal{R}]\!]$, the two servers jointly run Steps (3)–(6).

3) By running **OP1**, CS_1 retrieves the corresponding row of n in $[\![\mathsf{NodeTable}]\!][\alpha]$ and gets $n.E(\widehat{\beta}_\alpha)$ and $n.[\![\texttt{head}_\alpha]\!]$. Based on the property of **OP1**, CS_1 will not know the location of each $n \in \mathcal{R}$ in the table $[\![\mathsf{NodeTable}]\!][\alpha]$.

4) CS_1 randomly generates a bit $b \in \{0, 1\}$. Then, based on b, CS_1 homomorphically computes $E(\widetilde{x})$ through Eq. (3), where r_1 and r_2 are two random integers satisfying $0 < r_2 < r_1$ and $r_1 \in \{0, 1\}^{k_1 - 1}$. Then, it sends $E(\widetilde{x})$ to CS_2.

$$E(\widetilde{x}) = \begin{cases} r_1 \cdot (n.E(\widehat{\beta}_\alpha) + E(-1) \cdot E(\beta)) + r_2 \bmod \mathcal{N}, & b = 1, \\ r_1 \cdot (E(\beta) + E(-1) \cdot n.E(\widehat{\beta}_\alpha)) - r_2 \bmod \mathcal{N}, & b = 0. \end{cases} \quad (3)$$

5) On receiving $E(\widetilde{x})$, CS_2 decrypts it with SK and gets $\widetilde{x}$. After that, it compares $\widetilde{x}$ with $\mathcal{L}/2$. Based on the comparison result, it sends $\sigma = 1$ to CS_1 if $\widetilde{x} < \mathcal{L}/2$, and it sends $\sigma = 0$ otherwise.

6) If $\sigma = b$, indicating that $n.\widehat{\beta} \geq \beta$, CS_1 enqueues $(n.[\![\texttt{head}_\alpha]\!], \texttt{flag}_n)$ into $\texttt{Q}$, where the first field points to an edge $e_{i,j}$ in EdgeTable, and the second field $\texttt{flag}_n$ indicates whether this vertex $n \in \mathcal{U}$ or not.

Phase 3: Index Traversing. CS_1 initializes an empty array $\texttt{Edges}$. Then, it jointly executes the query with CS_2 by traversing the index with the help of the queue $\texttt{Q}$. They run the following steps on each tuple $([\![\texttt{ptr}(e_{i,j})]\!], \texttt{flag}_n)$ dequeued from $\texttt{Q}$ until $\texttt{Q}$ is empty.

1) By running **OP1** over $[\![\mathsf{EdgeTable}]\!][\alpha]$, CS_1 obliviously retrieves the row corresponding to $e_{i,j}$, and it obtains $[\![\texttt{ptr}(u_i)]\!]$, $[\![\texttt{ptr}(v_j)]\!]$, $[\![\texttt{next}_\mathtt{u}]\!]$, and $[\![\texttt{next}_\mathtt{v}]\!]$ of $e_{i,j}$.

In the following steps, we use n to denote the vertex under traversing and use n' to denote the current visiting neighbor of n as shown in Alg. 1. Specifically, when $\texttt{flag}_u = 1$, we set $n = u_i$ and $n' = v_j$; otherwise, we set $n = v_j$ and $n' = u_i$.

2) CS_1 runs **OP1** with the encrypted pointer $[\![\texttt{ptr}(n')]\!]$ on $[\![\mathsf{NodeTable}]\!][\alpha]$ to obliviously retrieve the corresponding $n'.E(\widehat{\beta}_\alpha)$ and $n'.[\![\texttt{head}_\alpha]\!]$.

3) By running Steps 4–5 in **Phase 2** with CS_2, the cloud server CS_1 securely compares $n'.E(\widehat{\beta}_\alpha)$ and $E(\beta)$. If $n'.\widehat{\beta}_\alpha < \beta$, the two servers skip the following steps and proceed to the next iteration. Otherwise, CS_1 enqueues $e_{i,j}.[\![\texttt{next}_{\mathtt{n}',\alpha}]\!]$ to $\texttt{Q}$.

4) With $[\![\texttt{ptr}(n')]\!]$ and $[\![\texttt{ptr}(e_{i,j})]\!]$, CS_1 runs **OP1** to retrieve $\overline{\mathsf{V}}_\mathtt{n}[\texttt{ptr}(n')]$ and $\overline{\mathsf{V}}_\mathtt{e}[\texttt{ptr}(e_{i,j})]$. After that, it runs **OP2** to respectively set $\overline{\mathsf{V}}_\mathtt{n}[\texttt{ptr}(n')]$ and $\overline{\mathsf{V}}_\mathtt{e}[\texttt{ptr}(e_{i,j})]$ to be $E(0)$. During this step, CS_1 does not know which locations in $\overline{\mathsf{V}}_\mathtt{n}$ and $\overline{\mathsf{V}}_\mathtt{e}$ have been accessed.

5) CS_1 computes $E(\widetilde{\mathsf{v}}_{n'}) = \overline{\mathsf{V}}_\mathtt{n}[\texttt{ptr}(n')] + r_1 \bmod \mathcal{N}$ and $E(\widetilde{\mathsf{v}}_e) = \overline{\mathsf{V}}_\mathtt{e}[\texttt{ptr}(e_{i,j})] + r_2 \bmod \mathcal{N}$, where r_1 and $r_2 \in \mathcal{M}$ are two random numbers chosen independently. Then, it sends $E(\widetilde{\mathsf{v}}_{n'})$ and $E(\widetilde{\mathsf{v}}_e)$ to CS_2.

6) On receiving $E(\widetilde{\mathsf{v}}_{n'})$ and $E(\widetilde{\mathsf{v}}_e)$, CS_2 decrypts them and returns $\widetilde{\mathsf{v}}_{n'}$ and $\widetilde{\mathsf{v}}_e$ to CS_1.

7) CS_1 further computes $\bar{\mathsf{v}}_{n'} = \widetilde{\mathsf{v}}_{n'} - r_1 \bmod \mathcal{N}$ and $\bar{\mathsf{v}}_e = \widetilde{\mathsf{v}}_e - r_2 \bmod \mathcal{N}$. If $\bar{\mathsf{v}}_{n'} = 1$, i.e., the vertex n' has not been visited, CS_1 enqueues $(n'.[\![\texttt{head}_\alpha]\!], 1 - \texttt{flag}_n)$ into $\texttt{Q}$. If $\bar{\mathsf{v}}_e = 1$ indicating that the edge $e_{i,j} = (u_i \rightarrow v_j)$ has not been added to $\texttt{Edges}$, then CS_1 appends $([\![\texttt{ptr}(u_i)]\!], [\![\texttt{ptr}(v_i)]\!])$ to $\texttt{Edges}$.

Phase 4: Result Re-encryption. CS_1 and CS_2 build the query response from $\texttt{Edges}$. For each tuple $([\![\texttt{ptr}(u_i)]\!], [\![\texttt{ptr}(v_j)]\!]) \in \texttt{Edges}$, they run the following steps.

1) CS_1 aggregates the encrypted pointer into one SHE ciphertext by computing

$$E(\texttt{ptr}(u_i)) = \sum\nolimits_{t=0}^{\ell_\mathtt{n} - 1} [\![\texttt{ptr}(u_i)]\!][t] \cdot 2^t \bmod \mathcal{N},$$

where $\ell_\mathtt{n} = \lceil \log_2(|\mathcal{U} \cup \mathcal{V}| + 1) \rceil$, and $[\![\texttt{ptr}(u_i)]\!][k]$ represents the k-th element in $[\![\texttt{ptr}(u_i)]\!]$. Similarly, it computes $E(\texttt{ptr}(v_j))$ from $[\![\texttt{ptr}(v_j)]\!]$.

2) CS_1 homomorphically computes $E(\widetilde{e}_{i,j}) = E(\bar{e}_{i,j} + r_3)$, where $\bar{e}_{i,j} = u_i \cdot (|\mathcal{V}| + 1) + v_j$, and $r_3 \in \mathcal{M}$ is a random number. After that, it sends $E(\widetilde{e}_{i,j})$ to CS_2.

3) CS_2 decrypts the received $E(\widetilde{e}_{i,j})$ and gets $\widetilde{e}_{i,j}$. Then, CS_2 encrypts it with $\texttt{TK}_2$, i.e., $\texttt{SE}(\texttt{TK}_2, \widetilde{e}_{i,j})$, and sends the resulting ciphertext to CS_1.

4) CS_1 encrypts r_3 with $\texttt{TK}_1$, i.e., $\texttt{SE}(\texttt{TK}_1, r_3)$. Then, it sends $(\texttt{SE}(\texttt{TK}_1, r_3), \texttt{SE}(\texttt{TK}_2, \widetilde{e}_{i,j}))$ to the query user.

By decrypting these edges from CS_1, the query user can reconstruct the returned (α, β)-core $\mathcal{G}_\mathsf{req} = \{\mathcal{U}_\mathsf{req}, \mathcal{V}_\mathsf{req}, \mathcal{E}_\mathsf{req}\}$.

Specifically, for each $(\mathtt{SE}(\mathtt{TK}_1, r_3), \mathtt{SE}(\mathtt{TK}_2, \widetilde{e}_{i,j}))$ from CS_1, the query user can decrypt them with $\mathtt{TK}_1$ and $\mathtt{TK}_2$ and get $(r_3, \widetilde{e}_{i,j})$. Then, he/she can compute $\bar{e}_{i,j} = \widetilde{e}_{i,j} - r_3 = u_i \cdot (|\mathcal{V}| + 1) + v_j$ and further extract $u_i = \lfloor \bar{e}_{i,j}/(|\mathcal{V}| + 1) \rfloor$ and $v_j = \bar{e}_{i,j} \bmod (|\mathcal{V}| + 1)$. Finally, the query user updates $\mathcal{G}_{\mathsf{req}}$ by computing $\mathcal{U}_{\mathsf{req}} \leftarrow \mathcal{U}_{\mathsf{req}} \cup \{u_i\}$, $\mathcal{V}_{\mathsf{req}} \leftarrow \mathcal{V}_{\mathsf{req}} \cup \{v_j\}$, and $\mathcal{E}_{\mathsf{req}} \leftarrow \mathcal{E}_{\mathsf{req}} \cup \{u_i \rightarrow v_j\}$.

4.3 Security Analysis for the Basic Scheme

In this subsection, we analyze the security of our basic scheme. As proved in [34], the ciphertexts obtained through Eq. (1) are IND-CPA secure. Based on this, we recall the definition of security model for securely implementing an ideal functionality with semi-honest adversaries in [35] and prove the two *honest-but-curious* cloud servers CS_1 and CS_2 cannot obtain 1) the plaintext and structure of the dataset (the bipartite graph); 2) the plaintext of the encrypted query requests and the corresponding results; and 3) the access pattern, i.e., the vertices and edges visited during executing queries. The security model consists of a real model and an ideal model as follows.

Real Model. The real-world execution of our basic scheme Π_B happens between the two servers $\{\mathsf{CS}_1, \mathsf{CS}_2\}$, and $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ is the pair of adversaries controlling the two servers, respectively. Assume that the inputs of CS_1 and CS_2 are respectively $x = \{\mathtt{K}_1, [\![\mathsf{EdgeTable}]\!], [\![\mathsf{NodeTable}]\!], [\![\mathsf{req}]\!]\}$ and $y = \{SK, \mathtt{K}_2\}$, and z is auxiliary input, e.g., public parameters. Then, with the inputs x, y, and z, the execution of Π_B under $\mathcal{A}$ in the real model can be defined as

$$\mathrm{REAL}_{\Pi_B, \mathcal{A}, z}(x, y) \stackrel{def}{=} \{\mathrm{Output}^{\Pi_B}(x, y), \mathrm{View}^{\Pi_B}(x, y), z\},$$

in which pairs $\mathrm{Output}^{\Pi_B}(x, y) = \{\mathrm{Output}_i^{\Pi_B}(x, y)\}_{i=1}^2$ and $\mathrm{View}^{\Pi_B}(x, y) = \{\mathrm{View}_i^{\Pi_B}(x, y)\}_{i=1}^2$ are the outputs and views of $\{\mathsf{CS}_i\}_{i=1}^2$ during the execution, respectively.

Ideal Model. In the ideal-world execution, there is a pair of ideal functionalities $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2\}$ respectively implementing the computation of the two servers, denoted as f_1 and f_2, and each cloud server CS_i interacts with the corresponding ideal functionality $\mathcal{F}_i$. Here, the execution of $f = (f_1, f_2)$ under simulators $\mathtt{Sim} = \{\mathtt{Sim}_1, \mathtt{Sim}_2\}$ in the ideal-world model on input pair (x, y) and auxiliary input z is defined as

$$\mathrm{IDEAL}_{\mathcal{F}, \mathtt{Sim}, z}(x, y) \stackrel{def}{=} \left\{ \begin{matrix} f_1(x, y), \mathtt{Sim}_1(x, f_1(x, y)), \\ f_2(x, y), \mathtt{Sim}_2(y, f_2(x, y)), z \end{matrix} \right\}.$$

Definition 2 (Security of Π_B against semi-honest adversary)**.** *Let $\mathcal{F}$ be a deterministic functionality and Π_B a protocol between the two servers. We say that Π_B securely realizes $\mathcal{F}$ if there exists $\mathtt{Sim}$ of probabilistic polynomial-time (PPT) transformations (where $\mathtt{Sim} = \mathtt{Sim}(\mathcal{A})$) such that for semi-honest PPT adversaries $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$, for x, y and z, for $\{\mathsf{CS}_1, \mathsf{CS}_2\}$ holds:*

$$\mathrm{REAL}_{\Pi_B, \mathcal{A}, z}(x, y) \stackrel{c}{\approx} \mathrm{IDEAL}_{\mathcal{F}, \mathtt{Sim}, z}(x, y),$$

where $\stackrel{c}{\approx}$ compactly denotes computational indistinguishability.

Next, we show that our basic scheme can protect the plaintext and structure of the dataset, the plaintext of query requests and results, and the access pattern.

Theorem 1. *The basic scheme can protect the plaintext and structure of the original bipartite graph $\mathcal{G}$, the plaintext of query requests and results, and the access pattern against the adversaries $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$.*

Proof. Here, we show how to construct the simulators $\mathtt{Sim} = \{\mathtt{Sim}_1, \mathtt{Sim}_2\}$. Given $\{x, f_1(x, y), z\}$, where $f_1(x, y) = \{\{\sigma = b\}, \{\bar{\mathtt{v}}_{n'}\}, \{\bar{\mathtt{v}}_e\}, |\mathtt{Edges}|\}$ denotes the output of CS_1 while conducting a query, $\mathtt{Sim}_1$ simulates $\mathcal{A}_1$ as follows: i) $\mathtt{Sim}_1$ randomly chooses $\{\mathtt{SE}(\mathtt{TK}_2', \widetilde{e}_{i,j})\}'$, an array of $|\mathtt{Edges}|$ SE ciphertexts corresponding to a random secret key $\mathtt{TK}_2'$; ii) it outputs $\mathcal{A}_1$'s view. In addition to x and $f_1(x, y)$, $\mathcal{A}_1$ receives $\{\mathtt{SE}(\mathtt{TK}_2, \widetilde{e}_{i,j})\}$ in the real execution, while $\mathtt{Sim}_1$ outputs $\{\mathtt{SE}(\mathtt{TK}_2', \widetilde{e}_{i,j})\}'$ during the ideal execution. Since $\mathtt{TK}_2$ is randomly generated, based on the security of the underlying SE scheme, the views of $\mathcal{A}_1$ in the two executions are indistinguishable. On the other hand, given $f_2(x, y) = \{|\{\widetilde{\mathtt{v}}_{n'}\}|, |\{\widetilde{\mathtt{v}}_e\}|, |\mathtt{Edges}|\}$, $\mathtt{Sim}_2$ simulates $\mathcal{A}_2$ as follows: i) it respectively chooses four vectors of random plaintexts for the SHE scheme of lengths $|\{\widetilde{\mathtt{v}}_e\}|$, $|\mathtt{Edges}|$, $|\{\widetilde{\mathtt{v}}_{n'}\}|$, and $|\{\widetilde{\mathtt{v}}_e\}|$, denoted by $\{\widetilde{x}\}'$, $\{\widetilde{e}_{i,j}\}'$, $\{\widetilde{\mathtt{v}}_{n'}\}'$, and $\{\widetilde{\mathtt{v}}_e\}'$; ii) it constructs the view of $\mathcal{A}_2$. In addition to x and $f_2(x, y)$, $\mathcal{A}_2$ receives $\{\{\widetilde{x}\}, \{\widetilde{e}_{i,j}\}, \{\widetilde{\mathtt{v}}_{n'}\}, \{\widetilde{\mathtt{v}}_e\}\}$ during the real execution, while $\mathtt{Sim}_2$ outputs $\{\{\widetilde{x}\}', \{\widetilde{e}_{i,j}\}', \{\widetilde{\mathtt{v}}_{n'}\}', \{\widetilde{\mathtt{v}}_e\}'\}$ during the ideal execution. Since all SHE plaintexts received by $\mathcal{A}_2$ are added with random numbers by CS_1, the views of $\mathcal{A}_2$ in the two executions are indistinguishable.

Thus, the adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ obtain nothing other than $\{x, f_1(x, y) = \{\{\sigma = b\}, \{\bar{\mathtt{v}}_{n'}\}, \{\bar{\mathtt{v}}_e\}, |\mathtt{Edges}|\}, z\}$ and $\{y, f_2(x, y) = \{|\{\widetilde{v}_{n'}\}|, |\{\widetilde{v}_e\}|, |\mathtt{Edges}|\}\}$, respectively. Hence, $\mathcal{A}_1$ and $\mathcal{A}_2$ cannot derive any useful information related to i) the plaintext and structure of of the original bipartite graph $\mathcal{G}$; ii) the plaintext of the query requests and results; and iii) which of the vertices and edges are visited, i.e., the access pattern. □

5 Our Security-Enhanced Query Scheme

In this section, to further hide the structure of the query requests and results (i.e., $|\mathcal{U} \cap \mathcal{R}|$, $|\mathcal{V} \cap \mathcal{R}|$, $|\mathcal{U}_{\mathsf{req}}|$, $|\mathcal{V}_{\mathsf{req}}|$, and $\{deg(n) \mid \forall n \in \mathcal{U}_{\mathsf{req}} \cup \mathcal{V}_{\mathsf{req}}\}$), we propose a scheme to enhance the security of the basic scheme. After that, we present the security analysis of the resulting security-enhanced scheme.

5.1 Description of Our Security-Enhanced Scheme

Compare to the basic scheme, the security-enhanced one has exactly the same *System Initialization* algorithm. As for the *Query Encryption* algorithm, the only difference is that the encrypted set of query vertices $[\![\mathcal{R}]\!] = \{[\![\mathtt{ptr}(n)]\!] \mid n \in \mathcal{R} \subseteq (\mathcal{U} \cup \mathcal{V})\}$, i.e., the $\mathtt{flag}_{\mathtt{n}}$s in that of the basic scheme are removed as they will reveal the structure of the query requests. Hence, we present its *Graph Outsourcing* and *Query Conducting* algorithms as follows.

5.1.1 Graph Outsourcing

After running *System Initialization*, DO encrypts the bipartite graph $\mathcal{G}$ with PP and SK by running the following steps.

- First, DO builds the table $\mathcal{T}$ containing $\widehat{\beta}_{n,\alpha_k}$ for each vertex $n \in (\mathcal{U} \cup \mathcal{V})$ and all possible $\alpha_k = 1, \cdots, \alpha_{max}$.
- Then, DO builds two tables $\mathsf{NodeTable}^*$ and $\mathsf{EdgeTable}^*$ in the following steps.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3169386, IEEE Transactions on Dependable and Secure Computing

Index	head_1	...
1	1	
2	2	
3	5	...
4	7	
5	8	
6	11	
7	1	
8	3	
9	4	...
10	6	
11	10	

(a) NodeTable*

Index	u_i	v_j	$\alpha_1 = 1$						...
			head_u	next_u	$\widehat{\beta}_u$	head_v	next_v	$\widehat{\beta}_v$	...
1	1	7	0	0	4	0	2	4	
2	2	7	0	3	4	1	5	4	
3	2	8	2	4	4	0	8	2	
4	2	9	2	0	4	0	11	2	
5	3	7	0	6	4	1	7	4	
6	3	10	5	0	4	0	9	2	...
7	4	7	0	0	4	1	0	4	
8	5	8	0	9	2	3	0	2	
9	5	10	8	10	2	6	0	2	
10	5	11	8	0	2	0	0	1	
11	6	9	0	0	2	4	0	2	

(b) EdgeTable*

Fig. 5. The two tables NodeTable* and EdgeTable* built from the linked lists $\{L_{n,\alpha} \mid n \in (\mathcal{U} \cup \mathcal{V}), \alpha = 1, \cdots, \alpha_{max}\}$.

1) DO first constructs the linked lists $\{L_{n,\alpha_k} \mid n \in (\mathcal{U} \cup \mathcal{V}), \alpha_k = 1, \cdots, \alpha_{max}\}$.

2) DO builds a table NodeTable* where each row represents a vertex $n \in (\mathcal{U} \cup \mathcal{V})$, and the table has α_{max} columns. For each $\alpha_k = 1, \cdots, \alpha_{max}$, the table records the header of L_{n,α_k} for each vertex n, denoted by $n.\texttt{head}_{\alpha_k}$.

3) DO further builds a table EdgeTable* where each row represents an edge $e_{i,j} = (u_i \in \mathcal{U}, v_j \in \mathcal{V}) \in \mathcal{E}$, and the table has $2 + 6 \times \alpha_{max}$ columns. In addition to two columns u_i and v_j, there are α_{max} groups of columns. For the α_k-th group, there are 6 columns, namely, $\texttt{head}_{\texttt{u},\alpha_k}$ and $\texttt{head}_{\texttt{v},\alpha_k}$ representing the headers of L_{u_i,α_k} and L_{v_j,α_k}, $\texttt{next}_{\texttt{u},\alpha_k}$ and $\texttt{next}_{\texttt{v},\alpha_k}$ representing the next edges after $e_{i,j}$ in the two linked lists, and $\widehat{\beta}_{\texttt{u},\alpha_k}$ and $\widehat{\beta}_{\texttt{v},\alpha_k}$ representing $\widehat{\beta}_{u_i,\alpha_k}$ and $\widehat{\beta}_{v_j,\alpha_k}$. Note that, for $e_{i,j}.\texttt{head}_{\texttt{u},\alpha}$ and $e_{i,j}.\texttt{head}_{\texttt{v},\alpha}$, they will be set to 0 if they are equal to the edge's index. For example, as shown in Fig. 5, $L_{v_2,1} = 3$ (the 8-th line in NodeTable*), so we have $e_{2,2}.\texttt{head}_{\texttt{v},\alpha} = 0$ in the third edge in EdgeTable*.

• After that, DO encrypts NodeTable* and EdgeTable* into $[\![\text{NodeTable*}]\!]$ and $[\![\text{EdgeTable*}]\!]$. Specifically, he/she runs the following steps.

1) For each $n.\texttt{head}_{\alpha_k}$ in NodeTable*, DO encrypts it into $n.[\![\texttt{head}_{\alpha_k}]\!]$. Similarly, he/she encrypts $\texttt{head}_{\texttt{u},\alpha}$, $\texttt{head}_{\texttt{v},\alpha}$, $\texttt{next}_{\texttt{u},\alpha}$ and $\texttt{next}_{\texttt{v},\alpha}$ of EdgeTable* and gets $[\![\texttt{head}_{\texttt{u},\alpha}]\!]$, $[\![\texttt{head}_{\texttt{v},\alpha}]\!]$, $[\![\texttt{next}_{\texttt{u},\alpha}]\!]$ and $[\![\texttt{next}_{\texttt{v},\alpha}]\!]$.

2) For each $\texttt{head}_{\texttt{u},\alpha}$ in EdgeTable*, DO further generates a ciphertext $E(\texttt{head}_{\texttt{u},\alpha} \neq 0)$, which will be $E(1)$ if $\texttt{head}_{\texttt{u},\alpha} \neq 0$, and it will be $E(0)$ otherwise. Similarly, DO computes $E(\texttt{head}_{\texttt{v},\alpha} \neq 0)$, $E(\texttt{next}_{\texttt{u},\alpha} \neq 0)$, and $E(\texttt{next}_{\texttt{v},\alpha} \neq 0)$ for EdgeTable*.

3) For $\widehat{\beta}_{u_i,\alpha_k}$ and $\widehat{\beta}_{v_j,\alpha_k}$, DO respectively encrypts them into SHE ciphertexts $E(\widehat{\beta}_{\texttt{u},\alpha_k})$ and $E(\widehat{\beta}_{\texttt{v},\alpha_k})$.

4) Furthermore, DO computes $\bar{e}_{i,j} = u_i \times (|\mathcal{V}| + 1) + v_j$ and encrypts it into an SHE ciphertext $E(\bar{e}_{i,j})$.

At the end of the algorithm, DO submits $[\![\text{NodeTable*}]\!]$ and $[\![\text{EdgeTable*}]\!]$ to the cloud server CS_1.

5.1.2 Query Conducting

With the two tables, the two cloud servers $\{\text{CS}_1, \text{CS}_2\}$ can jointly answer encrypted (α, β)-core queries from query users. Before delving into the details, we first introduce an operator for the cloud server CS_1.

OP4: Encrypted Queue. To enhance the privacy of the basic scheme, we replace the queue Q in Section 4.2.4 with an encrypted one, denoted by $[\![\texttt{Q}]\!]$, which can insert an element if an encrypted flag $E(\texttt{enable})$ is $E(1)$, and ignore the insertion if $E(\texttt{enable})$ is $E(0)$. In this way, $[\![\texttt{Q}]\!]$ can hide whether an insertion to the queue actually inserts an element or not. $[\![\texttt{Q}]\!]$ comprises two components, namely, $[\![\texttt{Q}]\!].\texttt{arr}$ and $[\![\texttt{Q}]\!].\texttt{tail}$. On the one hand, $[\![\texttt{Q}]\!].\texttt{arr}$ is an array of encrypted elements in $[\![\texttt{Q}]\!]$, and it is initialized to be an empty array. On the other hand, $[\![\texttt{Q}]\!].\texttt{tail}$ is an array containing $[\![\texttt{Q}]\!].\texttt{arr}$.len() SHE ciphertexts of 0 and one ciphertext of 1, and the location $E(1)$ in $[\![\texttt{Q}]\!].\texttt{tail}$ indicates the first empty space in $[\![\texttt{Q}]\!].\texttt{arr}$. For example, after several enqueue and dequeue operations, a possible status of $[\![\texttt{Q}]\!]$ might be $[\![\texttt{Q}]\!].\texttt{arr} = \{E(39), E(11), E(6), E(0), E(0)\}$, and $[\![\texttt{Q}]\!].\texttt{tail} = \{E(0), E(0), E(0), E(1), E(0), E(0)\}$.

Algorithm 3 Operations on the encrypted queue $[\![\texttt{Q}]\!]$

```
1:  procedure ENQUEUE([[Q]], E(elem), E(enable))
2:      E(elem) = E(elem) × E(enable)
3:      append([[Q]].arr, 0)
4:      append([[Q]].tail, 0)
5:      for i = 0, ··· , [[Q]].arr.len()−1 do
6:          [[Q]].arr[i] = [[Q]].arr[i] + [[Q]].tail[i] × E(elem) mod N
7:      for i = [[Q]].tail.len()−1, ··· , 0 do
8:          if i ≠ 0 then
9:              prev = [[Q]].tail[i − 1]
10:         else
11:             prev = 0
12:         curr = [[Q]].tail[i]
13:         [[Q]].tail[i] = E(enable) × (prev − curr) + curr mod N

14: procedure DEQUEUE([[Q]])
15:     E(elem) = [[Q]].arr.removeFirst()
16:     E(isEmpty) = [[Q]].tail.removeFirst()
17:     return (E(elem), E(isEmpty))
```

Based on these two components, $[\![\texttt{Q}]\!]$ can support ENQUEUE and DEQUEUE, as shown in Alg. 3. These two procedures respectively run as follows.

• ENQUEUE: While enqueueing an encrypted element $E(\texttt{elem})$, CS_1 first respectively appends a value 0 at the end of both $[\![\texttt{Q}]\!].\texttt{arr}$ and $[\![\texttt{Q}]\!].\texttt{tail}$. Then, it updates each $[\![\texttt{Q}]\!].\texttt{arr}[i]$ by computing

$$[\![\texttt{Q}]\!].\texttt{arr}[i] = [\![\texttt{Q}]\!].\texttt{arr}[i] + E(\texttt{elem}) \times E(\texttt{enable}) \times [\![\texttt{Q}]\!].\texttt{tail}[i] \bmod \mathcal{N}.$$

That is, the encrypted element will only be added to the location where $[\![\texttt{Q}]\!].\texttt{tail}[i] = E(1)$ and when $E(\texttt{enable}) = E(1)$. After that, the algorithm updates $[\![\texttt{Q}]\!].\texttt{tail}$ based on $E(\texttt{enable})$. If $E(\texttt{enable}) = E(1)$, the algorithm inserts an $E(0)$ at the beginning of $[\![\texttt{Q}]\!].\texttt{tail}$; otherwise, it appends an $E(0)$ at the end of $[\![\texttt{Q}]\!].\texttt{tail}$, as shown in Line 13 of the algorithm.

• DEQUEUE. While dequeueing an element $E(\texttt{elem})$ from $[\![\texttt{Q}]\!]$, CS_1 removes the first elements in $[\![\texttt{Q}]\!].\texttt{arr}$ and $[\![\texttt{Q}]\!].\texttt{tail}$, and respectively denotes these two elements as $E(\texttt{elem})$ and $E(\texttt{isEmpty})$. Then, it sends $E(\texttt{isEmpty})$ to CS_2, and the latter returns $\texttt{isEmpty}$. If $\texttt{isEmpty} = 1$, CS_1 knows that $[\![\texttt{Q}]\!]$ is empty. Otherwise, $E(\texttt{elem})$ is the encrypted element dequeued from $[\![\texttt{Q}]\!]$.

Based on **OP4**, along with the three operators defined in Section 4.2.4, CS_1 and CS_2 can conduct an (α, β)-core query in the following phases.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3169386, IEEE Transactions on Dependable and Secure Computing

Phase 1: Token Verification. Similar to **Phase 1** in Section 4.2.4, the two servers CS_1 and CS_2 verify the encrypted query token $[\![\mathtt{req}]\!] = \{[\![\mathcal{R}]\!], [\![\mathtt{ptr}(\alpha)]\!], E(\beta), \mathtt{SE}(\mathtt{K}_1, \mathtt{TK}_1 \| ts \| uid), \mathtt{SE}(\mathtt{K}_2, \mathtt{TK}_2 \| ts \| uid)\}$ and respectively obtain $\mathtt{TK}_1$ and $\mathtt{TK}_2$. Then, CS_1 obtains $[\![\mathsf{NodeTable}^*]\!][\alpha]$ and $[\![\mathsf{EdgeTable}^*]\!][\alpha]$ through **OP1**.

Phase 2: Queue Initialization. CS_1 initializes an empty encrypted queue $[\![\mathtt{Q}]\!]$ following **OP4**. Furthermore, it allocates two vectors $\overline{\mathsf{V}}_\mathtt{n}$ and $\overline{\mathsf{V}}_\mathtt{e}$ defined in **OP2**. Different from those in Section 4.2.4, both of them have the same length of $|\mathcal{E}|$. Then, for each element $[\![\mathtt{ptr}(n)]\!] \in [\![\mathcal{R}]\!]$, CS_1 runs the following two steps.

1) By running **OP1**, CS_1 retrieves the row in $[\![\mathsf{NodeTable}^*]\!][\alpha]$ corresponding to $[\![\mathtt{ptr}(n)]\!]$, and it gets $n.[\![\mathtt{head}_\alpha]\!]$.

2) CS_1 runs **OP4** to enqueue the obtained $n.[\![\mathtt{head}_\alpha]\!]$ into the encrypted queue $[\![\mathtt{Q}]\!]$ with $E(\mathtt{enable}) = 1$. Then, it runs **OP2** to set $\overline{\mathsf{V}}_\mathtt{n}[n.[\![\mathtt{head}_\alpha]\!]] = E(0)$.

Phase 3: Index Traversing. CS_1 and CS_2 jointly conduct the query by traversing the bipartite graph with the help of $[\![\mathtt{Q}]\!]$. Specifically, they run the following steps on each element $[\![\mathtt{ptr}(e_{i,j})]\!]$ dequeued from $[\![\mathtt{Q}]\!]$ until $[\![\mathtt{Q}]\!]$ is empty.

1) CS_1 retrieves the row corresponding to $[\![\mathtt{ptr}(e_{i,j})]\!]$ in $[\![\mathsf{EdgeTable}^*]\!][\alpha]$ through **OP1**, and it gets $E(\bar{e}_{i,j})$, $[\![\mathtt{head}_{\mathtt{u},\alpha}]\!]$, $[\![\mathtt{next}_{\mathtt{u},\alpha}]\!]$, $E(\widehat{\beta}_{\mathtt{u},\alpha})$, $[\![\mathtt{head}_{\mathtt{v},\alpha}]\!]$, $[\![\mathtt{next}_{\mathtt{v},\alpha}]\!]$, $E(\widehat{\beta}_{\mathtt{v},\alpha})$.

2) CS_1 randomly generates a bit $b \in \{0,1\}$, and two positive random integers r_1 and r_2 satisfying $0 < r_2 < r_1 < 2^{k_1-1}$. Then, it homomorphically computes $E(\widetilde{x}) = E(r_1(\widehat{\beta}_{\mathtt{u},\alpha} - \beta) + r_2)$ if $b = 0$, or $E(\widetilde{x}) = E(r_1(\beta - \widehat{\beta}_{\mathtt{u},\alpha}) - r_2)$, otherwise. After that, it sends $E(\widetilde{x})$ to CS_2.

3) On receiving $E(\widetilde{x})$, CS_2 decrypts it and gets $\widetilde{x}$. If $\widetilde{x} < \mathcal{L}/2$, CS_2 sets $E(\sigma) = E(1)$ and $E(\overline{\sigma}) = E(0)$; otherwise, it sets $E(\sigma) = E(0)$ and $E(\overline{\sigma}) = E(1)$. Then, it sends $(E(\sigma), E(\overline{\sigma}))$ to CS_1.

4) On receiving $(E(\sigma), E(\overline{\sigma}))$, if $b = 0$, CS_1 sets $E(\widehat{\beta}_{\mathtt{u},\alpha} \geq \beta) = E(\sigma)$ and $E(\widehat{\beta}_{\mathtt{u},\alpha} < \beta) = E(\overline{\sigma})$; otherwise, it sets $E(\widehat{\beta}_{\mathtt{u},\alpha} \geq \beta) = E(\overline{\sigma})$ and $E(\widehat{\beta}_{\mathtt{u},\alpha} < \beta) = E(\sigma)$. Similarly, CS_1 gets two encrypted flags $E(\widehat{\beta}_{\mathtt{v},\alpha} \geq \beta)$ and $E(\widehat{\beta}_{\mathtt{v},\alpha} < \beta)$.

5) CS_1 homomorphically computes an encrypted flag

$$E(\mathtt{enable}_1) = E(\widehat{\beta}_{\mathtt{u},\alpha} \geq \beta) E(\mathtt{head}_{\mathtt{u},\alpha} \neq 0) \overline{\mathsf{V}}_\mathtt{n}[\mathtt{head}_{\mathtt{u},\alpha}] \bmod \mathcal{N},$$

and it runs ENQUEUE($[\![\mathtt{Q}]\!]$, $[\![\mathtt{head}_{\mathtt{u},\alpha}]\!]$, $E(\mathtt{enable}_1)$) to try enqueueing $[\![\mathtt{head}_{\mathtt{u},\alpha}]\!]$ into $[\![\mathtt{Q}]\!]$. Based on the property of **OP4**, $[\![\mathtt{head}_{\mathtt{u},\alpha}]\!]$ will be enqueued if $E(\mathtt{enable}_1)$ is $E(1)$, which indicates that i) $\widehat{\beta}_{\mathtt{u},\alpha} \geq \beta$; ii) $\mathtt{head}_{\mathtt{u},\alpha}$ is not empty; and iii) $\mathtt{head}_{\mathtt{u},\alpha}$ has not been visited.

6) Similarly, CS_1 computes three encrypted flags

$$E(\mathtt{enable}_2) = E(\widehat{\beta}_{\mathtt{u},\alpha} \geq \beta) E(\mathtt{next}_{\mathtt{u},\alpha} \neq 0) \overline{\mathsf{V}}_\mathtt{n}[\mathtt{next}_{\mathtt{u},\alpha}] \bmod \mathcal{N},$$
$$E(\mathtt{enable}_3) = E(\widehat{\beta}_{\mathtt{v},\alpha} \geq \beta) E(\mathtt{head}_{\mathtt{v},\alpha} \neq 0) \overline{\mathsf{V}}_\mathtt{n}[\mathtt{head}_{\mathtt{v},\alpha}] \bmod \mathcal{N},$$
$$E(\mathtt{enable}_4) = E(\widehat{\beta}_{\mathtt{v},\alpha} \geq \beta) E(\mathtt{next}_{\mathtt{v},\alpha} \neq 0) \overline{\mathsf{V}}_\mathtt{n}[\mathtt{next}_{\mathtt{v},\alpha}] \bmod \mathcal{N}.$$

Based on these flags, CS_1 respectively runs ENQUEUE to try enqueueing $[\![\mathtt{next}_{\mathtt{u},\alpha}]\!]$, $[\![\mathtt{head}_{\mathtt{v},\alpha}]\!]$, and $[\![\mathtt{next}_{\mathtt{v},\alpha}]\!]$ into $[\![\mathtt{Q}]\!]$. Note that, although CS_1 invokes four ENQUEUE operations for each $[\![\mathtt{ptr}(e_{i,j})]\!]$, there will be at most three elements been inserted. This is because that, for a certain edge $(\mathtt{u}, \mathtt{v})$, it must fall into one of the two cases: i) it is the first element in the linked lists, then $[\![\mathtt{head}_{\mathtt{u},\alpha}]\!]$ or $[\![\mathtt{head}_{\mathtt{v},\alpha}]\!]$ will be empty; ii) it is not the first element in the linked lists, then $[\![\mathtt{head}_{\mathtt{u},\alpha}]\!]$ or $[\![\mathtt{head}_{\mathtt{v},\alpha}]\!]$ must has been visited. In both cases, $[\![\mathtt{head}_{\mathtt{u},\alpha}]\!]$ or $[\![\mathtt{head}_{\mathtt{v},\alpha}]\!]$ will not be enqueued. Hence, CS_1 removes the last element from $[\![\mathtt{Q}]\!]$ after running the four insertions.

7) CS_1 computes an encrypted $E(\mathtt{flag}) = E(\widehat{\beta}_{\mathtt{u},\alpha} \geq \beta) E(\widehat{\beta}_{\mathtt{v},\alpha} \geq \beta) \overline{\mathsf{V}}_\mathtt{e}[e_{i,j}] \bmod \mathcal{N}$ and $E(\widetilde{e}_{ij}) = E(\bar{e}_{i,j} + r)$, where $r \in \mathcal{M}$ is a random number. Then, CS_1 sends $\mathtt{SE}(\mathtt{TK}_1, r)$, $E(\mathtt{flag})$, and $E(\widetilde{e}_{i,j})$ to CS_2.

8) On receiving $\mathtt{SE}(\mathtt{TK}_1, r)$, $E(\mathtt{flag})$ and $E(\widetilde{e}_{i,j})$, CS_2 decrypts $E(\mathtt{flag})$ and $E(\widetilde{e}_{i,j})$ to get $\mathtt{flag}$ and $\widetilde{e}_{ij}$. If $\mathtt{flag} = 1$, CS_2 sends $(\mathtt{SE}(\mathtt{TK}_1, r), \mathtt{SE}(\mathtt{TK}_2, \widetilde{e}_{i,j}))$ to the query user.

For the query user, it first initializes an empty bipartite graph $\mathcal{G}_\mathsf{req} = (\mathcal{U}_\mathsf{req}, \mathcal{V}_\mathsf{req}, \mathcal{E}_\mathsf{req})$. Upon receiving each $(\mathtt{SE}(\mathtt{TK}_1, r), \mathtt{SE}(\mathtt{TK}_2, \widetilde{e}_{i,j}))$, the query user uses $\mathtt{TK}_1$ and $\mathtt{TK}_2$ to respectively decrypt the two parts and gets $(r, \widetilde{e}_{i,j})$. Then, he/she can compute $\bar{e}_{i,j} = \widetilde{e}_{i,j} - r = u_i \times (|\mathcal{V}|+1) + v_j$. Based on $\bar{e}_{i,j}$, the query user further obtains $u_i = \lfloor \bar{e}_{i,j} / (|\mathcal{V}|+1) \rfloor$ and $v_i = \bar{e}_{i,j} \bmod (|\mathcal{V}|+1)$. After that, the query user updates $\mathcal{G}_\mathsf{req}$ by computing $\mathcal{U}_\mathsf{req} = \mathcal{U}_\mathsf{req} \cup \{u_i\}$, $\mathcal{V}_\mathsf{req} = \mathcal{V}_\mathsf{req} \cup \{v_j\}$, and $\mathcal{E}_\mathsf{req} = \mathcal{E}_\mathsf{req} \cup \{(u_i, v_j)\}$. Finally, after receiving all messages from CS_2, the query user can recover the query result $\mathcal{G}_\mathsf{req} = (\mathcal{U}_\mathsf{req}, \mathcal{V}_\mathsf{req}, \mathcal{E}_\mathsf{req})$.

5.2 Security Analysis for the Enhanced Scheme

Similar to the security model defined in Section 4.3, we formally define the security model for our security-enhanced scheme Π_E, which consists of a real model and an ideal model as follows.

Real Model. The real-world execution of our security-enhanced scheme Π_E happens between $\{\mathsf{CS}_1, \mathsf{CS}_2\}$, and $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$ is the pair of adversaries controlling the two servers, respectively. Assume that the inputs of CS_1 and CS_2 are respectively $x = \{\mathtt{K}_1, [\![\mathsf{NodeTable}^*]\!], [\![\mathsf{EdgeTable}^*]\!], [\![\mathtt{req}]\!]\}$ and $y = \{SK, \mathtt{K}_2\}$, and z is the auxiliary input, e.g., public parameters. With the inputs x, y, and z, the execution of Π_E under $\mathcal{A}$ in the real model can be defined as

$$\mathrm{REAL}_{\Pi_E, \mathcal{A}, z}(x, y) \stackrel{def}{=} \{\mathrm{Output}^{\Pi_E}(x, y), \mathrm{View}^{\Pi_E}(x, y), z\},$$

in which pairs $\mathrm{Output}^{\Pi_E}(x, y) = \{\mathrm{Output}_i^{\Pi_E}(x, y)\}_{i=1}^2$ and $\mathrm{View}^{\Pi_E}(x, y) = \{\mathrm{View}_i^{\Pi_E}(x, y)\}_{i=1}^2$ are the outputs and views of $\{\mathsf{CS}_i\}_{i=1}^2$, during the execution, respectively.

Ideal Model. in the ideal-world execution, there is a pair of ideal functionalities $\mathcal{F}^* = \{\mathcal{F}_1^*, \mathcal{F}_2^*\}$ respectively implementing the computation of the two servers, denoted as f_1^* and f_2^*, and each cloud server CS_i interacts with the corresponding ideal functionality $\mathcal{F}_i^*$. Here, the execution of $f^* = (f_1^*, f_2^*)$ under simulators $\mathtt{Sim}^* = \{\mathtt{Sim}_1^*, \mathtt{Sim}_2^*\}$ in the ideal-world model on input pair (x, y) and auxiliary input z is defined as

$$\mathrm{IDEAL}_{\mathcal{F}^*, \mathtt{Sim}^*, z} \stackrel{def}{=} \left\{ \begin{matrix} f_1^*(x, y), \mathtt{Sim}_1^*(x, f_1^*(x, y)), \\ f_2^*(x, y), \mathtt{Sim}_2^*(y, f_2^*(x, y)), z \end{matrix} \right\}.$$

Definition 3 (Security of Π_E against semi-honest adversary)**.** *Let $\mathcal{F}^*$ be a deterministic functionality and Π_E a protocol between the two servers. We say that Π_E securely realizes $\mathcal{F}$ if there exists $\mathtt{Sim}^*$ of PPT transformations (where $\mathtt{Sim}^* = \mathtt{Sim}^*(\mathcal{A})$) such that for semi-honest PPT adversaries $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$, for x, y, and z, for $\{\mathsf{CS}_1, \mathsf{CS}_2\}$ holds:*

$$\mathrm{REAL}_{\Pi_E, \mathcal{A}, z}(x, y) \stackrel{c}{\approx} \mathrm{IDEAL}_{\mathcal{F}^*, \mathtt{Sim}^*, z}(x, y).$$

Theorem 2. *The security-enhanced scheme can protect the plaintext and structure of the original bipartite graph $\mathcal{G}$, the plaintext and structure of the query requests and results, and the access pattern against the adversaries $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$.*

Proof. We show how to construct the simulators $\mathtt{Sim}^* = \{\mathtt{Sim}_1^*, \mathtt{Sim}_2^*\}$. Given $\{x, f_1^*(x,y), z\}$, where $f_1^*(x,y) = \{|\mathcal{E}_{\mathsf{vis}}|\}$, and $\mathcal{E}_{\mathsf{vis}}$ represents the set of edges that have been visited, $\mathtt{Sim}_1^*$ simulates $\mathcal{A}_1$ as follows: i) $\mathtt{Sim}_1^*$ first generates an array $\{\mathtt{isEmpty}\}'$ of $|\mathcal{E}_{\mathsf{vis}}|$ zeros followed by a one; ii) it randomly generates $4 \times |\mathcal{E}_{\mathsf{vis}}|$ random SHE ciphertexts to populate four arrays $\{E(\widehat{\beta}_{\mathtt{u},\alpha} \geq \beta)\}'$, $\{E(\widehat{\beta}_{\mathtt{v},\alpha} \geq \beta)\}'$, $\{E(\widehat{\beta}_{\mathtt{u},\alpha} < \beta)\}'$, and $\{E(\widehat{\beta}_{\mathtt{v},\alpha} < \beta)\}'$; iii) it outputs $\mathcal{A}_1$'s view. In addition to x and $f_1^*(x,y)$, $\mathcal{A}_1$ receives $\{\{\mathtt{isEmpty}\}, \{E(\widehat{\beta}_{\mathtt{u},\alpha} \geq \beta)\}, \{E(\widehat{\beta}_{\mathtt{v},\alpha} \geq \beta)\}, \{E(\widehat{\beta}_{\mathtt{u},\alpha} < \beta)\}, \{E(\widehat{\beta}_{\mathtt{v},\alpha} < \beta)\}\}$, while $\mathtt{Sim}_1^*$ outputs $\{\{\mathtt{isEmpty}\}', \{E(\widehat{\beta}_{\mathtt{u},\alpha} \geq \beta)\}', \{E(\widehat{\beta}_{\mathtt{v},\alpha} \geq \beta)\}', \{E(\widehat{\beta}_{\mathtt{u},\alpha} < \beta)\}', \{E(\widehat{\beta}_{\mathtt{v},\alpha} < \beta)\}'\}$. Based on the security of the SHE scheme, the views of $\mathcal{A}_1$ in the two executions are indistinguishable. On the other hand, given x and $f_2^*(x,y) = \{|\mathcal{E}_{\mathsf{vis}}|, |\mathcal{E}_{\mathsf{req}}|\}$, $\mathtt{Sim}_2^*$ simulates $\mathcal{A}_2$ as follows: i) $\mathtt{Sim}_2^*$ builds $\{\mathtt{isEmpty}\}'$ in the same approach used by $\mathtt{Sim}_1^*$; ii) it randomly generates an array $\{\mathtt{flag}\}'$ of $|\mathcal{E}_{\mathsf{vis}}|$ elements, where $|\mathcal{E}_{\mathsf{req}}|$ randomly located elements are 1s, and the other elements are 0s; iii) it randomly generates $3|\mathcal{E}_{\mathsf{vis}}|$ SHE ciphertexts to populate array $\{E(\widetilde{x})\}'$ of length $2|\mathcal{E}_{\mathsf{vis}}|$ and $\{E(\widetilde{e}_{i,j})\}'$ of length $|\mathcal{E}_{\mathsf{vis}}|$; iv) it randomly generates an array $\{\mathtt{SE}(\mathtt{TK}_1', r)\}'$ containing $|\mathcal{E}_{\mathsf{vis}}|$ random AES ciphertexts. In addition to y and $f_2^*(x,y)$, $\mathcal{A}_2$ receives $\{\widetilde{x}\}$, $\{E(\widetilde{e}_{i,j})\}$ and $\{\mathtt{SE}(\mathtt{TK}_1, r)\}$, while $\mathtt{Sim}_2^*$ outputs $\{\widetilde{x}\}'$, $\{E(\widetilde{e}_{i,j})\}'$ and $\{\mathtt{SE}(\mathtt{TK}_1, r)\}'$, as the corresponding plaintexts of the received ciphertexts contain random numbers chosen by CS_1, the two views of $\mathcal{A}_2$ during the two executions are indistinguishable.

Thus, the two adversaries can only obtain $\{x, f_1^*(x,y), z\}$ and $\{y, f_2^*(x,y), z\}$, and based on which, they cannot derive any useful information related to i) the plaintext and structure of the original bipartite graph $\mathcal{G}$; ii) the plaintext and structure of the query requests and results; and iii) the access pattern during conducting queries. □

The above analysis shows that, comparing to our basic scheme, our security-enhanced one can further protect the structure of the query requests and results, i.e., $|\mathcal{U} \cap \mathcal{R}|$, $|\mathcal{V} \cap \mathcal{R}|$, $|\mathcal{U}_{\mathsf{req}}|$, $|\mathcal{V}_{\mathsf{req}}|$, and $\{deg(n_i) \mid n_i \in \mathcal{U}_{\mathsf{req}} \cup \mathcal{V}_{\mathsf{req}}\}$.

6 Performance Evaluation

To evaluate the performance of our proposed schemes, we first theoretically analyze the computational overhead of the schemes. After that, we implement both schemes and evaluate their performance through extensive experiments.

6.1 Theoretical Analysis

In this subsection, we analyze the computational costs of the basic scheme and the security-enhanced one focusing on the three algorithms, namely, *Graph Outsourcing*, *Query Generating*, and *Query Conducting*.

Graph Outsourcing. In the graph outsourcing algorithms in both schemes, the data owner DO constructs the encrypted index and uploads it to the cloud. Specifically, DO first builds the linked list L_{n,α_k} for each $n \in (\mathcal{U} \cup \mathcal{V})$ and $\alpha_k = 1, \cdots, \alpha_{max}$. Based on these linked lists, DO builds $\{\mathsf{NodeTable}, \mathsf{EdgeTable}\}$ and $\{\mathsf{NodeTable}^*, \mathsf{EdgeTable}^*\}$ in the basic scheme and the security-enhanced one, respectively. In the basic scheme, the data owner builds NodeTable and EdgeTable, where NodeTable contains $|\mathcal{U} \cup \mathcal{V}| \times 2\alpha_{max}$ cells and EdgeTable contains $|\mathcal{E}| \times 2(\alpha_{max}+1)$ cells. To encrypt these two tables, DO needs to run respectively $|\mathcal{U} \cup \mathcal{V}| \times \alpha_{max} \times (\ell_{\mathtt{e}} + 1)$ and $|\mathcal{E}| \times (2\ell_{\mathtt{n}} + 2\ell_{\mathtt{e}}\alpha_{max})$ SHE encryption operations, where $\ell_{\mathtt{e}} = \lceil \log_2(|\mathcal{E}| + 1) \rceil$ and $\ell_{\mathtt{n}} = \lceil \log_2(|\mathcal{U} \cup \mathcal{V}| + 1) \rceil$. While in the security-enhanced scheme, NodeTable* and EdgeTable* have $|\mathcal{U} \cup \mathcal{V}| \times \alpha_{max}$ and $|\mathcal{E}| \times (2 + 6 \times \alpha_{max})$ cells, respectively. Hence, DO needs to run $|\mathcal{U} \cup \mathcal{V}| \times \ell_{\mathtt{e}} \times \alpha_{max}$ and $|\mathcal{E}| \times (2\ell_{\mathtt{e}} + (4\ell_{\mathtt{e}} + 2)\alpha_{max})$ to encrypt these two tables.

Query Generating. In both the basic scheme and the security-enhanced scheme, an authorized query user runs the same algorithm to encrypt his/her query request $\mathsf{req} = \{\alpha, \beta, \mathcal{R}\}$. Specifically, in this algorithm, the query user generates $(\lceil \log(\alpha_{max}+1) \rceil + 1 + |\mathcal{R}| \cdot \ell_{\mathtt{n}})$ SHE ciphertexts and 2 ciphertexts of the symmetric-key encryption scheme.

Query Conducting. On receiving a query request from an authorized user, the two cloud servers jointly conduct the query and return the query result to the user. Specifically, the query conducting process in both schemes comprises four parts, namely, i) preparing for query conduction, which includes computing $\{[\![\mathsf{NodeTable}]\!][\alpha], [\![\mathsf{EdgeTable}]\!][\alpha]\}$ in the basic scheme or $\{[\![\mathsf{NodeTable}^*]\!][\alpha], [\![\mathsf{EdgeTable}^*]\!][\alpha]\}$ in the security-enhanced scheme; ii) initializing a queue based on the encrypted query vertices $[\![\mathcal{R}]\!]$; iii) iterating over the two resulting tables with the queue; and iv) building the query result. We respectively discuss the computational cost of the two schemes in each part as follows.

- In the first part, CS_1 prepares for query conduction. Specifically, in the basic scheme, to compute $[\![\mathsf{NodeTable}]\!][\alpha]$, CS_1 needs to first conduct roughly $\alpha \cdot \lceil \log_2(\alpha+1) \rceil$ Homo-Mul-II operations to compute the flag for each columns indicating whether the column links to the querying α. After that, CS_1 computes $(1+\ell_{\mathtt{e}}) \cdot \alpha \cdot |\mathcal{U} \cup \mathcal{V}|$ Homo-Mul-I operations and $(1 + \ell_{\mathtt{e}}) \cdot (\alpha - 1) \cdot |\mathcal{U} \cup \mathcal{V}|$ Homo-Add-I operations. Furthermore, CS_1 runs **OP3** with CS_2 to refresh all ciphertexts in the resulting $[\![\mathsf{NodeTable}]\!][\alpha]$, during which CS_1 conducts $2 \cdot \alpha(1+\ell_{\mathtt{e}})|\mathcal{U} \cup \mathcal{V}|$ Homo-Add-I operations, and CS_2 conducts $(1 + \ell_{\mathtt{e}})|\mathcal{U} \cup \mathcal{V}|$ SHE encryptions and decryptions. For simplicity, hereinafter, we only consider the number of Homo-Mul-I and Homo-Mul-II operations, SHE encryptions and decryptions, and we respectively denote their costs as $\mathcal{C}_{\mathsf{Mul1}}$, $\mathcal{C}_{\mathsf{Mul2}}$, $\mathcal{C}_{\mathsf{Enc}}$, and $\mathcal{C}_{\mathsf{Dec}}$. Then, during computing $[\![\mathsf{NodeTable}]\!][\alpha]$, the computational costs for CS_1 and CS_2 are respectively $\alpha\lceil \log_2(\alpha+1) \rceil \cdot \mathcal{C}_{\mathsf{Mul2}} + (1+\ell_{\mathtt{e}})\alpha|\mathcal{U} \cup \mathcal{V}| \cdot \mathcal{C}_{\mathsf{Mul1}}$ and $(1 + \ell_{\mathtt{e}})|\mathcal{U} \cup \mathcal{V}|(\mathcal{C}_{\mathsf{Enc}} + \mathcal{C}_{\mathsf{Dec}})$. On the other hand, by reusing the flags, the computational costs for CS_1 and CS_2 to jointly compute $[\![\mathsf{EdgeTable}]\!][\alpha]$ are respectively $2\alpha\ell_{\mathtt{e}}|\mathcal{E}| \cdot \mathcal{C}_{\mathsf{Mul1}}$ and $2\ell_{\mathtt{e}}|\mathcal{E}|(\mathcal{C}_{\mathsf{Enc}} + \mathcal{C}_{\mathsf{Dec}})$. Hence, in the basic scheme, the overall computation cost in this part for the two servers is $\alpha\lceil \log_2(\alpha+1) \rceil \mathcal{C}_{\mathsf{Mul2}} + ((\ell_{\mathtt{e}}+1)|\mathcal{U} \cup \mathcal{V}| + 2\ell_{\mathtt{n}}|\mathcal{E}|)(\alpha\mathcal{C}_{\mathsf{Mul1}} + \mathcal{C}_{\mathsf{Enc}} + \mathcal{C}_{\mathsf{Dec}})$. Similarly, for the security-enhanced scheme in this part, the overall computational cost for the two servers is $\alpha\lceil \log_2(\alpha+1) \rceil \mathcal{C}_{\mathsf{Mul2}} + (\ell_{\mathtt{e}}|\mathcal{U} \cup \mathcal{V}| + (4\ell_{\mathtt{e}}+2)|\mathcal{E}|)(\alpha\mathcal{C}_{\mathsf{Mul1}} + \mathcal{C}_{\mathsf{Enc}} + \mathcal{C}_{\mathsf{Dec}})$.
- In the second part, CS_1 populates an empty queue based on the encrypted set of query vertices $[\![\mathcal{R}]\!]$. Specifi-

cally, in the basic scheme, CS_1 retrieves the records of all $[\![\mathtt{ptr}(n_i)]\!] \in [\![\mathcal{R}]\!]$ in NodeTable and enqueues each $n_i.[\![\mathtt{head}]\!]$ only if the corresponding $E(\widehat{\beta}) \geq E(\beta)$. Therefore, the computational cost for the basic scheme in this part is roughly $|\mathcal{R}|(|\mathcal{U} \cup \mathcal{V}|\ell_{\mathtt{n}}\mathcal{C}_{\mathsf{Mul2}} + (1+\ell_{\mathtt{e}})(|\mathcal{U} \cup \mathcal{V}|\mathcal{C}_{\mathsf{Mul1}} + \mathcal{C}_{\mathsf{Enc}} + \mathcal{C}_{\mathsf{Dec}}) + |\mathcal{U} \cup \mathcal{V}|\mathcal{C}_{\mathsf{Mul2}} + \mathcal{C}_{\mathsf{Enc}} + \mathcal{C}_{\mathsf{Dec}})$, where for each n_i, the first three terms represent the cost for retrieving the record from NodeTable and update the visited array, and the rest terms $(\mathcal{C}_{\mathsf{Enc}} + \mathcal{C}_{\mathsf{Dec}})$ represents the cost for comparing $\widehat{\beta}$ and β. On the other hand, in the security-enhanced scheme, CS_1 enqueues $u_i.[\![\mathtt{head}]\!]$ for all $[\![\mathtt{ptr}(u_i)]\!] \in [\![\mathcal{R}]\!]$. Therefore, the overall computational cost for the security-enhanced scheme in this part is $|\mathcal{R}| \cdot (|\mathcal{U} \cup \mathcal{V}|\ell_{\mathtt{n}}\mathcal{C}_{\mathsf{Mul2}} + \ell_{\mathtt{e}}(|\mathcal{U} \cup \mathcal{V}|\mathcal{C}_{\mathsf{Mul1}} + \mathcal{C}_{\mathsf{Enc}} + \mathcal{C}_{\mathsf{Dec}}) + |\mathcal{E}|\mathcal{C}_{\mathsf{Mul1}})$, where for each $u_i \in \mathcal{R}$, the first two terms represent the cost for retrieving a record from NodeTable*, the last term represents the cost for running **OP2** to update the visited array $\overline{\mathsf{V}}_{\mathtt{n}}$. Note that, in the security-enhanced scheme, the length of $\overline{\mathsf{V}}_{\mathtt{n}}$ is $|\mathcal{E}|$ as detailed in Section 5.

• In the third part, the two servers jointly iterate over the two encrypted tables {NodeTable, EdgeTable}, or {NodeTable*, EdgeTable*}. Focusing on each iteration, we analyze the computational cost of the two proposed schemes in this part as follows.

In the basic scheme, CS_1 dequeues a pointer $[\![\mathtt{ptr}(e_{i,j})]\!]$ in each iteration. Then, the two servers retrieve the corresponding record from $[\![\mathsf{EdgeTable}]\!][\alpha]$ and bootstrap the ciphertexts in the resulting record with a computational cost of $(|\mathcal{E}|\ell_{\mathtt{e}}\mathcal{C}_{\mathsf{Mul2}} + (2\ell_{\mathtt{n}} + 2\ell_{\mathtt{e}} + 2)(|\mathcal{E}|\mathcal{C}_{\mathsf{Mul1}} + \mathcal{C}_{\mathsf{Enc}} + \mathcal{C}_{\mathsf{Dec}}) + |\mathcal{E}|\mathcal{C}_{\mathsf{Mul1}})$. After that, based on the obtained $[\![\mathtt{ptr}(n')]\!]$, the servers retrieve the record from NodeTable and update the $\overline{\mathsf{V}}_{\mathtt{n}}$ with a computational cost of $(|\mathcal{U} \cup \mathcal{V}|\ell_{\mathtt{n}}\mathcal{C}_{\mathsf{Mul2}} + (\ell_{\mathtt{e}} + 1)(|\mathcal{U} \cup \mathcal{V}|\mathcal{C}_{\mathsf{Mul1}} + \mathcal{C}_{\mathsf{Enc}} + \mathcal{C}_{\mathsf{Dec}}) + |\mathcal{U} \cup \mathcal{V}|\mathcal{C}_{\mathsf{Mul1}})$. Furthermore, the servers jointly compare $n'.E(\beta)$ and $E(\widehat{\beta})$, and CS_1 checks whether n' and $e_{i,j}$ have been visited with a computational cost of $\mathcal{C}_{\mathsf{Enc}} + \mathcal{C}_{\mathsf{Dec}} + (|\mathcal{U} \cup \mathcal{V}|\ell_{\mathtt{e}}\mathcal{C}_{\mathsf{Mul2}} + |\mathcal{U} \cup \mathcal{V}|\mathcal{C}_{\mathsf{Mul1}}) + (|\mathcal{E}|\ell_{\mathtt{n}}\mathcal{C}_{\mathsf{Mul2}} + |\mathcal{E}|\mathcal{C}_{\mathsf{Mul1}})$. Therefore, the overall computational cost for the basic scheme in each iteration is $(2\ell_{\mathtt{n}} + 3\ell_{\mathtt{e}} + 4)(\mathcal{C}_{\mathsf{Enc}} + \mathcal{C}_{\mathsf{Dec}}) + ((\ell_{\mathtt{e}} + 3)|\mathcal{U} \cup \mathcal{V}| + (2\ell_{\mathtt{n}} + 2\ell_{\mathtt{e}} + 4)|\mathcal{E}|)\mathcal{C}_{\mathsf{Mul1}} + (|\mathcal{U} \cup \mathcal{V}| + |\mathcal{E}|)(\ell_{\mathtt{e}} + \ell_{\mathtt{n}})\mathcal{C}_{\mathsf{Mul2}}$.

In the security-enhanced scheme, after dequeuing a pointer $[\![\mathtt{ptr}(e_{i,j})]\!]$, the servers read the corresponding record from EdgeTable* with a computational cost of $(|\mathcal{E}|\ell_{\mathtt{e}}\mathcal{C}_{\mathsf{Mul2}} + (4\ell_{\mathtt{e}} + 2)(|\mathcal{E}|\mathcal{C}_{\mathsf{Mul1}} + \mathcal{C}_{\mathsf{Enc}} + \mathcal{C}_{\mathsf{Dec}}) + |\mathcal{E}|\mathcal{C}_{\mathsf{Mul1}})$. After that, they jointly compare $u_i.E(\widehat{\beta})$ and $v_i.E(\widehat{\beta})$ with $E(\beta)$, and the computational cost is $2(\mathcal{C}_{\mathsf{Enc}} + \mathcal{C}_{\mathsf{Dec}})$. Then, to compute $\{E(\mathtt{enable}_i)\}_{i=1}^4$, the computational cost for the servers is $4|\mathcal{E}|(\ell_{\mathtt{e}}\mathcal{C}_{\mathsf{Mul2}} + \mathcal{C}_{\mathsf{Mul1}})$. Finally, based on $\{E(\mathtt{enable}_i)\}_{i=1}^4$, the servers run four enqueue operations with a computational cost of $4|[\![\mathtt{Q}]\!].\mathtt{arr}|(\ell_{\mathtt{e}} + 2)\mathcal{C}_{\mathsf{Mul1}}$, where $|[\![\mathtt{Q}]\!].\mathtt{arr}|$ represents the length of $[\![\mathtt{Q}]\!].\mathtt{arr}$. Therefore, the overall computational cost for the security-enhanced scheme in each iteration is $(4\ell_{\mathtt{e}} + 4)(\mathcal{C}_{\mathsf{Enc}} + \mathcal{C}_{\mathsf{Dec}}) + ((4\ell_{\mathtt{e}} + 7)|\mathcal{E}| + 4|[\![\mathtt{Q}]\!].\mathtt{arr}|(\ell_{\mathtt{e}} + 2))\mathcal{C}_{\mathsf{Mul1}} + 5|\mathcal{E}|\ell_{\mathtt{e}}\mathcal{C}_{\mathsf{Mul2}}$.

• In the fourth part, the two servers re-encrypt the query results. Specifically, in the basic scheme, CS_1 packs the two pointers $[\![\mathtt{ptr}(u_i)]\!]$, $[\![\mathtt{ptr}(v_i)]\!]$ for $e_{i,j} \in \mathtt{Edges}$, and then decrypts the resulting ciphertext with CS_2. Hence, the overall computational cost for the basic scheme in this part is $|\mathtt{Edges}|((2\ell_{\mathtt{n}} + 1)\mathcal{C}_{\mathsf{Mul2}} + \mathcal{C}_{\mathsf{Dec}})$. On the other hand, for the security-enhanced scheme, the two servers only need to decrypt one ciphertext for each edge in $|\mathtt{Edges}|$, i.e., their computational cost is $|\mathtt{Edges}|\mathcal{C}_{\mathsf{Dec}}$.

6.2 Experimental Results

In this section, we evaluate the performance of our proposed schemes. Specifically, we implement our basic scheme and security-enhanced scheme in Rust, and evaluate the implementation on an platform equipped with an Intel(R) Xeon(R) Gold 6140 CPU @2.30GHz, 64GB RAM and running Ubuntu 21.04 LTS operating system. We adopt two bipartite graph datasets, namely, Wikiquote Edits (da)[1] and NIPS full papers[2], respectively denoted as $\mathcal{G}_{\mathsf{WIKI}}$ and $\mathcal{G}_{\mathsf{NIPS}}$. To demonstrate the performance of our schemes on datasets of different sizes, we randomly generate five datasets by randomly sampling 5000, 7500, 10000, 12500, 15000 edges from $\mathcal{G}_{\mathsf{WIKI}}$ and $\mathcal{G}_{\mathsf{NIPS}}$, respectively, denoted by $\{\mathcal{G}_{\mathsf{WIKI}}^{(1)}, \mathcal{G}_{\mathsf{WIKI}}^{(2)}, \mathcal{G}_{\mathsf{WIKI}}^{(3)}, \mathcal{G}_{\mathsf{WIKI}}^{(4)}, \mathcal{G}_{\mathsf{WIKI}}^{(5)}\}$, and $\{\mathcal{G}_{\mathsf{NIPS}}^{(1)}, \mathcal{G}_{\mathsf{NIPS}}^{(2)}, \mathcal{G}_{\mathsf{NIPS}}^{(3)}, \mathcal{G}_{\mathsf{NIPS}}^{(4)}, \mathcal{G}_{\mathsf{NIPS}}^{(5)}\}$. In the table, we denote the maximum numbers of neighbors for a vertex in $\mathcal{U}$ and $\mathcal{V}$ as α_{max} and β_{max}, respectively. Recalling that α and β are respectively the constraints on number of neighbors for vertices in $\mathcal{U}$ and $\mathcal{V}$, we can swap $\mathcal{U}$ and $\mathcal{V}$ such that $\alpha_{max} \leq \beta_{max}$ to reduce the number of columns in the index. Furthermore, to demonstrate the distribution of each vertex's degree in $\mathcal{U}$ and $\mathcal{V}$ for each dataset, we compute an array containing the degrees of all vertices for each dataset and plot the sorted arrays in Fig. 6.

TABLE 1
Details of the Four Datasets

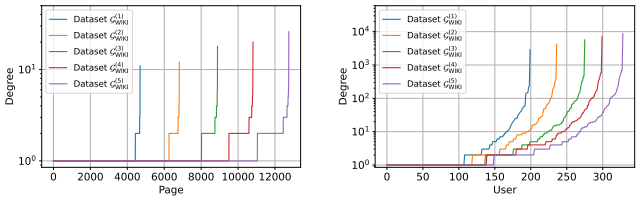
	$\mathcal{U}$	$\mathcal{V}$	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	α_{max}	β_{max}
$\mathcal{G}_{\mathsf{WIKI}}^{(1)}$	Page	User	4694	200	5000	11	2884
$\mathcal{G}_{\mathsf{WIKI}}^{(2)}$	Page	User	6819	237	7500	12	4271
$\mathcal{G}_{\mathsf{WIKI}}^{(3)}$	Page	User	8886	276	10000	18	5684
$\mathcal{G}_{\mathsf{WIKI}}^{(4)}$	Page	User	10809	300	12500	20	7144
$\mathcal{G}_{\mathsf{WIKI}}^{(5)}$	Page	User	12750	329	15000	26	8664
$\mathcal{G}_{\mathsf{NIPS}}^{(1)}$	Document	Word	1414	2657	5000	13	13
$\mathcal{G}_{\mathsf{NIPS}}^{(2)}$	Document	Word	1472	3403	7500	18	20
$\mathcal{G}_{\mathsf{NIPS}}^{(3)}$	Document	Word	1491	4040	10000	22	25
$\mathcal{G}_{\mathsf{NIPS}}^{(4)}$	Document	Word	1490	4474	12500	23	30
$\mathcal{G}_{\mathsf{NIPS}}^{(5)}$	Document	Word	1490	4880	15000	23	35

Graph Outsourcing. As analyzed in Section 6.1, the computational cost for the data owner to outsource the graph is linear to the number of nodes $|\mathcal{U} \cup \mathcal{V}|$ and the number of edges $|\mathcal{E}|$, which is also demonstrated in Fig. 7. Furthermore, as α_{max} is larger for $\mathcal{G}_{\mathsf{NIPS}}^{(i)}$ datasets, the time consumption for DO to encrypt the edge tables is slightly higher than that for the $\mathcal{G}_{\mathsf{WIKI}}^{(i)}$ datasets.

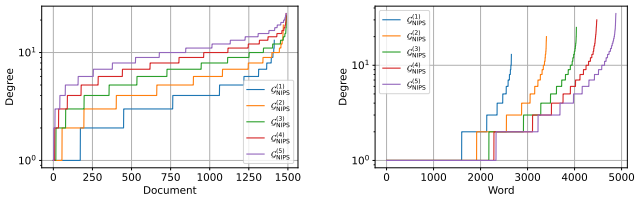
Query Generating. As shown in Fig. 8, the time consumption for an authorized user to encrypt a query request increases with the number of query vertices $|\mathcal{R}|$. Furthermore, comparing with $\mathsf{G}_{\mathsf{NIPS}}^{(i)}$ datasets, $\mathcal{G}_{\mathsf{WIKI}}^{(i)}$ datasets have more vertices and larger α_{max}, so it takes more time for the user to encrypt a query request.

1. http://konect.cc/networks/edit-dawikisource/
2. http://konect.cc/networks/bag-nips/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3169386, IEEE Transactions on Dependable and Secure Computing

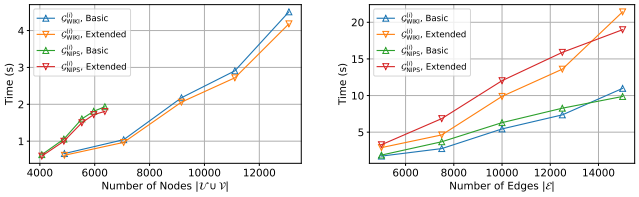


(a) Degree of each $\mathcal{U}$ in $\mathcal{G}^{(i)}_{\mathsf{WIKI}}$'s (b) Degree of each $\mathcal{V}$ in $\mathcal{G}^{(i)}_{\mathsf{WIKI}}$'s



(c) Degree of each $\mathcal{U}$ in $\mathcal{G}^{(i)}_{\mathsf{NIPS}}$'s (d) Degree of each $\mathcal{V}$ in $\mathcal{G}^{(i)}_{\mathsf{NIPS}}$'s

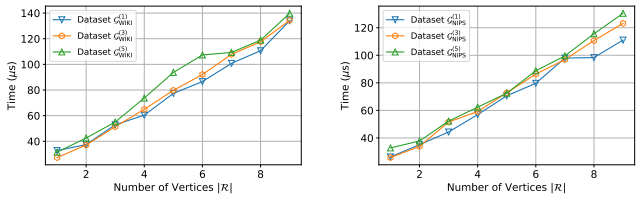
Fig. 6. The distributions of the sorted degrees for vertices in $\mathcal{U}$ and $\mathcal{V}$ in both datasets. Specifically, we compute the degree of each vertex in $\mathcal{U}$ and $\mathcal{V}$ of $\mathcal{G}^{(i)}_{\mathsf{WIKI}}$'s and $\mathcal{G}^{(i)}_{\mathsf{NIPS}}$'s, and plot the sorted degrees in the figures.



(a) Encrypting node tables (b) Encrypting edge tables

Fig. 7. The average time consumption for DO to encrypt {NodeTable, EdgeTable} or {NodeTable*, EdgeTable*} in the two proposed scheme for the datasets $\mathcal{G}^{(i)}_{\mathsf{WIKI}}$ and $\mathcal{G}^{(i)}_{\mathsf{NIPS}}$.

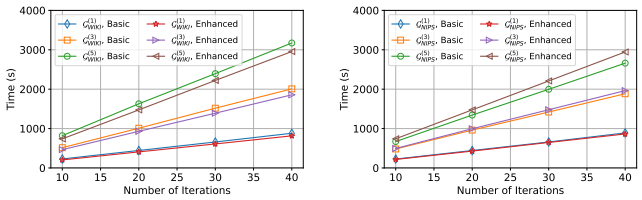
Query Conducting. In Fig. 9, we plot the overall time consumption for the two servers to handle (α, β)-core queries. Specifically, since the time consumption significantly affected by the chosen query vertices $u_i \in \mathcal{R}$, α, and β, in the figures, we show the relationship between average time consumption for a query and the number of iteration of the query, i.e., the number of edges been visited during conducting the query. As shown in the figures, the time consumption linearly increases with the number of iterations. Furthermore, as shown in the figures, it takes similar time for the security-enhanced scheme to query on $\mathcal{G}^{(i)}_{\mathsf{WIKI}}$ datasets and $\mathcal{G}^{(i)}_{\mathsf{NIPS}}$ datasets, while the time



(a) For $\mathcal{G}^{(i)}_{\mathsf{WIKI}}$ datasets (b) For $\mathcal{G}^{(i)}_{\mathsf{NIPS}}$ datasets

Fig. 8. The average time consumption for an authorized user to encrypt a query request containing different numbers of vertices $|\mathcal{R}|$.

consumption for the basic scheme to process a query on $\mathcal{G}^{(i)}_{\mathsf{WIKI}}$ is higher than that for a query on $\mathcal{G}^{(i)}_{\mathsf{NIPS}}$. The differences are introduced mainly at when retrieving a row from $[\![\mathsf{NodeTable}]\!][\alpha]$, $[\![\mathsf{NodeTable}^*]\!][\alpha]$, $[\![\mathsf{EdgeTable}]\!][\alpha]$ and $[\![\mathsf{EdgeTable}^*]\!][\alpha]$, as shown in Fig. 11. Specifically, in the third part of the security-enhanced scheme, it will only traverse the edge table $[\![\mathsf{EdgeTable}^*]\!]$ and the two types of datasets have the same number of edges. However, in the third part of the basic scheme, the two servers will read both the edge table EdgeTable and the node table NodeTable, and the $\mathcal{G}^{(i)}_{\mathsf{WIKI}}$ datasets contain more vertices than the $\mathcal{G}^{(i)}_{\mathsf{NIPS}}$ ones.

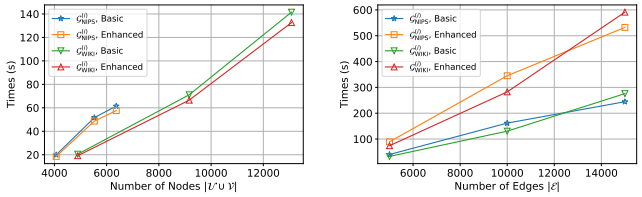


(a) For $\mathcal{G}^{(i)}_{\mathsf{WIKI}}$ datasets (b) For $\mathcal{G}^{(i)}_{\mathsf{NIPS}}$ datasets

Fig. 9. The average time consumption for conducting a query with different numbers of iterations.

To elaborate further on our schemes' performance in the query conducting phase, we specifically evaluate the time consumption for i) computing $\{[\![\mathsf{NodeTable}]\!][\alpha], [\![\mathsf{EdgeTable}]\!][\alpha]\}$ or $\{[\![\mathsf{NodeTable}^*]\!][\alpha], [\![\mathsf{EdgeTable}^*]\!][\alpha]\}$; ii) retrieving a record from $\{[\![\mathsf{NodeTable}]\!][\alpha], [\![\mathsf{EdgeTable}]\!][\alpha]\}$ or $\{[\![\mathsf{NodeTable}^*]\!][\alpha], [\![\mathsf{EdgeTable}^*]\!][\alpha]\}$; and iii) enqueuing an element into $[\![\mathsf{Q}]\!]$ as follows.

- As shown in Fig. 10, on the same dataset, the time consumption for retrieving $[\![\mathsf{NodeTable}]\!][\alpha]$ is higher than that for $[\![\mathsf{NodeTable}^*]\!][\alpha]$, while retrieving $[\![\mathsf{EdgeTable}^*]\!][\alpha]$ consumes more time than retrieving $[\![\mathsf{EdgeTable}]\!][\alpha]$. This is mainly because that, although NodeTable and $[\![\mathsf{NodeTable}^*]\!][\alpha]$ has the same number of rows, NodeTable contains more columns than NodeTable*, while EdgeTable* has more columns than EdgeTable, as shown in Table 1.

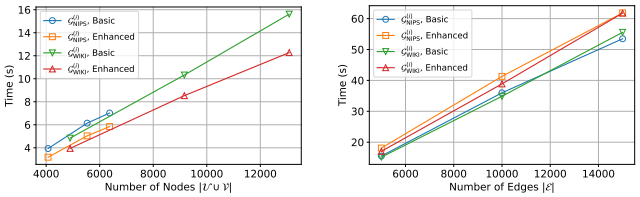


(a) For node tables (b) For edge tables

Fig. 10. The average time consumption for computing $\{[\![\mathsf{NodeTable}]\!][\alpha], [\![\mathsf{EdgeTable}]\!][\alpha]\}$ or $\{[\![\mathsf{NodeTable}^*]\!][\alpha], [\![\mathsf{EdgeTable}^*]\!][\alpha]\}$.

- As shown in Fig. 11, we plot the average time consumption for retrieving a record from $[\![\mathsf{NodeTable}]\!][\alpha]$, $[\![\mathsf{NodeTable}^*]\!][\alpha]$, $[\![\mathsf{EdgeTable}]\!][\alpha]$ and $[\![\mathsf{EdgeTable}^*]\!][\alpha]$. As shown in the figure, the average time consumption for retrieving a record from the tables increases with the number of records in the tables.
- To demonstrate the efficiency of the encrypted queue $[\![\mathsf{Q}]\!]$, we plot Fig. 12 to show the time consumption for enqueuing an element into a queue $[\![\mathsf{Q}]\!]$ with different

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3169386, IEEE Transactions on Dependable and Secure Computing



(a) Reading node tables (b) Reading edge tables

Fig. 11. The average time consumption for retrieving a row from $[\![\texttt{NodeTable}]\!][\alpha]$, $[\![\texttt{NodeTable}^*]\!][\alpha]$, $[\![\texttt{EdgeTable}]\!][\alpha]$ and $[\![\texttt{EdgeTable}^*]\!][\alpha]$ versus different sizes of tables.

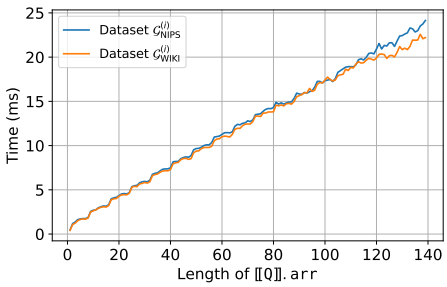


Fig. 12. The average time consumption for enqueuing an element into $[\![\texttt{Q}]\!]$ with different lengths of $[\![\texttt{Q}]\!].\texttt{arr}$.

lengths of $[\![\texttt{Q}]\!].\texttt{arr}$. As shown in the figure, the time consumption increases with the length of $[\![\texttt{Q}]\!].\texttt{arr}$, and when $|[\![\texttt{Q}]\!].\texttt{arr}| = 150$, enqueuing a pointer represented as an array of $\lceil \log_2(15000+1) \rceil = 6$ SHE ciphertexts takes 25 ms.

As for the query user, on receiving the query result from the servers, he/she only needs to run several decryptions of the underlying SE scheme, which is usually negligible. Hence, the overall computational cost for the query user to launch a query to the proposed schemes lies in the query generation phase, and it is analyzed to be efficient in Section 6.1. Therefore, the overall computational cost for the query user to launch an (α, β)-core query to the proposed schemes is efficient.

7 Related Work

In this section, we review some related works, which are closely related to our proposed scheme in terms of privacy-preserving graph query.

To preserve data privacy while querying over a graph, many schemes [10]–[17] based on k-anonymity have been proposed. On the one hand, some of them aim to preserve the privacy of vertices. Hay et al. [10] presented a perturbation technique to perform a sequence of random edge deletions and edge insertions, such that there will be at least k candidate vertices being matched by any structural query over the resulting graph. Liu et al. [14] proposed a solution to ensure that, for any vertex in the graph, there exist at least $k-1$ vertices having the same degree with it while minimizing the number of edge modifications. Focusing on neighborhood attacks, Zhou et al. [12] proposed a social network anonymization solution based on adding edges, such that with the anonymized graph, aggregated network queries (e.g., average distance) can be answered with satisfactory accuracy. Different from Zhou et al.'s work, Liu et al. [16] proposed a scheme to anonymize weighted social networks to resist the weighted 1*-neighborhood attack. On the other hand, some schemes were proposed to protect the privacy of edges. To prevent link re-identification, Zheleva et al. [11] proposed a scheme based on edge clustering and removal. Ying et al. [13] proposed an algorithm to anonymize a graph to prevent adversaries from checking the existance of certain links. In [15], the authors considered ϵ-differential privacy for edges and proposed a graph anonymization solution that can preserve the structural information for specific data analysis tasks, such as degree distribution, cut query and shortest path length. Chang et al. [17] proposed a framework to conduct subgraph matching, where the structural and label privacy are preserved by employing the k-automorphism model. However, the neighbors of a vertex in a graph may vary after being anonymized by these k-anonymity–based schemes, the accuracy of the (α, β)-core query might be significantly degraded. Hence, they cannot be applied to our scenario.

There are also many schemes [18]–[28] built upon homomorphic encryption schemes or multi-party computing. Some of the schemes [18]–[22] focus on traditional graph problems, such as shortest distance problem, minimum spanning tree, and maximum flow problem. Aly et al. [18] proposed a scheme to solve problems such as the shortest distance problem, through multi-party computation. Blanton et al. [19] built algorithms based on ORAM to obliviously implement fundamental graph algorithms, e.g., breadth-first search and single-source single-destination shortest path. As these two schemes represent a graph by its adjacency matrix, they cannot efficiently handle large sparse matrices. Meng et al. [20] proposed schemes to process approximate shortest distance queries with symmetric-key encryption and somewhat homomorphic encryption. To handle accurate shortest path computation, Wu et al. [21] built a scheme by employing private information retrieval (PIR) and garbled circuits. Based on additive homomorphic encryption scheme and garbled circuits, Wang et al. [22] built their scheme to conduct accurate shortest distance queries, and their scheme can support graph updates. Some of the schemes [23], [24], [26] were proposed to handle subgraph queries. Cao et al. [23] proposed a scheme to handle subgraph queries with a filtering-and-verification manner by converting filtration into inner products. In [24], Fan et al. considered to support subgraph query processing, where the data graph is publicly known and the query structure/topology is kept secret. Xu et al. [26] built their scheme upon a somewhat homomorphic encryption scheme to process strong simulation queries over a plaintext large graph, while preserving the privacy of queries. Meanwhile, some works [27], [28] have been proposed to support message passing algorithms, which can support many traditional graph queries; however, deploying message passing algorithms to achieve (α, β)-core queries is still challenging. However, these schemes are tailored for specific types of queries, and cannot efficiently support (α, β)-core queries.

8 Conclusion

In this paper, we have presented an efficient and privacy-preserving (α, β)-core query scheme for bipartite graphs in

the cloud. Specifically, we first designed an index containing two tables built from the bipartite graph, where each row in the two tables respectively represents a vertex and an edge in the bipartite graph. Then, we built our basic scheme where the index and query requests are encrypted by the SHE scheme. Furthermore, we built a security-enhanced scheme by revising the index structure and designing an encrypted queue. The security analysis shows that the basic scheme can protect the plaintext and structure of the dataset and the plaintext of the query requests and results, and achieve access pattern privacy, while the security-enhanced scheme can further preserve the structure privacy of the query requests and results. After that, we demonstrated the performance of our proposed schemes through extensive experiments, and the result shows that our proposed schemes are indeed computationally efficient. In our future work, we will further evaluate our schemes in real-world scenarios.

ACKNOWLEDGMENTS

This research was supported in part by NSERC Discovery Grants (04009), ZJNSF LZ18F020003, NSFC U1709217.

REFERENCES

[1] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos, "Neighborhood formation and anomaly detection in bipartite graphs," in *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005), 27-30 November 2005, Houston, Texas, USA*. IEEE Computer Society, 2005, pp. 418–425.

[2] Z. Liu, L. Cui, W. Guo, W. He, H. Li, and J. Gao, "Predicting hospital readmission using graph representation learning based on patient and disease bipartite graph," in *DASFAA*. Springer, 2020, pp. 385–397.

[3] N. Benchettara, R. Kanawati, and C. Rouveirol, "Supervised machine learning applied to link prediction in bipartite social networks," in *International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2010, Odense, Denmark, August 9-11, 2010*. IEEE Computer Society, 2010, pp. 326–330.

[4] D. Ding, H. Li, Z. Huang, and N. Mamoulis, "Efficient fault-tolerant group recommendation using alpha-beta-core," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*. ACM, 2017, pp. 2047–2050.

[5] K. Wang, W. Zhang, X. Lin, Y. Zhang, L. Qin, and Y. Zhang, "Efficient and effective community search on large-scale bipartite graphs," in *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*. IEEE, 2021, pp. 85–96.

[6] T. Wu, Y. Wang, Y. Wang, E. Zhao, and Y. Yuan, "Leveraging graph-based hierarchical medical entity embedding for healthcare applications," *Scientific reports*, vol. 11, no. 1, pp. 1–13, 2021.

[7] Y. Zheng, R. Lu, Y. Guan, J. Shao, and H. Zhu, "Achieving efficient and privacy-preserving exact set similarity search over encrypted data," *IEEE Trans. Dependable Secur. Comput.*, 2020.

[8] Y. Zheng, R. Lu, Y. Guan, J. Shao, and H. Zhu, "Efficient and privacy-preserving similarity range query over encrypted time series data," *IEEE Trans. Dependable Secur. Comput.*, pp. 1–1, 2021.

[9] S. Zhang, S. Ray, R. Lu, Y. Zheng, and J. Shao, "Preserving location privacy for outsourced most-frequent item query in mobile crowdsensing," *IEEE Internet Things J.*, vol. 8, no. 11, pp. 9139–9150, 2021.

[10] M. Hay, G. Miklau, D. Jensen, P. Weis, and S. Srivastava, "Anonymizing social networks," *Computer science department faculty publication series*, p. 180, 2007.

[11] E. Zheleva and L. Getoor, "Preserving the privacy of sensitive relationships in graph data," in *International workshop on privacy, security, and trust in KDD*. Springer, 2007, pp. 153–171.

[12] B. Zhou and J. Pei, "Preserving privacy in social networks against neighborhood attacks," in *24th ICDE*. IEEE Computer Society, 2008, pp. 506–515.

[13] X. Ying and X. Wu, "Randomizing social networks: a spectrum preserving approach," in *SDM 2008*. SIAM, 2008, pp. 739–750.

[14] K. Liu and E. Terzi, "Towards identity anonymization on graphs," in *SIGMOD 2008*. ACM, 2008, pp. 93–106.

[15] R. Chen, B. C. M. Fung, P. S. Yu, and B. C. Desai, "Correlated network data publication via differential privacy," *VLDB J.*, vol. 23, no. 4, pp. 653–676, 2014.

[16] Q. Liu, G. Wang, F. Li, S. Yang, and J. Wu, "Preserving privacy with probabilistic indistinguishability in weighted social networks," *IEEE Trans. Parallel Distributed Syst.*, vol. 28, no. 5, pp. 1417–1429, 2017.

[17] Z. Chang, L. Zou, and F. Li, "Privacy preserving subgraph matching on large graphs in cloud," in *SIGMOD 2016*. ACM, 2016, pp. 199–213.

[18] A. Aly, E. Cuvelier, S. Mawet, O. Pereira, and M. V. Vyve, "Securely solving simple combinatorial graph problems," in *17th FC*, vol. 7859. Springer, 2013, pp. 239–257.

[19] M. Blanton, A. Steele, and M. Aliasgari, "Data-oblivious graph algorithms for secure computation and outsourcing," in *8th ASIA CCS*. ACM, 2013, pp. 207–218.

[20] X. Meng, S. Kamara, K. Nissim, and G. Kollios, "GRECS: graph encryption for approximate shortest distance queries," in *Proceedings of the 22nd ACM CCS, 2015*. ACM, 2015, pp. 504–517.

[21] D. J. Wu, J. Zimmerman, J. Planul, and J. C. Mitchell, "Privacy-preserving shortest path computation," in *23rd NDSS*. The Internet Society, 2016.

[22] Q. Wang, K. Ren, M. Du, Q. Li, and A. Mohaisen, "Secgdb: Graph encryption for exact shortest distance queries with efficient updates," in *21st FC*, vol. 10322. Springer, 2017, pp. 79–97.

[23] N. Cao, Z. Yang, C. Wang, K. Ren, and W. Lou, "Privacy-preserving query over encrypted graph-structured data in cloud computing," in *ICDCS 2011*. IEEE Computer Society, 2011, pp. 393–402.

[24] Z. Fan, B. Choi, J. Xu, and S. S. Bhowmick, "Asymmetric structure-preserving subgraph queries for large graphs," in *31st ICDE*. IEEE Computer Society, 2015, pp. 339–350.

[25] Z. Fan, B. Choi, Q. Chen, J. Xu, H. Hu, and S. S. Bhowmick, "Structure-preserving subgraph query services," in *32nd ICDE*. IEEE Computer Society, 2016, pp. 1532–1533.

[26] L. Xu, J. Jiang, B. Choi, J. Xu, and S. S. Bhowmick, "Privacy preserving strong simulation queries on large graphs," in *37th ICDE*. IEEE, 2021, pp. 61–72.

[27] K. Nayak, X. S. Wang, S. Ioannidis, U. Weinsberg, N. Taft, and E. Shi, "Graphsc: Parallel secure computation made easy," in *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. IEEE Computer Society, 2015, pp. 377–394.

[28] T. Araki, J. Furukawa, K. Ohara, B. Pinkas, H. Rosemarin, and H. Tsuchida, "Secure graph analysis at scale," in *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*. ACM, 2021, pp. 610–629.

[29] H. Mahdikhani, R. Lu, Y. Zheng, J. Shao, and A. A. Ghorbani, "Achieving O($\log^3$n) communication-efficient privacy-preserving range query in fog-based iot," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 5220–5232, 2020.

[30] V. Nikolaenko, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft, and D. Boneh, "Privacy-preserving matrix factorization," in *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*. ACM, 2013, pp. 801–812.

[31] Y. Zheng, H. Duan, and C. Wang, "Learning the truth privately and confidently: Encrypted confidence-aware truth discovery in mobile crowdsensing," *IEEE Trans. Inf. Forensics Secur.*, vol. 13, no. 10, pp. 2475–2489, 2018.

[32] G. Xu, H. Li, Y. Zhang, S. Xu, J. Ning, and R. Deng, "Privacy-preserving federated deep learning with irregular users," *IEEE Trans. Dependable Secur. Comput.*, pp. 1–1, 2020.

[33] H. Yu, X. Jia, H. Zhang, X. Yu, and J. Shu, "Psride: Privacy-preserving shared ride matching for online ride hailing systems," *IEEE Trans. Dependable Secur. Comput.*, vol. 18, no. 3, pp. 1425–1440, 2021.

[34] Y. Guan, R. Lu, Y. Zheng, S. Zhang, J. Shao, and G. Wei, "Toward privacy-preserving cybertwin-based spatio-temporal keyword query for its in 6g era," *IEEE Internet Things J.*, pp. 1–1, 2021.

[35] O. Goldreich, *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2009.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3169386, IEEE Transactions on Dependable and Secure Computing

Yunguo Guan is a PhD student of the Faculty of Computer Science, University of New Brunswick, Canada. His research interests include applied cryptography and game theory.

Jun Shao received the Ph.D. degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China, in 2008.

He was a Post-Doctoral Fellow with the School of Information Sciences and Technology, Pennsylvania State University, Pennsylvania, PA, USA, from 2008 to 2010. He is currently a Professor with the School of Computer and Information Engineering, Zhejiang Gongshang University, Hangzhou, China. His current research interests include network security and applied cryptography.

Rongxing Lu (S'09-M'11-SM'15-F'21) is a University Research Scholar, an associate professor at the Faculty of Computer Science (FCS), University of New Brunswick (UNB), Canada. Before that, he worked as an assistant professor at the School of Electrical and Electronic Engineering, Nanyang Technological University (NTU), Singapore from April 2013 to August 2016. Rongxing Lu worked as a Postdoctoral Fellow at the University of Waterloo from May 2012 to April 2013. He was awarded the most prestigious "Governor General's Gold Medal", when he received his PhD degree from the Department of Electrical & Computer Engineering, University of Waterloo, Canada, in 2012; and won the 8th IEEE Communications Society (ComSoc) Asia Pacific (AP) Outstanding Young Researcher Award, in 2013. Dr. Lu is an IEEE Fellow. His research interests include applied cryptography, privacy enhancing technologies, and IoT-Big Data security and privacy. He has published extensively in his areas of expertise (with H-index 76 from Google Scholar as of Feb. 2022), and was the recipient of 9 best (student) paper awards from some reputable journals and conferences. Currently, Dr. Lu serves as the Chair of IEEE ComSoc CIS-TC (Communications and Information Security Technical Committee), and the founding Co-chair of IEEE TEMS Blockchain and Distributed Ledgers Technologies Technical Committee (BDLT-TC). Dr. Lu is the Winner of 2016-17 Excellence in Teaching Award, FCS, UNB.

Guiyi Wei is a professor of the School of Computer and Information Engineering at Zhejiang Gongshang University. He obtained his Ph.D. in Dec 2006 from Zhejiang University, where he was advised by Cheung Kong chair professor Yao Zheng. His research interests include wireless networks, mobile computing, cloud computing, social networks and network security.

Yandong Zheng received her M.S. degree from the Department of Computer Science, Beihang University, China, in 2017 and she is currently pursuing her Ph.D. degree in the Faculty of Computer Science, University of New Brunswick, Canada. Her research interest includes cloud computing security, big data privacy and applied privacy.

Songnian Zhang received his M.S. degree from Xidian University, China, in 2016 and he is currently pursuing his Ph.D. degree in the Faculty of Computer Science, University of New Brunswick, Canada. His research interest includes cloud computing security, big data query and query privacy.