# PACS: Privacy-Preserving Attribute-Driven Community Search over Attributed Graphs

Fangyuan Sun[†], Yaxi Yang[‡], Jia Yu[†✉], Jianying Zhou[‡]

[†]Qingdao University [‡]Singapore University of Technology and Design

sakgofish@gmail.com, yxyangjnu@gmail.com, qduyujia@gmail.com, jianying_zhou@sutd.edu.sg

***Abstract*—In data-driven applications, attribute-driven community search has attracted increasing attention, which aims to help users find high-quality subgraphs that meet specific requirements over attributed graphs. Nevertheless, few works consider data privacy when performing community search. One critical reason is that real-world graphs continue to grow in size, and attribute-driven community search involves computing complex metrics on encrypted graph data, including structural cohesiveness and attribute correlation, which are too time-consuming to be practical.**

**This paper is the first to propose a practical scheme for Privacy-preserving Attribute-driven Community Searches on the cloud, named as PACS. PACS enables servers to efficiently respond to attribute-driven community searches in near-millisecond time, without accessing sensitive information about the attributed graph and search results. To achieve this, we design two structures, a secure community index and a secure edge table, for protecting the privacy of the original attributed graph. The secure community index enables cloud servers to efficiently identify the target community that meets structural cohesiveness and has the highest attribute score. In particular, we employ inner product encryption to evaluate the attribute-driven scores of communities based on encrypted attribute vectors. The secure edge table, constructed by BGN homomorphic encryption, allows cloud servers to securely retrieve the edge information of the target community without knowing its details. We perform a thorough security analysis that demonstrates PACS achieves CQA2-security. Experimental evaluations on real-world social network datasets show that PACS achieves near-millisecond efficiency in processing attribute-driven community searches.**

## I. Introduction

Attributed graphs are widely adopted to represent complex real-world networks, as they effectively capture rich semantic information in networks [1], [2]. Entities in the network are abstracted as vertices in the attributed graph, while connections between entities are abstracted as edges between vertices. The characteristics of these entities, such as user identifiers in the social network, are described as vertex attributes in the form of keywords. Attributed graphs are applicable to a wide range of domains, including social networks [3], [4], citation networks [2], and biology networks [5], [6].

Attribute-driven community search aims to identify communities that satisfy specific structural cohesiveness and have attributes correlated with the search attributes [7], [8]. It has been widely adopted in various recommendation systems [9]–[11] due to its ability to provide meaningful and personalized results for search users. For instance, it can use product attributes in a social network to identify potential user groups for targeted marketing [3], [12] or use research themes in a citation network to locate research teams focused on a specific field for academic collaboration [13]–[15].

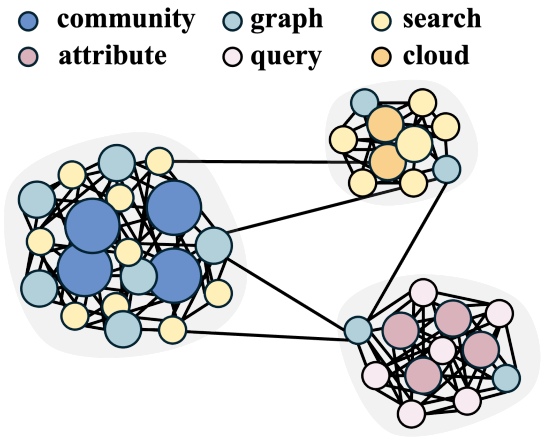



Fig. 1: An illustration of a citation network.

In the citation network illustrated in Fig. 1, the attributes represent the research topics of researchers. It is a core component of attributed graphs, directly reflecting the characteristics of the original data and the needs of search users. When "community" and "search" are specified as search attributes, the research group (i.e., community) on the left side intuitively looks like a suitable candidate for this attribute-driven community search. This observation comes from two facts within this community: (1) the strong collaborative relationships among

✉ Corresponding author: Jia Yu

This work was completed during the first author's visit to the Singapore University of Technology and Design.

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.241586
www.ndss-symposium.org

researchers (structural cohesiveness) and (2) a higher number of researchers focusing on "community" and "search" topics (attribute correlation). As real-world networks continue to expand, data owners have tended to outsource large-scale graphs representing these networks to powerful cloud servers [16], [17]. This not only reduces the burden of local storage to lower costs but also provides users with a more efficient search service.

However, the privacy of both the data owner and search users would be compromised if the graph were directly outsourced to cloud servers [18]–[20]. To address this issue, Chase et al. [21] proposed the concept of graph encryption. They use cryptographic primitives to encrypt the graph, thereby safeguarding the privacy of graphs while enabling efficient searches. Based on graph encryption, numerous privacy-preserving schemes have been proposed for different graph searches, such as reachability search [22], [23], shortest distance search [24]–[28] and subgraph search [29]–[33]. Recently, researchers have proposed several schemes for privacy-preserving community search [34]–[36]. However, those works are designed for unattributed graphs and thus fall short in capturing both the structural and semantic aspects necessary for attributed community search.

What will happen if we directly extend those existing community search works to the attributed version? While it is possible to extend the techniques in [34], [35] to support attributed community search by incorporating a large number of complex symmetric homomorphic operations on encrypted graphs, such an extension would significantly hinder the cloud server's ability to process queries efficiently. A secure tree index was proposed in [36], which significantly improves the efficiency of minimum community search. However, this scheme still ignores attribute information and can only retrieve the vertices of the target community, which limits its practicality in real community search scenarios. For attributed graphs, Sun et al. [37] proposed an efficient privacy-preserving similar community search scheme. Their scheme enables cloud servers to securely retrieve communities that are similar to a given query community in both structure and attribute characteristics, focusing on community–community relationships. Essentially, it matches communities based on the internal structural and attribute features within each community. However, attribute-driven community search aims to identify communities that are most relevant to a user-specified set of query attributes, focusing on query–community relevance. Due to this fundamental difference in search problems, the techniques used in [37] are not suitable for constructing an efficient privacy-preserving attribute-driven community search scheme.

To perform privacy-preserving attribute-driven community search, two key privacy requirements must be satisfied. The first is to protect attribute privacy while still allowing efficient evaluation of how well community attributes match user-specified preferences, that is, the correlations between query attributes and community attributes. Additionally, since the target community embodies the user's preferences, it is essential to securely return its complete details to the user. The scheme must support the secure and efficient retrieval and return of the full information of the target community without exposing any sensitive data.

To address the above challenges, we propose the first Privacy-preserving Attribute-driven Community Search over attributed graphs, named PACS. Our contributions are as follows.

- To ensure the privacy of the attributed graph, particularly its attribute information, we design a secure community index. We employ Asymmetric Inner Product Encryption to encode vertex attributes and search attributes as encrypted vectors. The correlation among community attributes and search attributes is quantified as the attribute-driven score by computing the inner product of two encrypted vectors. During the above evaluation process, cloud servers do not gain any valid information about the attributes of the graphs and users. Using the secure community index, cloud servers can securely and efficiently identify the target community that satisfies the structural requirement and achieves the highest attribute-driven scores.
- To ensure the privacy of the target community, we design a secure edge table based on BGN homomorphic encryption. By performing homomorphic multiplication on the secure edge table, cloud servers can securely retrieve and output the edges of the target community without knowing any detailed information about it. During the search process, cloud servers are unable to access any meaningful information about the target community, including its size, vertices, edges, or core number.
- In PACS, cloud servers can securely and efficiently conduct attribute-driven community searches without accessing sensitive information about the original data, search users, or search results. We analyze the security of PACS based on the CQA2-security model, demonstrating its resilience against adaptive chosen-query attacks. Experiments conducted on real datasets confirm that PACS efficiently answers attribute-driven community searches.

**Organization:** The remainder of this paper is structured as follows. In Section 2, we outline related work. Section 3 presents the preliminaries of our scheme. In Section 4, we formally define the problem addressed in this paper and present the system definition, system model, and threat model. In Section 5, we first introduce the structures of the indexes designed in our scheme and then provide the detailed implementation of PACS. The detailed security analysis and experimental evaluation are given in Sections 6 and 7, respectively. Finally, our conclusion and future work are presented in Section 8.

## II. Related work

### A. Attribute-driven Community Search

Attribute-driven community search emphasizes that the target community should exhibit a strong correlation with the attributes specified by the search user.

Fang et al. [38] quantified this correlation by calculating the number of search attributes shared among the vertices of the community. They proposed a community search scheme based on the CL-tree to identify the $k$-core community that contains the search vertex and has the most shared search attributes. Huang et al. [6] defined an attribute relevance score to measure the correlation between community attributes and search attributes. They proposed a scheme that constructs an attributed truss index for the original graph to search for the closest $k$-truss community containing the search vertex and having the highest score. These schemes identify the community containing the search vertex while satisfying specified attribute requirements. This type of attribute-driven community search has limited real-world applicability because its search scope is confined around the search vertex. For attribute-driven community search without search vertex, Zhang et al. [39] proposed a keyword-centric community search scheme that utilizes a keyword closeness function to measure the distance between the community vertex and the search attribute. They designed a core-based inverted index to search for the $k$-core community with the minimal keyword closeness. Chen et al. [40] proposed a contextual community search scheme. Given a set of search attributes, this scheme returns a community with structural and contextual cohesiveness by defining a contextual score and contextual density. The scheme proposed by Zhu et al. [10] aims to search for a size-constrained $k$-truss community that contains all the search attributes in the attribute graph. Ye et al. [13] developed a top-$r$ keyword-based community search that considers both the semantic similarity of search attributes with community attributes and the semantic similarity among vertices within the community. This type of attribute-driven community search without the search vertex can provide more candidate communities that satisfy the attribute requirements of search users. However, the attribute evaluation metrics employed in these schemes are computationally complex and time-consuming.

### B. Privacy-preserving Graph Search

The goal of privacy-preserving graph search is to secure private information in graphs through anonymity techniques or cryptographic techniques while maintaining the ability to perform searches over graphs [22], [24], [25], [29], [30].

In recent years, privacy-preserving community search has emerged as a popular topic in this field. Guan et al. [34] proposed a privacy-preserving scheme for community searches on bipartite graphs. Their scheme employs symmetric homomorphic encryption (SHE) [41] to construct two secure tables from the bipartite graph. Cloud servers determine the $(\alpha, \beta)$-community containing the search vertices by performing homomorphic operations on these secure tables. Guan et al. [35] designed a privacy-preserving $k$-truss community search scheme by constructing a secure table-based index using SHE. The cloud server traverses this index to identify the $k$-truss community containing the search vertex by employing secure logic gates built on SHE. In [36], the authors focused on the minimum community search problem and proposed an effective privacy-preserving scheme. They utilize BGN homomorphic encryption with obfuscation techniques to generate two secure and efficient indexes. By searches on these indexes, cloud servers can identify the approximate minimum $k$-core community containing the search vertex without knowing any sensitive information. Sun et al. [37] focused on the community-community similarity and proposed a privacy-preserving scheme that enables cloud servers to find the closest similar community. The scheme uses the center vertex to "package" the entire community and builds three secure indexes to store the structural and attribute features of communities. It then employs Bloom filters and a set of secure protocols based on these indexes to compute the similarities and distances among communities. Different from the above research, this paper explores privacy-preserving attribute-driven community search and proposes a practical scheme that supports secure and efficient evaluation of query–community relevance.

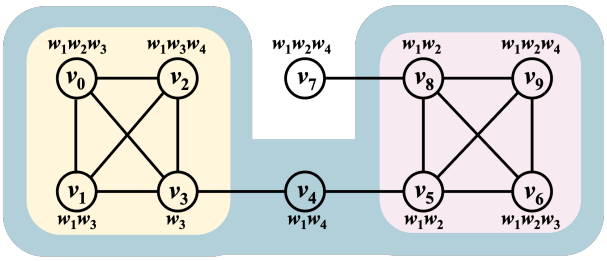



Fig. 2: An attributed graph G.

## III. Preliminaries

In this section, we will define an attributed graph and related concepts. Then, we give the cryptographic tools that we used for building PACS, including an asymmetric inner product encryption and BGN homomorphic encryption.

### A. Attributed Graph

*Definition 1:* (Attributed Graph) An attributed graph is defined as $G = (V, E, W)$, where $V$ is the set of vertices, $E$ is the set of edges, and $W$ is the set of attributes associated with the vertices.

For each vertex $v \in V$, the set of attributes associated with $v$ is denoted by $attr(v) \subseteq W$. For an attribute $w \in W$, we say the vertex $v$ covers $w$ if $w \in attr(v)$. The set of vertices covering $w$ in $G$ is denoted by $V_w \subseteq V$. We first introduce two metrics related to the quality of a community in an attributed graph, structural cohesiveness and attribute correlation.

*1) Structural Cohesiveness:* The structural cohesiveness is used to evaluate the strength of the connections between vertices within a community. We adopt the most commonly used metric in the research of community search, the $k$-core model [9], [42]–[45]. A $k$-core community is formally defined as follows.

*Definition 2:* ($k$-core community) Given an integer $k \geq 0$, a $k$-core community $C$, denoted by $C = (V(C), E(C), W(C))$,

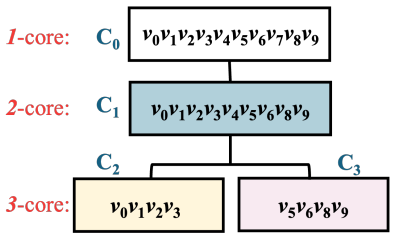


Fig. 3: The core tree index of G.

is a connected subgraph of $G$ in which each vertex $v \in V(C)$ has a degree $deg_C(v)$ that satisfies $deg_C(v) \geq k$.

The value $k$ is referred to as the core number of the community $C$. Due to the nesting property, all $k$-core communities of $G$ can be organized into a tree-like structure known as the core tree index [12], [46].

**Example 1.** *Given an attributed graph $G$, as shown in Fig. 2, the core tree index of $G$ is depicted in Fig. 3. There are four communities $C_0, C_1, C_2$ and $C_3$. Since $G$ is a connected graph, $C_0$ is a 1-core community that includes all the vertices of $G$. The community $C_1 = G/\{v_7\}$ is a 2-core community, while $C_2 = \{v_0, v_1, v_2, v_3\}$ and $C_3 = \{v_5, v_6, v_8, v_9\}$ are two 3-core communities.*

*2) Attribute Correlation:* Attribute correlation measures the similarity between the attribute set of a community and a specified attribute set. We adopt the attribute-driven score defined in [6], [47] to quantify this correlation. A higher attribute-driven score signifies a greater similarity between two sets. The attribute-driven score is defined as follows.

*Definition 3:* (Attribute-Driven Score) Given a community $C$ of $G$ and a non-empty attribute set $W' \subseteq W$, the attribute-driven score of $C$ is defined as $f(C, W') = \sum_{w \in W'} \frac{|V_w \cap V(C)|^2}{|V(C)|}$, where $|V_w \cap V(C)|$ is the number of vertices covering the attribute $w$ in $C$ and $|V(C)|$ is the number of vertices in $C$.

**Example 2.** *Consider the attributed graph $G$ with $W = \{w_1, w_2, w_3, w_4\}$, as shown in Fig. 2. For an attribute set $W' = \{w_1, w_3\}$, we have $f(C_2, W') = \frac{3^2|_{w_1} + 4^2|_{w_3}}{4} = 6.25$ and $f(C_3, W') = \frac{4^2|_{w_1} + 1^2|_{w_3}}{4} = 4.25$. If $W' = \{w_2, w_4\}$, then $f(C_2, W') = \frac{1^2|_{w_2} + 1^2|_{w_4}}{4} = 0.5$ and $f(C_3, W') = \frac{4^2|_{w_2} + 1^2|_{w_4}}{4} = 4.25$.*

### B. Cryptographic Tools

*1) AIPE:* **A**symmetric **I**nner **P**roduct **E**ncryption supports the computation of the inner product on encrypted vectors [48]. AIPE comprises the following four algorithms.

- **AIPE.Setup** $(1^\lambda, 1^t) \rightarrow (pp, msk)$: Given a security parameter $\lambda$ and vector length $t$, this algorithm generates the public parameter $pp = N$ and the master security key $msk = (\delta, h, M_1, M_2, \vec{x})$. $N$ is defined as $N = pq$, where $p$ and $q$ are two large $\lambda$-bit primes. $\delta = lcm(p-1, q-1)$ and $h = h_0^{2N} \mod N^2$ with $h_0 \in \mathbb{Z}^*_{N^2}$. $M_1$ and $M_2$ are two invertible $t \times t$ matrices over $\mathbb{Z}_N$, and $\vec{x}$ is a random vector of length $2t$, where each $x_{1 \leq i \leq 2t} \in [1, \delta N/2]$.
- **AIPE.EncP** $(msk, \vec{p}) \rightarrow [\vec{p}]$: Given the master secret key $msk$ and a data vector $\vec{p} = [p_1, p_2, ...p_t]^T \in \mathbb{Z}_N^t$, this algorithm computes

$$\begin{cases} [\hat{p_1}, \hat{p_2}, ...\hat{p_t}] = \vec{p}^T \cdot M_1 \mod N, \\ [\hat{p}_{t+1}, \hat{p}_{t+2}, ..., \hat{p}_{2t}] = \vec{p}^T \cdot M_2 \mod N. \end{cases} \quad (1)$$

This algorithm selects a random value $r \xleftarrow{R} \mathbb{Z}_N$, and computes

$$\begin{cases} C_0 = h^r \mod N^2, \\ C_{1 \leq i \leq 2t} = (1 + \hat{p}_i N) \cdot h^{r \cdot s_i} \mod N^2. \end{cases} \quad (2)$$

The algorithm finally outputs $[\hat{p}] = [C_0, C_1, ..., C_{2t}]$.
- **AIPE.EncQ** $(msk, \vec{q}) \rightarrow \langle \hat{q} \rangle$: Given the master secret key $msk$ and a query vector $\vec{q} = [q_1, q_2, ...q_t]^T \in \mathbb{Z}_N^t$, this algorithm first randomly splits $\vec{q}$ into two vectors $\vec{q_a}$ and $\vec{q_b}$ such that $\vec{q} = \vec{q_a} + \vec{q_b} \mod N$ and computes

$$\begin{cases} [\hat{q_1}, \hat{q_2}, ...\hat{q_t}] = \vec{q_a}^T \cdot (M_1^{-1})^T \mod N, \\ [\hat{q}_{t+1}, \hat{q}_{t+2}, ..., \hat{q}_{2t}] = \vec{q_b}^T \cdot (M_2^{-1})^T \mod N. \end{cases} \quad (3)$$

Let $b = \sum_{i=1}^{2t} s_i \cdot \hat{q_i} \in \mathbb{Z}$. Define

$$\begin{cases} K_0 = b \mod \delta, \\ K = \hat{q_i}. \qquad 1 \leq i \leq 2t \end{cases} \quad (4)$$

The algorithm finally outputs $\langle \vec{q} \rangle = [K_0, K_1, ..., K_{2t}]$.
- **AIPE.IP** $(pp, [\vec{q}], \langle \hat{q} \rangle) \rightarrow \vec{p}^T \vec{q}$: Given the public parameter $pp$, an encrypted data vector $[\vec{q}]$ and an encrypted query vector $\langle \vec{q} \rangle$, the algorithm outputs the inner product of $[\vec{q}]$ and $\langle \vec{q} \rangle$. It computes

$$C_{\vec{q}^T \vec{p}} = C_0^{-K_0} \cdot \prod_{i=1}^{2t} C_i^{K_i} \mod N^2 \quad (5)$$

and

$$\vec{q}^T \vec{p} = \frac{C_{\vec{q}^T \vec{p}} - 1 \mod N^2}{N}. \quad (6)$$

*2) BGN:* **B**oneh-**G**oh-**N**issim homomorphic encryption [49], [50] supports an unlimited number of homomorphic additions and a single homomorphic multiplication. BGN encryption comprises the following three algorithms.

- **BGN.Setup**$(1^\lambda) \rightarrow (pk, sk)$: Given a security parameter $\lambda$, this algorithm generates the public key $pk$ and the private key $sk$. It first employs a composite bilinear parameter generator $\mathcal{CG}en(\lambda)$ to obtain the system parameters $(N, \mathbb{G}, \mathbb{G}_T, g, e)$, where $N = pq$, $g$ is a generator of $\mathbb{G}$ with order $N$ and $e$ is a bilinear map $\mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. Let $h = g^q$. Then $h$ is a random generator of the subgroup of $\mathbb{G}$ with order $p$. The algorithm finally outputs $pk = (N, \mathbb{G}, \mathbb{G}_T, g, e, h)$ and $sk = p$.
- **BGN.Enc**$(pk, m) \rightarrow [\![m]\!]$: Given the public key $pk$ and a message $m \in \{0, 1, ..., T\}$, where $2T \ll q$, this algorithm encrypts $m$ as follows:

$$[\![m]\!] = g^m h^r \in \mathbb{G}, \qquad r \xleftarrow{R} \mathbb{Z}_N. \quad (7)$$

- **BGN.Dec** $(sk, [\![m]\!]) \rightarrow m$: Given the private key $sk$ and a ciphertext $[\![m]\!]$, the algorithm decrypts $[\![m]\!]$ as follows. It first computes

$$[\![m]\!]^p = (g^m h^r)^p = (g^p)^m. \quad (8)$$

Let $\hat{g} = g^p$. Recover $m$ by computing the discrete logarithm of $[\![m]\!]^p$ base $\hat{g}$.

Given two messages $m_1$,$m_2$ and their ciphertexts $[\![m_1]\!] \in \mathbb{G}$ and $[\![m_2]\!] \in \mathbb{G}$, BGN has the following homomorphic properties:

- $[\![m_1]\!] \cdot [\![m_2]\!] = [\![m_1 + m_2]\!] \in \mathbb{G}$.
- $[\![m_1]\!]^{m_2} = [\![m_1 m_2]\!]$.
- $e([\![m_1]\!], [\![m_2]\!]) = [\![m_1 m_2]\!] \in \mathbb{G}_T$.

## IV. Definitions and Models

### A. Problem Definition

In this paper, we consider a generalized search scenario for attributed graphs, where the community search is solely guided by search attributes without specified search vertices. This attribute-driven community search enables a thorough exploration of valuable information within the attributed graph [13], [39], [40]. The problem addressed in this paper is defined as follows.

*Definition 4:* (Attribute-Driven Community Search) Given an attributed graph $G = \{V, E, W\}$, a non-empty attribute set $W' \subseteq W$ and an integer $\theta > 0$, an attribute-driven community search $q = (W', \theta)$ returns a community $C$ that satisfies the following conditions:

- Structural Cohesiveness: The core number of $C$ is at least $\theta$;
- Attribute Correlation: The attribute-driven score $f(C, W')$ satisfies $f(C, W') > 0$ and is maximized among all possible communities that meet condition (1).

**Example 3.** *Continuing* **Example 2**, *given an attribute-driven community search* $q = (\{w_1, w_3\}, 3)$, *there are two communities* $C_2$ *and* $C_3$, *whose core numbers are* $\geq 3$. *Since* $f(C_2, W') > f(C_3, W')$, $C_2$ *is returned as the result of* $q$.

Next, taking the privacy requirements into account, we define a privacy-preserving, attribute-driven community search scheme as follows.

*Definition 5:* The PACS scheme is defined as $\prod_{\text{PACS}} = \{\text{KeyGen, EncIndex, GenToken, Search, Decrypt}\}$, comprising five polynomial-time algorithms that operate as follows.

- $K \rightarrow$ **KeyGen** $(\lambda, t)$ is a key generation algorithm that takes a security parameter $\lambda$ and a vector length $t$ as input and outputs a key set $K = \{k_1, k_2, (pk, sk), (pp, msk)\}$.
- $(CI, ET) \rightarrow$**EncIndex**$(K, cIndex, eTable)$ is a secure index generation algorithm. This algorithm takes a key set $K$, a community index $cIndex$ and an edge table $eTable$ as input and outputs a secure community index $CI$ and a secure edge table $ET$.
- $Token \rightarrow$**GenToken**$(K, \vec{s}, \theta)$ is a token generation algorithm that takes a key set $K$, a search attribute vector $\vec{s}$ and an integer $\theta$ as input and outputs a search token $Token = \{[\vec{s}], [\![\theta]\!]\}$.
- $R \rightarrow$**Search**$(CI, ET, Token)$ is a privacy-preserving search algorithm that takes a secure community index $CI$, a secure edge table $ET$ and a search token $Token$ as input and outputs an encrypted edge table $R$.

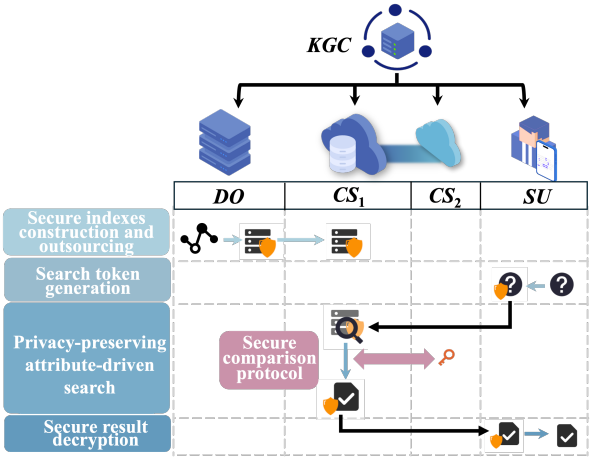



Fig. 4: System model.

- $edge \rightarrow$**Decrypt**$(K, R)$ is a decryption algorithm that takes a key set $K$ and an encrypted edge table $R$ as input and outputs an edge table $edge$.

### B. System Model

In our system model, as shown in Fig. 4, there are four types of participants: the key generation center ($KGC$), the data owner ($DO$), the search user ($SU$) and the cloud servers ($CS_1$ and $CS_2$).

- **Key Generation Center**: $KGC$ is responsible for generating and distributing keys to other participants.
- **Data Owner**: $DO$ processes the original graph data and is responsible for constructing secure indexes based on this data. After generating the secure indexes, $DO$ transmits them to $CS_1$. Once the transmission is finished, $DO$ can switch to an offline state.
- **Search User**: When an $SU$ wants to perform an attribute-driven community search, they first generate a search token based on the search requirements. The token is then transmitted to $CS_1$. After receiving an encrypted result from $CS_1$, the $SU$ uses its own keys to decrypt it and obtain the target community.
- **Cloud Server**s: $CS_1$ is responsible for storing secure indexes. Upon receiving tokens from the $SU$, $CS_1$ performs privacy-preserving attribute-driven community searches using these indexes. During the search, a secure protocol is executed between $CS_1$ and $CS_2$. In this protocol, $CS_2$, which holds the secret key of the BGN, is responsible for decrypting and re-encrypting intermediate results. With the assistance of $CS_2$, $CS_1$ ultimately obtains the encrypted search result and sends it back to the $SU$.

### C. Threat Model

The threat model we adopt is consistent with existing privacy-preserving graph search schemes [25], [36], [51]. In the threat model, $KGC$, $DO$ and $SU$ are assumed to be fully trusted. As the generator and distributor of keys, $KGC$ is resistant to all types of attacks and has secure channels

for transmitting keys to other participants. Adversaries cannot eavesdrop on or tamper with any keys by attacking $KGC$. As the owner of the original data, $DO$ is trusted to honestly construct and encrypt the indexes. These secure indexes are transmitted securely to $CS_1$. As the initiator of searches, the $SU$ has no incentive to tamper with or falsify the search tokens, as doing so would prevent them from achieving their original search objectives. The $SU$ can honestly generate the search tokens and securely transmit them to $CS_1$. All three participants are honest, as any deviation would undermine their own objectives.

The cloud servers $CS_1$ and $CS_2$ are assumed to be semi-honest and non-collusive participants. They cannot leak or tamper with the data, but they may be curious about it. Their curiosity extends not only to the original data and indexes but also to the intermediate data generated during the privacy-preserving search. They will attempt to infer private information about the data or users based on this intermediate data. $CS_1$ and $CS_2$ are expected to honestly execute the search process and securely return the results to the $SU$. The non-collusive assumption ensures that $CS_1$ and $CS_2$ operate independently without sharing information to analyze or infer sensitive data. This assumption is reasonable since $CS_1$ and $CS_2$ can be operated by independent cloud providers or administrative domains (e.g., different commercial cloud platforms), so they will not collude with each other due to conflicting interests [25], [26], [34].

## V. PACS Construction

In this section, we first provides a brief introduction to the high-level framework of PACS. Subsequently, we introduce the fundamental data structures of our scheme and provide a detailed explanation of the construction of PACS.

### A. High-level Overview

To ensure the privacy of attribute information, PACS uses AIPE to transform both community attributes and search attributes into encrypted attribute vectors. $CS_1$ efficiently evaluates the attribute-driven score of a community by invoking a secure inner product algorithm AIPE.IP on encrypted vectors. This design ensures that cloud servers learn nothing about the attribute information of either the attributed graph or the search user while still being able to assess the correlation between community attributes and search attributes. Another challenge PACS addresses is how to retrieve the edges of the target community without exposing them to cloud servers. To achieve this, PACS utilizes BGN homomorphic encryption to construct an encrypted edge vector for each community and an encrypted edge table for the attributed graph. By performing homomorphic multiplication operations between the edge vector of the target community and the $eTable$, PACS securely extracts the edges belonging to the target community. Edges not in the target community are effectively filtered out, as they are multiplied by $[\![0]\!]$. This design ensures that cloud servers learn nothing about the edge information of the target community.

Based on the above designs, PACS constructs a secure community index and a secure edge table to enable cloud servers to efficiently respond to privacy-preserving attribute-driven community searches. The overall framework is illustrated in Fig. 5.

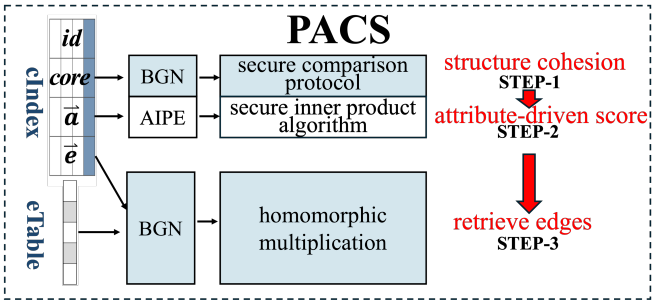



Fig. 5: The framework of PACS.

### B. Index Construction

Given an attributed graph $G$ with $m$ vertices, $n$ edges, and $t$ attributes, we construct an edge table $eTable$ and a community index $cIndex$ to replace $G$ outsourced to $CS_1$.

*1) Edge Table $eTable$:* The edge table $eTable$ is an array of length $n$, which stores the edge information of $G$. For each edge $(u, v) \in G$, it is randomly stored in the $eTable$. The $eTable$ corresponding to the attributed graph shown in Fig. 2 is displayed in Fig. 6.

*2) Community Index $cIndex$:* The community index $cIndex$ is a two-dimensional array that stores the community information in $G$. Each row of the $cIndex$ records the identifier $id$, the core number $core$, the attribute and the edge information of a community in $G$. Given a community $C$, its edge information is represented as a vector of length $n$, denoted as $\vec{e}$. For each $0 \leq i < n$, $\vec{e}_i$ is set to 1 if the edge stored in $eTable[i]$ is contained in $C$.

The attribute information of $C$ is represented as a vector of length $t$, denoted as $\vec{a}$. For each $0 \leq i < t$, $\vec{a}_i$ records the number of vertices in $C$ that cover the attribute $w_i$. Similarly, for a search attribute set $S$, we can represent it as a vector $\vec{s}$ of length $t$, consisting only of 0s and 1s. For an attribute $w_i (0 \leq i < t)$, $\vec{s}_i$ is set to 1 if $w_i \in S$; otherwise, it is set to 0. Furthermore, the number of vertices in $C$ that cover $w_i$ can be calculated as $|V_{w_i} \cap V(C)| = \vec{a}_i \vec{s}_i$.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $v_5 \| v_9$ | $v_8 \| v_9$ | $v_0 \| v_3$ | $v_1 \| v_3$ | $v_6 \| v_9$ | $v_1 \| v_2$ | $v_7 \| v_8$ | $v_0 \| v_1$ | $v_4 \| v_5$ | $v_5 \| v_6$ | $v_0 \| v_2$ | $v_5 \| v_8$ | $v_6 \| v_8$ | $v_2 \| v_3$ | $v_3 \| v_4$ |

Fig. 6: eTable for G.

**Example 4.** *Reconsidering **Example 2**, we have the attribute vectors of $C_1$, $C_2$ and $C_3$ are $[8,5,5,3]$,$[3,1,4,1]$ and $[4,4,1,1]$, respectively. For the search attribute set $S = \{w_1, w_3\}$, $\vec{s} = [1,0,1,0]$. The number of vertices in $C_2$ that cover $w_1$ is caculated as $|V_{w_1} \cap V(C2)| = (C_2.\vec{a}_1)\vec{s}_1 = 3 \times 1 = 3$.*

| id | core | $\vec{a}$ | $\vec{e}$ |
|---|---|---|---|
| $C_2$ | 3 | [2.25 0.25 4.00 0.25] | [0 0 1 1 0 1 0 1 0 0 1 0 0 1 0] |
| G | 1 | [8.10 3.60 2.50 0.40] | [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1] |
| $C_1$ | 2 | [7.11 2.78 2.78 1.00] | [1 1 1 1 1 1 0 1 1 1 1 1 1 1 1] |
| $C_3$ | 3 | [4.00 4.00 0.25 0.50] | [1 1 0 0 1 0 0 0 0 1 0 1 1 0 0] |

Fig. 7: cIndex for G.

To simplify the computation of the attribute-driven score for a community, we preprocess its attribute vector. Given a community $C$ with $z$ vertices, the attribute vector $\vec{a} = [a_0, a_1, ..., a_{t-1}]$ of $C$, and a search attribute vector $\vec{s} = [s_0, s_1, ..., s_{t-1}]$, we have

$$\begin{aligned} f(C,S) &= \sum_{i=0}^{t-1} \frac{(a_i s_i)^2}{z} \\ &= \sum_{i=0}^{t-1} \frac{(a_i s_i)^2}{z} \\ &= \frac{(a_0 s_0)^2 + (a_1 s_1)^2 + ... + (a_{t-1} s_{t-1})^2}{z} \\ &= \frac{{a_0}^2 s_0 + {a_1}^2 s_1 + ... + {a_{t-1}}^2 s_{t-1}}{z} \qquad //\vec{s} \in [0,1]^t \\ &= \frac{{a_0}^2}{z} s_0 + \frac{{a_1}^2}{z} s_1 + ... + \frac{{a_{t-1}}^2}{z} s_{t-1} \\ &= [\frac{{a_0}^2}{z}, \frac{{a_1}^2}{z}, ..., \frac{{a_{t-1}}^2}{z}] \cdot [s_0, s_1, ..., s_{t-1}] \\ &= \overrightarrow{(a^2/z)} \cdot \vec{s}, \end{aligned}$$

where $\cdot$ represents the inner product operation.

According to the above conclusion, we preprocess all attribute vectors $\vec{a}$ into $\overrightarrow{a^2/z}$, which are then stored in $cIndex$. By doing this, the attribute-driven score of a community can be efficiently obtained by computing the inner product of two vectors. For clarity and simplicity, we continue to use $\vec{a}$ in the remainder of this paper to represent the processed vectors.

**Example 5.** *Continue considering $G$ shown in Fig. 2. The processed attribute vector of $C_2$ is $\vec{a} = [2.25, 0.25, 4.00, 0.25]$ as $|V(C_2)| = 4$. The community index $cIndex$ of $G$ is illustrated in Fig. 7.*

### C. Complete Scheme Construction

In this section, we present the details and complete construction of PACS. We begin by defining a pseudo-random function (PRF) and a pseudo-random permutation (PRP) as follows.

$$PRF: \{0,1\}^\lambda \leftarrow \{0,1\}^\lambda \times \{0,1\}^*,$$
$$PRP: \{0,1\}^* \leftarrow \{0,1\}^\lambda \times \{0,1\}^*,$$

where $\lambda$ ia a security parameter.

Then, we describe each algorithm used in PACS as follows.

- **KeyGen.** Given a security parameter $\lambda$ and vector length $t$, $KGC$ employs **Algorithm 1** to generate a key set $K$. The keys $k_1$ and $k_2$ are used for generating PRF and PRP, respectively. This algorithm invokes BGN.KeyGen($\lambda$) to generate a key pair $(pk, sk)$ for the BGN encryption. Additionally, AIPE.KeyGen($\lambda, t$) is invoked to generate the public parameter $pp$ and the master security key $msk$ for AIPE.
  After obtaining all keys, $KGC$ distributes them to the various participants. $DO$ receives $\{k_1, k_2, pk, msk\}$ while the $SU$ is provided with $\{k_2, (pk, sk), msk\}$. $CS_1$ obtains $\{pk, pp\}$ and $CS_2$ obtains $\{sk\}$.

**Algorithm 1:** KeyGen

**Input:** A security parameter $\lambda$ and the vector length $t$;
**Output:** A key set $K$;

```
k_1 ←$ {0,1}^λ;
k_2 ←$ {0,1}^λ;
(pk, sk) ←BGN.KeyGen(λ);
(pp, msk) ←AIPE.KeyGen(λ, t);
return K = {k_1, k_2, (pk, sk), (pp, msk)}.
```

- **EncIndex.** After constructing $cIndex$ and $eTable$, $DO$ employs **Algorithm 2** to encrypt them and obtain a secure community index $CI$ and a secure edge table $ET$.
  Lines 3-8 describe the process for encrypting $cIndex$ to obtain the $CI$. This algorithm employs a PRF to conceal the community identifier $id$ for each community in $cIndex$. The core number $core$ of each community is encrypted using BGN encryption (line 5). This algorithm utilizes AIPE.EncP($msk, \vec{a}$) to encrypt the attribute vector $\vec{a}$, producing $[\vec{a}]$. Then, this algorithm invokes BGN.Enc($pk, \vec{e}$) to encrypt the edge vector $\vec{e}$. $[\![\vec{e}]\!]$ denotes the encrypted edge vector, where each bit is individually encrypted using the BGN encryption. To further enhance the security of our scheme, the information $(id', [\![core]\!], [\vec{a}], [\![\vec{e}]\!])$ of each community is randomly assigned to a row in the $CI$. This effectively obfuscates the nested relationships among communities in the graph. The process of encrypting $eTable$ is detailed in lines 9-12 of **Algorithm 2**. For each edge $(u||v)$ stored in $eTable$, this algorithm first utilizes the PRP function to conceal the actual information of the edge. BGN.Enc($pk, (u||v)'$) is invoked to prevent cloud servers from accessing the edge information. Finally, the encrypted edge $[\![u||v]\!]$ is added into the $ET$ in order.
- **GenToken.** When $SU$ wants to initiate an attribute-driven community search request, he/she utilizes **Algorithm 3** to generate a search token $Token$ and send it to $CS_1$. Given a search attribute vector $\vec{s}$, $SU$ employs AIPE.EncQ($msk, \vec{s}$) to encrypt it, generating $[\vec{s}]$. BGN.Enc($pk, \theta$) is performed to encrypt the core number constraint $\theta$, where $0 < \theta \leq k_{max}$. The algorithm finally outputs an encrypted search token $Token = \{[\vec{s}], [\![\theta]\!]\}$.

**Algorithm 2:** EncIndex

**Input:** A key set $K$, a community index $cIndex$ and an edge table $eTable$.
**Output:** A secure community index $CI$ and a secure edge table $ET$.

parse $K$ as $\{k_1, k_2, pk, msk\}$;
initialize two empty dictionaries $CI$ and $ET$;
// encrypt $cIndex$
**for** *each community in* $cIndex$ **do**
  $id' \leftarrow$PRF$(k_1, id)$;
  $[\![core]\!] \leftarrow$ BGN.Enc$(pk, core)$;
  $[\vec{a}] \leftarrow$ AIPE.EncP$(msk, \vec{a})$;
  $[\![\vec{e}]\!] \leftarrow$ BGN.Enc$(pk, \vec{e})$;
  $(id', [\![core]\!], [\vec{a}], [\![\vec{e}]\!])$ is randomly assigned to a row in $CI$;
// encrypt $eTable$
**for** *each edge* $u||v$ *in* $eTable$ **do**
  $(u||v)' \leftarrow PRP(k_2, u||v)$;
  $[\![u||v]\!] \leftarrow$ BGN.Enc$(pk, (u||v)')$;
  add $[\![u||v]\!]$ into $ET$;
**return** $(CI, ET)$.

**Algorithm 3:** GenToken

**Input:** A key set $K$, a search attribute vector $\vec{s}$ and an integer $\theta$.
**Output:** A search token $Token$.

parse $K$ as $\{pk, msk\}$;
$[\vec{s}] \leftarrow$ AIPE.EncQ$(msk, \vec{s})$;
$[\![\theta]\!] \leftarrow$BGN.Enc$(pk, \theta)$;
**return** $Token = \{[\vec{s}], [\![\theta]\!]\}$.

- **Search.** Before giving the details of **Search** algorithm, we first design a secure integer comparison protocol based on BGN encryption, as shown in **Protocol 1**. This protocol enables $CS$s to perform core number filtering on the encrypted data. Given two ciphertexts $[\![m_1]\!]$ and $[\![m_2]\!]$ encrypted using BGN encryption, the protocol **Com**$([\![m_1]\!], [\![m_2]\!])$ outputs a flag $f$ that indicates the relationship between $m_1$ and $m_2$.
  When $f = 0$, $CS_1$ concludes that $m_1 \geq m_2$. Otherwise, $m_1 < m_2$. In the process of the protocol, $CS_1$ obtains only the comparison result $f$ without learning the actual values of $m_1$ and $m_2$. Meanwhile, $CS_2$ is aware solely of the need to perform a comparison operation and does not gain any knowledge about $m_1$,$m_2$,$[\![m_1]\!]$,$[\![m_2]\!]$ or $f$.
  After receiving a search token $Token$ from an $SU$, $CS_1$ utilizes **Algorithm 4** to perform a privacy-preserving attribute-driven community search with the assistance of $CS_2$. This algorithm consists of three steps: core number filtering, attribute-driven score calculation, and result return. An empty table $R$ of length $n$ is first initialized to store the edge information of the search result. In STEP-1, this algorithm filters out communities whose core numbers do not meet the constraint $\theta$ specified by the $SU$ (lines 3-6). An empty dictionary $F$ is initialized to store candidate communities whose core numbers are equal to or greater than $\theta$. For each community $CI[i]$, this algorithm employs the protocol $Com()$ to determine whether its core number satisfies the constraint. After obtaining all candidate communities, this algorithm proceeds to STEP-2, invoking AIPE.IP to compute the attribute-driven scores for these communities (line 9). The community with the highest score in $F$ is selected as the target community for this attribute-driven community search. The edge vector of the target community is stored in $[\vec{g}]$ (line 10). Once the target community is determined before STEP-3, this algorithm retrieves the edge information of the target community. For $0 \leq i < n-1$, this algorithm performs the homomorphic multiplication operation of BGN encryption, $e([\vec{g}_i], ET[i])$ (line 12). Finally, the encrypted edge table $R$, which stores all the encrypted edges of the target community, is returned.
- **Decrypt.** After receiving the encrypted edge table $R$ from $CS_1$, the $SU$ employs **Algorithm 5** to decrypt it. For each entry in $R$, this algorithm successively calls

**Protocol 1:** Com$([\![m_1]\!], [\![m_2]\!])$

// $CS_1$
$[\![m_1']\!] \leftarrow [\![m_1]\!]^2 \cdot [\![1]\!]$;
$[\![m_2']\!] \leftarrow [\![m_2]\!]$;
select $b \in \{0, 1\}$ and $r \xleftarrow{R} \mathbb{Z}_N$;
**if** *b==1* **then**
  $[\![l]\!] \leftarrow ([\![m_1']\!] \cdot [\![m_2']\!]^{-1})^r$;
**else**
  $[\![l]\!] \leftarrow ([\![m_1']\!]^{-1} \cdot [\![m_2']\!])^r$;
send $[\![l]\!]$ to $CS_2$
// $CS_2$
$l \leftarrow$BGN.Enc$(sk, [\![l]\!])$;
**if** $l > T$ **then**
  $f' \leftarrow 1$;
**else**
  $f' \leftarrow 0$;
send $f'$ to $CS_1$;
// $CS_1$
**if** *b==1* **then**
  $f \leftarrow f'$;
**else**
  $f \leftarrow 1 - f'$;

**Algorithm 4:** Search

**Input:** A secure community index $CI$, a secure edge table $ET$ and a search token $Token$.
**Output:** An encrypted table $R$.

parse $Token$ as $\{[\vec{s}], [\![\theta]\!]\}$;
initialize an empty table $R$ of length $n$;
// STEP-1: Core Number Filtering
initialize an empty dictionary $F$;
**for** $i$ *from* 0 *to* $\delta$ **do**
  **if** $Com(CI[i].[\![core]\!], [\![\theta]\!])$ **then**
    add $CI[i]$ to $F$;
// STEP-2: Attribute-Driven Score Calculation
initialize an edge vector $[\vec{g}]$;
**for** *each community* $C$ *in* $F$ **do**
  $f(C, [\vec{s}]) \leftarrow$AIPE.IP$(pp, [\vec{s}], C.[\vec{a}])$;
$[\vec{g}] \leftarrow C.[\vec{e}]$, where $C$ is the community with the highest $f$;
// STEP-3: Result Return
**for** $i$ *from* 0 *to* $n-1$ **do**
  $R[i] \leftarrow e([\vec{g}]_i, ET[i])$;
**return** $R$.

**Algorithm 5:** Decrypt

**Input:** A secret key set $K$ and an encrypted edge table $R$.
**Output:** An edge table $edge$.

parse $K$ as $\{k_2, sk\}$;
initialize an empty table $edge$;
**for** $i$ *form* $0$ *to* $n-1$ **do**
  $(u||v)' \leftarrow$BGN.Dec$(sk, R[i])$;
  $u||v \leftarrow$PRP$^{-1}(k_2, (u||v)')$;
  **if** $u||v \neq 0$ **then**
    add $u||v$ into $edge$;
**return** $edge$.

BGN.Dec$(sk, R[i])$ and the inverse function of PRP, i.e., PRP$^{-1}(k_2, (u||v)')$, to obtain the edge $u||v$. If $u||v \neq 0$, it indicates that the edge $u||v$ is part of the target community. This algorithm ultimately returns all edges contained in the target community, denoted as $edge$.

## VI. Security Analysis

In this paper, we adopt the CQA2-security model, which is commonly used for evaluating the security of privacy-preserving graph search schemes [30], [36], [51].

We first define two leak functions, $\mathcal{L}_1$ and $\mathcal{L}_2$, to describe the information leaked from the encrypted data and the search procedures, respectively.

- $\mathcal{L}_1$: This function is defined to capture the information leaked from the encrypted index $CI$ and the encrypted table $ET$, $\mathcal{L}_1 = \{n, t, \delta\}$. $n$ represents the length of $ET$ and the number of edges in $G$. $t$ represents the length of the attribute vector, i.e., the number of attributes in $G$. $\delta$ denotes the length of the $CI$, which indicates the number of communities in $G$.
- $\mathcal{L}_2$: This function is defined to capture the leaked information during the search procedures. $\mathcal{L}_2$ consists of query pattern leakage and index pattern leakage. In the **GenToken** algorithm, AIPE.EncQ$(msk, \vec{s})$ randomly splits the search attribute vector $\vec{s}$, while BGN.Enc$(pk, \theta)$ encrypts the core number constraint $\theta$ with a random number. These methods ensure that the query pattern does not reveal whether a search has been repeated. It only discloses information about repeated index entries accessed during previous searches, denoted as $info_1$. The index pattern discloses information associated with a search request, denoted as $\{info_2, ore, num, list\}$. $info_2$ represents information of index entries accessed during a search. $ore$ represents the order information between the community core number and the constraint. $num$ denotes the number of candidate communities whose core numbers satisfy the constraint. $list$ is a list of length $num$, storing attribute-driven scores for all candidate communities.

*Definition 6:* (CQA2-Security) Let $\prod_{\text{PACS}}$ ={**KeyGen**, **EncIndex**, **GenToken**, **Search**, **Decrypt**} be a privacy-preserving attribute-driven community search scheme. Given an adversary $\mathcal{A}$, a simulator $\mathcal{S}$, and two leakage functions $\mathcal{L}_1$ and $\mathcal{L}_2$, the following two probabilistic experiments are defined.

1) **Real**$_{\prod,\mathcal{A}}(\lambda)$**:**
   - The challenger runs KeyGen$(\lambda, t)$ to generate a key set $K$, which is sent to $\mathcal{A}$. Then the challenger executes EncIndex $(K, cIndex, eTable)$ to get an encrypted community index $CI$ and an encrypted edge table $ET$, which are then sent to $\mathcal{A}$.
   - $\mathcal{A}$ adaptively submits a polynomial number of attribute-driven community searches. For each search, the challenger executes GenToken$(K, \vec{s}, \theta)$ to get a search token $Token$ and sends it to $\mathcal{A}$.

   Finally, $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$.
2) **Ideal**$_{\prod,\mathcal{A},\mathcal{S}}(\lambda)$**:**
   - Given $\mathcal{L}_1$, $\mathcal{S}$ simulates an encrypted community index $CI^*$ and an encrypted edge table $ET^*$ and sends them to $\mathcal{A}$.
   - $\mathcal{A}$ adaptively submits a polynomial number of attribute-driven community searches. For each search, $\mathcal{S}$ simulates a search token $Token^*$ based on $\mathcal{L}_2$ and sends it to $\mathcal{A}$.
   - Finally, $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$.

The scheme $\prod_{\text{PACS}}$ is $(\mathcal{L}_1, \mathcal{L}_2)$-secure against adaptive chosen-query attacks if for all probabilistic polynomial time (PPT) adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ and a negligible function negl$(\lambda)$ such that

$$|\Pr[\textbf{Real}_{\prod,\mathcal{A}}(\lambda) = 1] - \Pr[\textbf{Ideal}_{\prod,\mathcal{A},\mathcal{S}}(\lambda) = 1]| \leq \text{negl}(\lambda).$$

*Theorem 1:* The proposed scheme $\prod_{\text{PACS}}$ is $(\mathcal{L}_1, \mathcal{L}_2)$-secure against adaptive chosen-query attacks if cryptographic primitives PRP, PRF, AIPE and BGN are secure.

**Proof.** We conclude the proof by constructing the simulator $\mathcal{S}$. Given the leakage functions $\mathcal{L}_1$ and $\mathcal{L}_2$, $\mathcal{S}$ simulates a dummy index $CI^*$, a dummy table $ET^*$ and a sequence of searches $q^*$. For all PPT adversaries $\mathcal{A}$, if they cannot distinguish between **Real** experiment and **Ideal** experiment, $\prod$ is considered to be $(\mathcal{L}_1, \mathcal{L}_2)$-secure against the adaptive chosen-query attacks.

1) Simulating secure indexes: Given $\mathcal{L}_1$, $\mathcal{S}$ constructs a dummy community index $CI^*$ and a dummy edge table $ET^*$ as follows. The length of $CI^*$ is $\delta$ and each entry of $CI^*$ is generated using dummy information and the fake key set $K^*$. Specifically, for each community, $\mathcal{S}$ samples $id \xleftarrow{R} \{0, 1\}^\lambda$ and randomly selects an integer as its $core$. $\mathcal{S}$ simulates an attribute vector $\vec{a}$ of length $t$, where each element is a random integer and an edge vector $\vec{e}$ of length $n$, where each element is randomly chosen from $\{0, 1\}$. The dummy edge table $ET^*$ is simulated with length $n$, based on dummy vertex pairs and the fake key set $K^*$. For each entry, $\mathcal{S}$ randomly selects a vertex pair $(u||v)$ and encrypts it using the fake key set $K^*$.
2) Simulating search token : Let $q^* = \{q_1^*, q_2^*, ..., q_j^*\}$ be a sequence of searches. For each search $q_i^* = (\vec{s}_i^*, \theta_i^*) \in q^*$, $\mathcal{S}$ proceeds as follows. $\mathcal{S}$ generates a dummy search token $Token_i^*$ based on a random attribute vector $\vec{s}^*$ of length $2t$, a random core number constraint $\theta^*$ and the fake key set $K^*$.

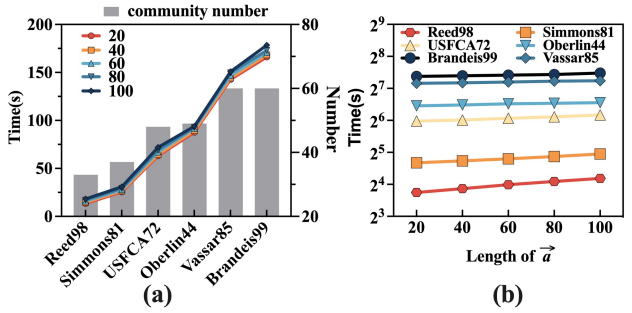



Fig. 8: Construction time of CI.

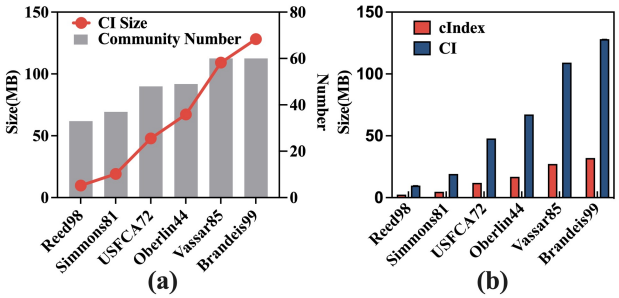



Fig. 9: CI size.

3) Simulating search: Given $\mathcal{L}_2$, $\mathcal{S}$ simulates the search procedure as follows. For each search token $Token^* = \{[\vec{s}]^*, [\![\theta]\!]^*\}$, $\mathcal{S}$ ensures the generated information during the search is consistent with the information provided by the leakage function $\mathcal{L}_2$.

Since PRP, PRF, AIPE, and BGN are secure, the dummy index $CI^*$, the dummy table $ET^*$ and the search sequence $q^*$ are indistinguishable from the real ones. Thus we have

$$|\Pr[\mathbf{Real}_{\prod,\mathcal{A}}(\lambda)=1]-\Pr[\mathbf{Ideal}_{\prod,\mathcal{A},\mathcal{S}}(\lambda)=1]|\leq \mathrm{negl}(\lambda),$$

where $\mathrm{negl}(\lambda)$ stands for a negligible function.

## VII. Performance Evaluation

In this section, we evaluate the efficiency of PACS using real network datasets.

TABLE I: Dataset statistics.

| Dataset | Number of Vertices | Number of Edges | Maximum core number | Number of Communities |
|---|---|---|---|---|
| Reed98 | 962 | 18812 | 35 | 33 |
| Simmons81 | 1519 | 32989 | 35 | 37 |
| USFCA72 | 2683 | 65253 | 44 | 48 |
| Oberlin44 | 2921 | 89913 | 50 | 49 |
| Vassar85 | 3069 | 119162 | 61 | 60 |
| Brandeis99 | 3899 | 137568 | 57 | 60 |

### A. Dataset

We select six datasets from the Network Repository[1], a publicly available and comprehensive collection of graph data. These datasets are extracted from Facebook, where each vertex represents a user and each edge represents a friendship tie between two users. The statistical details of the datasets are summarized in Table I. For each dataset, we randomly generate five attribute sets consisting of 20, 40, 60, 80 and 100 different attributes, respectively. In reality, communities are typically composed of members who share common or similar attributes. Therefore, in each attribute set, we randomly select five attributes for each community and ensure that each attribute is covered by 80% of the community's vertices. Finally, each vertex is additionally assigned 1 to 5 random attributes to simulate noise.

[1]https://networkrepository.com

### B. Setup

We adopt the method described in [12] to construct a core tree index for each dataset. The security parameter $\lambda$ is set to 512. We use the SHA3-512 implementation from the Crypto++ library[2] for the PRP and the PRF. Additionally, we utilize FLINT library[3], GMP library[4], Libhcs library[5] and Pairing-Based Cryptography library[6] to implement AIPE encryption, BGN homomorphic encryption. $DO$ and $SU$ are simulated on a personal computer equipped with an Apple M2 processor and 16 GB of RAM, running macOS Sonoma 14.4. The cloud servers ($CS_1$ and $CS_2$) are simulated on two Alibaba Cloud ECS instances with 2 vCPUs and 8 GiB of RAM, running Ubuntu 22.04.

### C. EncIndex Performance

In this section, we first compare the storage complexity of PACS with that of a non-privacy-preserving scheme [10], which supports similar attribute-driven community searches over attributed graphs.

Consider an attributed graph $G$ with $m$ vertices, $n$ edges and $t$ attributes. In the scheme proposed in [10], both the truss index and keyword index have a storage complexity of $\mathcal{O}(n)$, leading to an overall storage requirement of $\mathcal{O}(n)$. In PACS, $CI$ comprises $\delta$ quaternion entries of the form $(id', [\![core]\!], [\vec{a}], [\![\vec{e}]\!])$. $[\vec{a}]$ is an attribute vector of length $2t+1$, encrypted using AIPE.EncP. $[\![\vec{e}]\!]$ is an edge vector of length $n$, where each element is encrypted using BGN.Enc. Consequently, the storage complexity of $CI$ is $\mathcal{O}(\delta \cdot (t+n))$. The secure edge table $ET$ has a storage complexity of $\mathcal{O}(n)$, as it stores $n$ encrypted edge representations in the form $[\![u||v]\!]$. Thus, the total index size of PACS is $\mathcal{O}(\delta \cdot (t+n))$. While PACS introduces additional storage overhead compared to the scheme in [10], this is a necessary tradeoff to achieve the desired security guarantees. Moreover, in real-world networks, the number of communities $\delta$ and attributes $t$ are typically much smaller than the number of edges $n$.

[2]www.cryptopp.com
[3]https://flintlib.org/
[4]https://gmplib.org/
[5]https://tiehu.is/libhcs/
[6]https://crypto.stanford.edu/pbc/

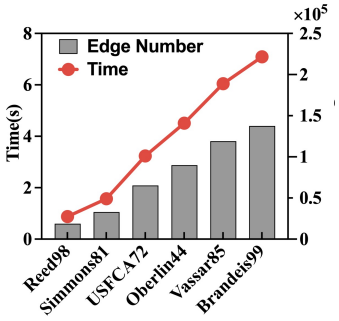



Fig. 10: Time for ET.

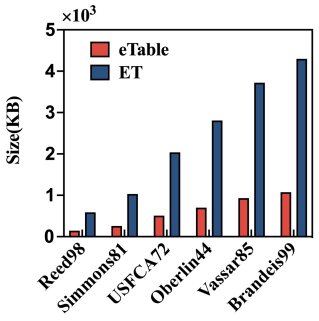



Fig. 11: Size of ET.

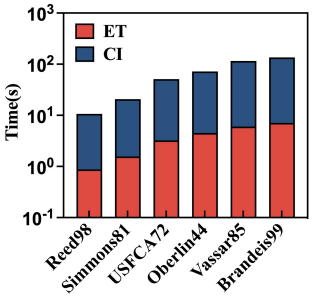



Fig. 12: Time-1 for EncIndex.

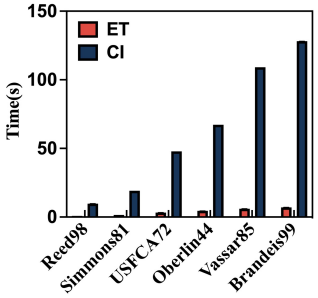



Fig. 13: Time-2 for EncIndex.

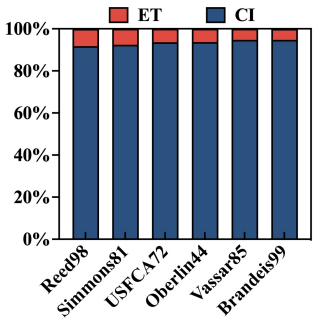



Fig. 14: Fraction of time.

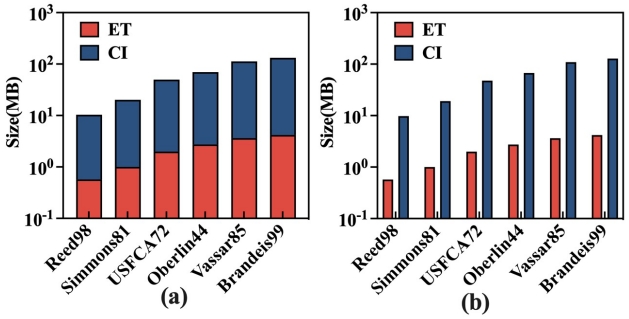



Fig. 15: Storage cost of EncIndex.

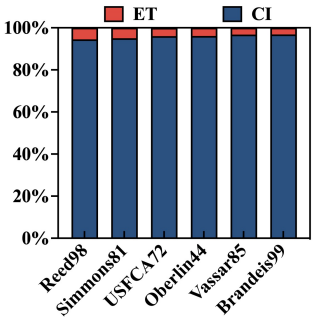



Fig. 16: Fraction of storage cost.

TABLE II: The size of cIndex and CI (MB).

| | | Reed98 | Simmons81 | USFCA72 | Oberlin44 | Vassar85 | Brandeis99 |
|---|---|---|---|---|---|---|---|
| t=20 | cIndex | 2.4429 | 4.7853 | 11.9523 | 16.8108 | 27.2793 | 32.0170 |
| | CI | 9.7715 | 19.1409 | 47.8091 | 67.2431 | 109.1169 | 128.0677 |
| t=40 | cIndex | 2.4455 | 4.7882 | 11.9560 | 16.8145 | 27.2839 | 32.0216 |
| | CI | 9.7819 | 19.1525 | 47.8237 | 67.2580 | 109.1352 | 128.0863 |
| t=60 | cIndex | 2.4481 | 4.7911 | 11.9596 | 16.8183 | 27.2884 | 32.0263 |
| | CI | 9.7923 | 19.1641 | 47.8384 | 67.2730 | 109.1535 | 128.1050 |
| t=80 | cIndex | 2.4507 | 4.7940 | 11.9633 | 16.8220 | 27.2930 | 32.0310 |
| | CI | 9.8027 | 19.1757 | 47.8530 | 67.2879 | 109.1719 | 128.1236 |
| t=100 | cIndex | 2.4533 | 4.7969 | 11.9670 | 16.8258 | 27.2976 | 32.0356 |
| | CI | 9.8130 | 19.1873 | 47.8677 | 67.3029 | 109.1902 | 128.1422 |

TABLE III: The size of eTable and ET (MB).

| | Reed98 | Simmons81 | USFCA72 | Oberlin44 | Vassar85 | Brandeis99 |
|---|---|---|---|---|---|---|
| eTable | 0.14 | 0.25 | 0.50 | 0.69 | 0.91 | 1.05 |
| ET | 0.57 | 1.01 | 1.99 | 2.74 | 3.64 | 4.20 |

*1) Secure community index $CI$:* For each community in $cIndex$, $EncIndex$ algorithm employs PRF, BGN and AIPE to encrypt the identifier, core number, attribute vector and edge vector. The construction time of $CI$ is primarily influenced by the number of communities in the dataset. A higher number of communities results in more entries generated in $CI$, leading to a longer construction time. The conclusion is summarized in Fig. 8(a). The second factor that affects $CI$ construction time is the lengths of the attribute vector and the edge vector, both of which are positively correlated with $CI$ construction time, as illustrated in Fig. 8(b). For the same dataset, the construction time of $CI$ exhibits a slight upward trend as the horizontal coordinate (the length of the attribute vector $t$) increases. The length of the edge vector is determined by the edges contained in the dataset. As shown in Fig. 8(b), for a fixed $t$, the construction time of $CI$ increases significantly with the number of edges in the dataset.

For the same dataset, the size of $CI$ depends solely on the size of the attribute set, which determines the length of the encrypted attribute vector in $CI$. Table II provides details of $CI$ size, showing that the attribute set size has minimal impact. An increase of 20 attributes results in an average $CI$ size

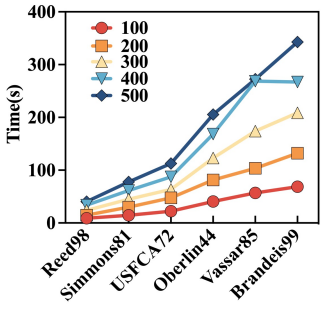



Fig. 17: t=20.

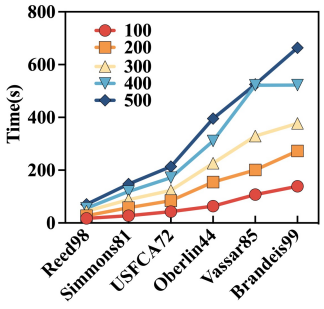



Fig. 18: t=40.

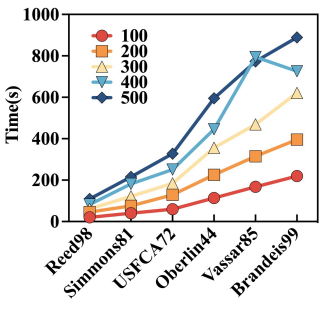



Fig. 19: t=60.

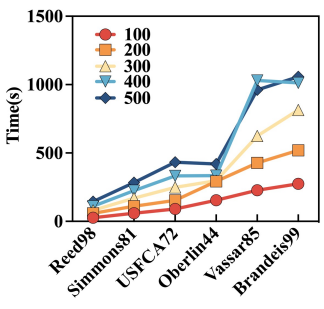



Fig. 20: t=80.

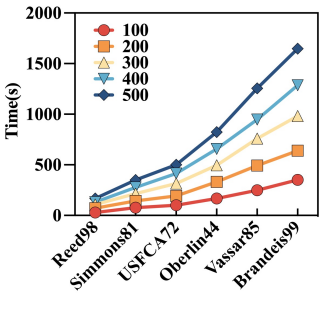



Fig. 21: t=100.

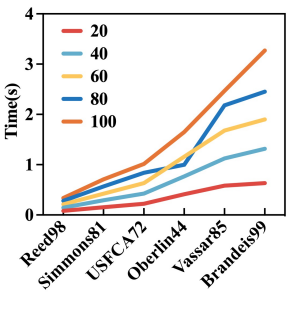



Fig. 22: Average search time.

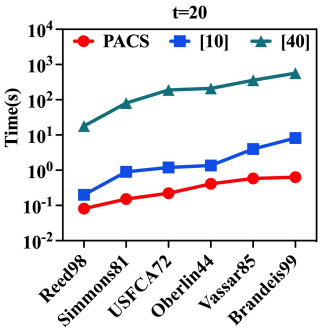



Fig. 23: t=20.

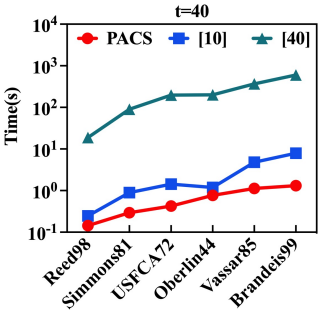



Fig. 24: t=40.

increase of only 0.015MB for each dataset. Fig. 9(a) visually depicts the relationship between the average $CI$ index size and the number of communities. Fig. 9(b) and Table II present a comparison of the size of $cIndex$ and $CI$. Due to the use of BGN and AIPE, the size of the encrypted index $CI$ is approximately four times larger than that of the unencrypted index $cIndex$.

*2) Secure edge table $ET$:* For each edge $u||v$ in $eTable$, $EncIndex$ algorithm secures it sequentially using PRP and BGN. The construction time of $ET$, as illustrated in Fig. 10, depends solely on the number of edges in the dataset. The detailed size information of $eTable$ and $ET$ is presented in Fig. 11 and Table III. In contrast to $CI$, $ET$ is more space-efficient, with a maximum size of 4.20MB and a minimum size of 0.57MB. As shown in Fig. 11 and Table III, the size of $ET$ is approximately four times larger than that of $eTable$.

In **EncIndex** algorithm, constructing $CI$ is significantly more complex and time-consuming than constructing $ET$, as shown in Figures 12-16. In both construction time and storage cost, $CI$ accounts for 90% or more. It is worth noting that the construction of $ET$ and $CI$ is a one-time process completed in the system initialization phase. Therefore, its performance is not a primary concern in practical deployments and does not affect the usability of PACS.

### D. Search Performance

We first analyze and compare the search complexity of PACS with that of two non-privacy-preserving attribute-driven community search schemes [10], [40].

The scheme in [40] introduces both an exact algorithm and an approximate algorithm for identifying the minimal contextual community based on triangle density and edge density. The search complexity of the exact algorithm is $\mathcal{O}((n + Tri(G))^3)$, while that of the approximate algorithm is $\mathcal{O}(m \log m + n \log n + Tri(G))$, where $Tri(G)$ denotes the number of triangles in $G$. The scheme in [10] constructs a truss index and a keyword index to identify the minimal dense truss community containing the search keywords. The keyword index is used to retrieve all components containing the search keywords and the truss index is used to identify the smallest such component as the search result. The search complexity of this scheme is $\mathcal{O}(\log \sqrt{n} p_{max})$, where $p_{max}$ is the maximum number of components in each layer of the truss index. In PACS, the $Search$ algorithm consists of three steps. In STEP-1, the algorithm executes **Protocol 1** $\delta$ times to identify candidate communities. Under the standard cryptographic assumption where the security parameter $\lambda$ is considered constant, each execution of **Protocol 1** has $\mathcal{O}(1)$ complexity. Thus, STEP-1 has total complexity of $\mathcal{O}(\delta)$. Let $num$ denote the number of candidate communities. In STEP-2, the algorithm executes AIPE.IP $num$ times to compute the attribute-driven scores of the candidate communities. Each AIPE.IP operation requires $2t + 1$ modulo exponentiations, resulting in $\mathcal{O}(t)$ complexity per execution. Therefore, the complexity of STEP-2 is $\mathcal{O}(num \cdot t)$. STEP-3 involves $n$ BGN multiplications to obtain the valid edge vector of the target community. Since each multiplication takes $\mathcal{O}(1)$ time when $\lambda$ is fixed, the total complexity is $\mathcal{O}(n)$. Consequently,

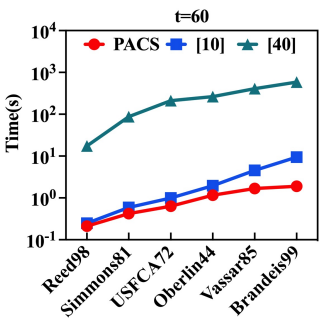



Fig. 25: t=60.

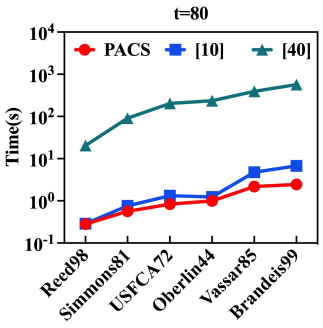



Fig. 26: t=80.

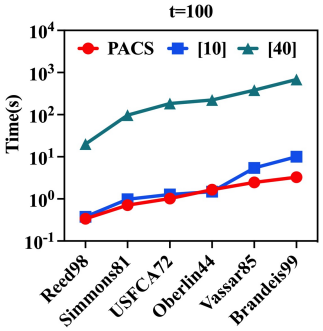



Fig. 27: t=100.

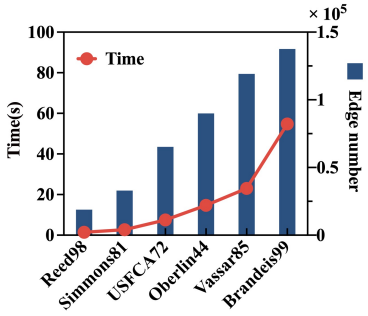



Fig. 28: Time for decryption.

the overall complexity of $Search$ algorithm in PACS is $\mathcal{O}(\delta + num \cdot t + n)$.

Next, we evaluate the efficiency of $Search$ algorithm through experiments. We generate five query sets containing 100, 200, 300, 400, and 500 attribute-driven community searches, respectively. Figures 17-21 illustrate the search time across different datasets with varying lengths of attribute vectors. It is evident that, for the same dataset, the search time increases as the length of the attribute vector grows. Longer attribute vectors mean that $CS_1$ requires more time to invoke AIPE.IP to calculate the community's attribute-driven score.

Fig. 22 displays the average time for a single search across datasets. With a fixed attribute vector length, the search time is correlated with the dataset size, especially the number of edges and the number of communities it contains. The number of communities determines the number of entries in $CI$. A higher number of communities requires $CS_1$ to execute the comparison protocol with the assistance of $CS_2$ more frequently for core number filtering. The number of edges (i.e., the length of the edge vector) determines the number of homomorphic multiplication operations that $CS_1$ performs to retrieve the edge information of the target community. The average time for a single search ranges from a minimum of $0.08s$ to a maximum of $2.44s$. For most datasets, except for the Brandeis99 dataset, our scheme can respond to searches within milliseconds.

We also implement two non-privacy-preserving attribute-driven community search schemes proposed in [10] and [40] for comparison. Although these two schemes address slightly different problems, they both aim to identify communities associated to the query attributes, and thus provide meaningful baselines for evaluating performance. Specifically, we implement the top-down search algorithm in [10], which identifies the minimal dense subgraph containing all query attributes based on pre-built index structures. We also implement the approximate algorithm in [40], which iteratively removes the vertex contributing the least to the contextual score to achieve an approximately optimal result. Since the search efficiency of both algorithms is significantly affected by the number of query attributes, we set the number of query attributes to four for fairness. The average search time under different values of $t$ is shown in Figs. 23–27. As illustrated, PACS achieves substantially faster search performance even compared with these non-privacy-preserving schemes, primarily because they adopt more complex structural and attribute relevance metrics.

### E. Decrypt Performance

The $SU$ employees BGN.Dec and $\text{PRP}^{-1}$ successively to decrypt an encrypted table $R$ of length $n$, thereby obtaining the edges contained in the target community. The average decryption time per search is shown in Fig. 28. The left Y-axis indicates the decryption time, while the right Y-axis represents the number of edges. Apart from the inherent cost of BGN.Dec, the performance of the Decrypt algorithm depends solely on the size of $n$ (i.e., the number of edges in the dataset). As illustrated in Fig. 28, the decryption time increases almost linearly (or even sublinearly) with the number of edges.

The bottleneck of decryption performance primarily lies in the discrete logarithm computation in BGN.Dec. In real-world implementations, the proposed scheme can further improve decryption performance by shortening the edge vectors or adopting faster discrete logarithm algorithms, such as Pollard's rho.

## VIII. Conclusion and Future Work

In this paper, we propose PACS, an effective privacy-preserving scheme for attribute-driven community searches over attributed graphs. PACS enables cloud servers to securely return the community that satisfies the core number constraint and achieves the highest attribute-driven score. Cloud servers can obtain the attribute-driven score of the community without knowing any sensitive information about the attributed graph by using asymmetric inner product encryption on a secure community index. By performing homomorphic multiplication operations on a secure edge table, PACS securely returns the edge information about the target community to the search user. The security analysis and experiments on real datasets demonstrate that PACS achieves CQA2-security and is efficient.

Although PACS enriches the diversity of privacy-preserving community searches, it cannot be directly applied to dynamic graphs. In dynamic graphs, any change to a vertex, edge, or attribute may affect the community structure or overall topology. This requires securely and accurately tracking community

evolution on encrypted indexes to identify which parts of the graph are affected by updates. It is complex to design privacy-preserving community search schemes for dynamic graphs. Therefore, exploring such schemes is a valuable and challenging topic for future research.

## ACKNOWLEDGMENT

This research is supported by National Cryptologic Science Fund of China (2025NCSF02035), the National Natural Science Foundation of China (62172245), Shandong Provincial Natural Science Foundation under Grant(ZR2024MF038) and Chinese Scholarship Council (CSC).

## REFERENCES

[1] Y. Wang, Y. Li, J. Fan, C. Ye, and M. Chai, "A survey of typical attributed graph queries," *World Wide Web*, vol. 24, no. 1, pp. 297–346, 2021.

[2] Y. Zhou, H. Cheng, and J. X. Yu, "Graph clustering based on structural/attribute similarities," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 718–729, 2009.

[3] F. Guo, Y. Yuan, G. Wang, X. Zhao, and H. Sun, "Multi-attributed Community Search in Road-social Networks," in *IEEE International Conference on Data Engineering*, ser. ICDE'21, 2021, pp. 109–120.

[4] T. Lappas, K. Liu, and E. Terzi, "Finding a team of experts in social networks," in *Proceedings of the ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 467–476.

[5] A. L. Hu and K. C. C. Chan, "Utilizing Both Topological and Attribute Information for Protein Complex Identification in PPI Networks," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 10, no. 3, pp. 780–792, 2013.

[6] X. Huang and L. V. S. Lakshmanan, "Attribute-driven community search," *Proceedings of the VLDB Endowment*, vol. 10, no. 9, pp. 949–960, 2017.

[7] Y. Fang, X. Huang, L. Qin, Y. Zhang, W. Zhang, R. Cheng, and X. Lin, "A survey of community search over big graphs," *The VLDB Journal*, vol. 29, no. 1, pp. 353–392, 2020.

[8] J. Yang, W. Yao, and W. Zhang, "Keyword Search on Large Graphs: A Survey," *Data Science and Engineering*, vol. 6, no. 2, pp. 142–162, 2021.

[9] Y. Wang, S. Ye, X. Xu, Y. Geng, Z. Zhao, X. Ke, and T. Wu, "Scalable Community Search with Accuracy Guarantee on Attributed Graphs," in *IEEE International Conference on Data Engineering*, ser. ICDE'24, 2024, pp. 1–15.

[10] Y. Zhu, Q. Zhang, L. Qin, L. Chang, and J. X. Yu, "Cohesive subgraph search using keywords in large networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 178–191, 2020.

[11] S. Amer-Yahia, S. B. Roy, A. Chawlat, G. Das, and C. Yu, "Group recommendation: Semantics and efficiency," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 754–765, 2009.

[12] Y. Fang, R. Cheng, Y. Chen, S. Luo, and J. Hu, "Effective and efficient attributed community search," *The VLDB Journal*, vol. 26, no. 6, pp. 803–828, 2017.

[13] J. Ye, Y. Zhu, and L. Chen, "Top-r keyword-based community search in attributed graphs," in *IEEE International Conference on Data Engineering*, ser. ICDE'23, 2023, pp. 1652–1664.

[14] J. Wang, L. Zhou, X. Wang, L. Wang, and S. Li, "Attribute-sensitive community search over attributed heterogeneous information networks," *Expert Systems with Applications*, vol. 235, pp. 1–49, 2024.

[15] Y. Song, L. Zhou, P. Yang, J. Wang, and L. Wang, "CS-DAHIN: Community Search Over Dynamic Attribute Heterogeneous Network," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–14, 2024.

[16] R. Chen, X. Weng, B. He, and M. Yang, "Large graph processing in the cloud," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 1123–1126.

[17] S. Carlin and K. Curran, "Cloud computing security," in *Pervasive and ubiquitous technology innovations for ambient intelligence environments*. IGI Global Scientific Publishing, 2013, pp. 12–17.

[18] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou, "vsql: Verifying arbitrary sql queries over dynamic outsourced databases," in *IEEE Symposium on Security and Privacy*, ser. S&P'17', 2017, pp. 863–880.

[19] C. Zuo, Z. Lin, and Y. Zhang, "Why does your data leak? uncovering the data leakage in cloud from mobile apps," in *IEEE Symposium on Security and Privacy*, ser. S&P'19, 2019, pp. 1296–1310.

[20] W. C. Garrison, A. Shull, S. Myers, and A. J. Lee, "On the practicality of cryptographically enforcing dynamic access control policies in the cloud," in *IEEE Symposium on Security and Privacy*, ser. S&P'16', 2016, pp. 819–838.

[21] M. Chase and S. Kamara, "Structured Encryption and Controlled Disclosure," in *Annual International Conference on the Theory and Application of Cryptology and Information Security*, ser. ASIACRYPT'10, 2010.

[22] Y. Song, X. Ge, J. Yu, R. Hao, and M. Yang, "Enabling Privacy-Preserving KK-Hop Reachability Query Over Encrypted Graphs," *IEEE Transactions on Services Computing*, vol. 17, no. 3, pp. 893–904, 2024.

[23] S. Yin, Z. Fan, P. Yi, B. Choi, J. Xu, and S. Zhou, "Privacy-Preserving Reachability Query Services," in *International Conference on Database Systems for Advanced Applications*, ser. DASFAA'14, vol. 8421, 2014, pp. 203–219.

[24] M. Shen, B. Ma, L. Zhu, R. Mijumbi, X. Du, and J. Hu, "Cloud-Based Approximate Constrained Shortest Distance Queries Over Encrypted Graphs With Privacy Protection," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 4, pp. 940–953, 2018.

[25] C. Zhang, L. Zhu, C. Xu, K. Sharif, C. Zhang, and X. Liu, "PGAS: Privacy-preserving graph encryption for accurate constrained shortest distance queries," *Information Sciences*, vol. 506, pp. 325–345, 2020.

[26] F. Wang, Z. Chen, L. Pan, L. Y. Zhang, and J. Zhou, "CryptGraph: An Efficient Privacy-Enhancing Solution for Accurate Shortest Path Retrieval in Cloud Environments," in *ACM Asia Conference on Computer and Communications Security*, ser. Asia CCS'24, 2024, pp. 1660–1674.

[27] S. Wang, X. Xiao, Y. Yang, and W. Lin, "Effective indexing for approximate constrained shortest path queries on large road networks," *Proceedings of the VLDB Endowment*, vol. 10, no. 2, pp. 61–72, 2016.

[28] X. Ge, J. Yu, H. Zhang, J. Bai, J. Fan, and N. N. Xiong, "SPPS: A Search Pattern Privacy System for Approximate Shortest Distance Query of Encrypted Graphs in IIoT," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 52, no. 1, pp. 136–150, 2022.

[29] X. Zuo, L. Li, H. Peng, S. Luo, and Y. Yang, "Privacy-Preserving Subgraph Matching Scheme With Authentication in Social Networks," *IEEE Transactions on Cloud Computing*, vol. 10, no. 3, pp. 2038–2049, 2022.

[30] X. Ge, J. Yu, and R. Hao, "Privacy-Preserving Graph Matching Query Supporting Quick Subgraph Extraction," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–15, 2023.

[31] K. Huang, H. Hu, S. Zhou, J. Guan, Q. Ye, and X. Zhou, "Privacy and efficiency guaranteed social subgraph matching," *The VLDB Journal*, vol. 31, no. 3, pp. 581–602, 2022.

[32] Z. Fan, B. Choi, Q. Chen, J. Xu, H. Hu, and S. S. Bhowmick, "Structure-preserving subgraph query services," in *IEEE International Conference on Data Engineering*, ser. ICDE'16. IEEE, 2016, pp. 1532–1533.

[33] Z. Chang, L. Zou, and F. Li, "Privacy preserving subgraph matching on large graphs in cloud," in *ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD'16, 2016, pp. 199–213.

[34] Y. Guan, R. Lu, Y. Zheng, S. Zhang, J. Shao, and G. Wei, "Achieving Efficient and Privacy-Preserving ($\alpha$,$\beta$)-Core Query over Bipartite Graphs in Cloud," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 3, pp. 1–15, 2022.

[35] Y. Guan, R. Lu, S. Zhang, Y. Zheng, J. Shao, and G. Wei, "kTCQ: Achieving Privacy-Preserving k-Truss Community Queries Over Outsourced Data," *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 4, pp. 2750–2765, Jul. 2024.

[36] F. Sun, J. Yu, and J. Hu, "Privacy-Preserving Approximate Minimum Community Search on Large Networks," *IEEE Transactions on Information Forensics and Security*, pp. 1–14, 2024.

[37] F. Sun, J. Yu, and J. Hu, "Privacy-preserving closest similar community search on attributed graphs," *IEEE Transactions on Information Forensics and Security*, vol. 20, pp. 6662–6677, 2025.

[38] Y. Fang, R. Cheng, S. Luo, and J. Hu, "Effective community search for large attributed graphs," *Proceedings of the VLDB Endowment*, vol. 9, no. 12, pp. 1233–1244, 2016.

[39] Z. Zhang, X. Huang, J. Xu, B. Choi, and Z. Shang, "Keyword-Centric Community Search," in *IEEE International Conference on Data Engineering*, ser. ICDE'19, 2019, pp. 422–433.
[40] L. Chen, C. Liu, K. Liao, J. Li, and R. Zhou, "Contextual Community Search Over Large Social Networks," in *IEEE International Conference on Data Engineering*, ser. ICDE'19. IEEE, 2019, pp. 88–99.
[41] 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 of Things Journal*, vol. 7, no. 6, pp. 5220–5232, 2020.
[42] C. Li, F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin, "Efficient progressive minimum k-core search," *Proceedings of the VLDB Endowment*, vol. 13, no. 3, pp. 362–375, 2019.
[43] L. Oettershagen, A. L. Konstantinidis, and G. F. Italiano, "Temporal Network Core Decomposition and Community Search," Sep. 2023.
[44] W. Luo, Q. Yang, Y. Fang, and X. Zhou, "Efficient Core Maintenance in Large Bipartite Graphs," *Proceedings of the ACM on Management of Data*, vol. 1, no. 3, pp. 1–26, 2023.
[45] A. Tian, A. Zhou, Y. Wang, X. Jian, and L. Chen, "Efficient Index for Temporal Core Queries over Bipartite Graphs," *Proceedings of the VLDB Endowment*, vol. 17, no. 11, pp. 2813–2825, 2024.
[46] N. Barbieri, F. Bonchi, E. Galimberti, and F. Gullo, "Efficient and effective community search," *Data Mining and Knowledge Discovery*, vol. 29, no. 5, pp. 1406–1433, 2015.
[47] Y. Teng, D. Jiang, M. Sun, L. Zhao, L. Xu, and C. Fan, "Privacy-Preserving Top-k Spatio-Textual Similarity Join," in *IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, ser. TrustCom'22, 2022, pp. 718–726.
[48] Z. Zhang, K. Wang, C. Lin, and W. Lin, "Secure Top-k Inner Product Retrieval," in *ACM International Conference on Information and Knowledge Management*, ser. CIKM'18, 2018, pp. 77–86.
[49] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-DNF Formulas on Ciphertexts," in *Theory of Cryptography Conference*, ser. TCC'05, vol. 3378, 2005, pp. 325–341.
[50] M. Zhou, Y. Zheng, Y. Guan, L. Peng, and R. Lu, "Efficient and privacy-preserving range-max query in fog-based agricultural IoT," *Peer-to-Peer Networking and Applications*, vol. 14, no. 4, pp. 2156–2170, 2021.
[51] M. Hu, L. Chen, G. Chen, Y. Mu, and R. H. Deng, "A Pruned Pendant Vertex Based Index for Shortest Distance Query Under Structured Encrypted Graph," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 6351–6363, 2024.

## APPENDIX A
## ARTIFACT APPENDIX

This artifact provides the implementation of "PACS: Privacy-Preserving Attribute-Driven Community Search over Attributed Graphs."

The PACS artifact is publicly available on Zenodo (DOI: https://doi.org/10.5281/zenodo.17927374) and can also be accessed on GitHub: https://github.com/sakurasfy/PACS.

### A. Description & Requirements

*1) How to access:* The artifact is provided as a compressed package (`PACS_artifact.zip`) uploaded to the NDSS submission system. It contains a self-contained Docker environment and all scripts to reproduce the results. Please refer to `README.md` for setup and evaluation instructions.

*2) Hardware dependencies:* Standard commodity hardware (desktop or laptop):

- **Minimum**: 2 CPU cores, 4GB RAM
- **Recommended**: 4+ CPU cores, 8GB+ RAM
- **Disk space**: 5GB (for Docker image and datasets)
- **Architecture**: x86_64 or ARM64

*3) Software dependencies:*

- **Docker Engine**: version 20.10 or later
- **Docker Compose**: optional, for convenience
- **Operating System**: Linux or macOS
- **Tested on**: Ubuntu 22.04, macOS (ARM64/Apple Silicon)

*4) Benchmarks:* All six datasets mentioned in the paper have been preprocessed and are provided in the `data/output/` directory. No additional data preparation is required for artifact evaluation. The `graph.h` and `graph.cpp` files are included only for users who wish to process new datasets from raw edge lists in `data/input/`.

### B. Major Claims

- (C1:) The construction time of the secure Community Index (CI) increases with the number of communities in the dataset and, to a lesser extent, with the length of the attribute vector and the number of edges. This claim can be reproduced by observing the terminal outputs of the artifact (Evaluation: Execution-[Results]) and comparing with Fig. 8 of the paper.
- (C2:) The size of the secure Community Index (CI) is primarily determined by the number of attributes. An increase of 20 attributes results in an average CI size increase of about 0.015MB for each dataset. This claim can be reproduced via (Evaluation: Execution-[Results]) and corresponds to Table II and Fig. 9 of the paper.
- (C3:) The construction time and the size of the secure Edge Table (ET) depend solely on the number of edges in the dataset. This claim can be verified via (Evaluation: Execution-[Results]) and corresponds to Fig. 10–11 and Table III.
- (C4:) The runtime of the privacy-preserving attribute-driven community search remains practical on six real-world social network datasets. This claim is reproducible via (Evaluation: Execution-[Results]) and corresponds to Section VII.D of the paper.
- (C5:) The time required for decrypting the search results increases almost linearly (or even sublinearly) with the number of edges $n$. This claim is reproducible via (Evaluation: Execution-[Results]) and corresponds to Section VII.E of the paper.

### C. Evaluation

The artifact automatically runs using the default dataset (Reed98) with 20 attributes. To reduce runtime, it performs 11 privacy-preserving attribute-driven community searches to demonstrate the effectiveness of our scheme.

*[Preparation]*

1) Ensure Docker (version 20.10 or later) is installed on the evaluation machine.
2) Extract the artifact package and navigate to its root directory.
3) The directory already contains the preprocessed datasets in `data/output/`; no additional data preparation is needed.
4) Review the detailed instructions in the provided `README.md`.

*[Execution]*

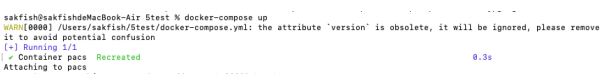
1) Build the Docker image:

```
docker build -t pacs:latest .
```

(First build time: about 5 minutes.)


2) Run the container:

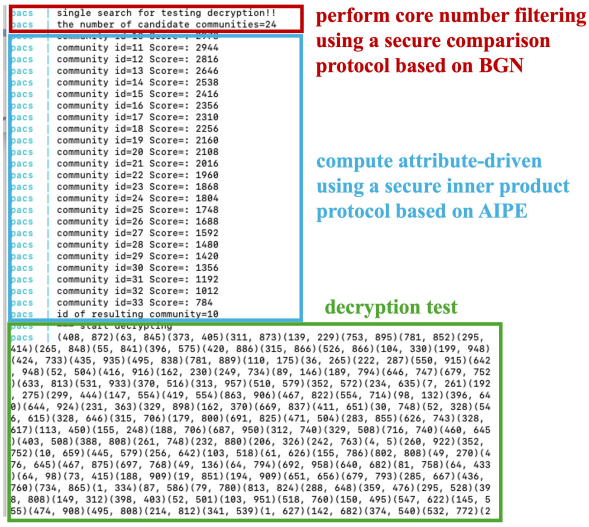
```
docker-compose up
```




3) The system will automatically:
   - Read the preprocessed file from `data/output` for the Reed98 dataset with 20 attributes;
   - Construct the secure Community Index (CI) and secure Edge Table (ET) using AIPE and BGN;

```
pacs  |
pacs  | --- start buildCI
pacs  | buildCI Completed!
pacs  | --- start buildET
pacs  | buildET Completed!
```

- Perform a single search to simulate user-side decryption (for measuring the BGN decryption cost, which takes about one minute);




If no matching community is found during the search, the experiment should be rerun to demonstrate the decryption effect.

- Execute ten privacy-preserving attribute-driven community searches;

```
pacs  | the number of candidate communities=4
pacs  | community id=30 Score=: 6102
pacs  | community id=31 Score=: 5364
pacs  | community id=32 Score=: 4554
pacs  | community id=33 Score=: 3528
pacs  | id of resulting community=30
pacs  | The resulting community of 10-10-th search is 30
pacs  |
```

*[Results]*

All statistical information is displayed directly to the terminal, including:

- Index size before and after encryption;
- Time to construct CI and ET;
- Decryption time (single test);
- Runtime for ten automatic searches.

```
-------------------------------------------------
Dataset: Reed98
Vertex number: 962
Edge number: 18812
Attribute number: 20

time for builtCI (s): 12.8846
time for builtET (s): 0.837774
time for 10 searches (s): 9.58215
single decryption time (s): 79.1819

size of cIndex (MB): 2.443
size of CI (MB): 9.803
size of eTable (MB): 0.144
size of ET (MB): 0.574
```

Observed runtime and index-size values are expected to be comparable to those figures in Section 7 of the paper (within a reasonable variation, e.g., ±20%, depending on hardware and Docker overhead).

### *D. Customization*

To run other datasets, modify the following in the source code and rebuild: $NODES\_NUM$, $EDGES\_NUM$, $ATTRIBUTE\_NUM$, and $Max\_Core$ in `Encrypt.h`, and $datasetname$ in `main.cpp` (line 399). Please refer to `README.md` file.