2011 31st International Conference on Distributed Computing Systems

Privacy-Preserving Query over Encrypted Graph-Structured Data in Cloud Computing

Ning Cao[†], Zhenyu Yang[†], Cong Wang[‡], Kui Ren[‡], and Wenjing Lou[†]
[†]Department of ECE, Worcester Polytechnic Institute, Worcester, MA 01609
Email: {ncao, zyyang, wjlou}@ece.wpi.edu
[‡]Department of ECE, Illinois Institute of Technology, Chicago, IL 60616
Email: {cong, kren}@ece.iit.edu

***Abstract*— In the emerging cloud computing paradigm, data owners become increasingly motivated to outsource their complex data management systems from local sites to the commercial public cloud for great flexibility and economic savings. For the consideration of users' privacy, sensitive data have to be encrypted before outsourcing, which makes effective data utilization a very challenging task. In this paper, for the first time, we define and solve the problem of privacy-preserving query over encrypted graph-structured data in cloud computing (PPGQ), and establish a set of strict privacy requirements for such a secure cloud data utilization system to become a reality. Our work utilizes the principle of "filtering-and-verification". We pre-build a feature-based index to provide feature-related information about each encrypted data graph, and then choose the efficient inner product as the pruning tool to carry out the filtering procedure. To meet the challenge of supporting graph query without privacy breaches, we propose a secure inner product computation technique, and then improve it to achieve various privacy requirements under the known-background threat model.**

I. Introduction

In the increasingly prevalent cloud computing, datacenters play a fundamental role as the major cloud infrastructure providers [1], such as Amazon, Google, and Microsoft Azure. Datacenters provide the utility computing service to software service providers who further provide the application service to end users through Internet. The later service has long been called "Software as a Service (SaaS)", and the former service has recently been called "Infrastructure as a Service (IaaS)", where the software service provider is also referred to as cloud service provider. To take advantage of computing and storage resources provided by cloud infrastructure providers, data owners outsource more and more data to the datacenters [2] through cloud service providers, e.g., the online storage service provider, which are not fully trusted by data owners. As a general data structure to describe the relation between entities, the graph has been increasingly used to model complicated structures and schemaless data, such as the personal social network (the social graph), the relational database, XML documents and chemical compounds studied by research labs [3]–[8]. Images in the personal album can also be modeled as the attributed relational graph (ARG) [9]. For the protection of users' privacy, these sensitive data have to be encrypted before outsourcing to the cloud. Moreover, some data are supposed to be shared among trusted partners. For example, the album owner may share family party photos with only authorized users including family members and friends. For another example, the lab director and members are given the authorization to access the entire lab data. In both cases, authorized users are usually planning to retrieve some portion of data they are interested rather than the entire dataset, mostly because of the "pay-for-use" billing rule in the cloud computing paradigm. Considering the large amount of data centralized in the datacenter, it is a very challenging task to effectively utilize the graph-structured data after encryption.

With the conventional graph data utilization method, we first take the query graph as an input, and then perform the graph containment query: *given a query graph as Q and a collection of data graphs as $\mathcal{G} = (G_1, G_2, \ldots, G_m)$, find all the supergraphs of Q in $\mathcal{G}$, denoted as $\mathcal{G}_Q$*. The straightforward solution is to check whether Q is subgraph isomorphic to every G_i in $\mathcal{G}$ or not. However, checking subgraph isomorphism is NP-complete, and therefore it is infeasible to employ such costly solution. To efficiently solve the graph containment query problem, there have been a lot of proposed techniques [3]–[8], most of which follow the principle of "filtering-and-verification". In the filtering phase, a pre-built feature-based index is utilized to prune as many data graphs from the dataset as possible and output the candidate supergraph set. Every feature in the index is a fragment of a data graph, e.g., the subgraph. In the verification phase, each candidate supergraph is verified by checking subgraph isomorphism. Since the candidate supergraph set is much smaller than the entire dataset, such approach involves less subgraph isomorphism checking, and therefore is significantly more efficient than the straightforward solution. However, when data graphs are stored in the encrypted form in the cloud, the encryption excludes the filtering method which is based on the plaintext index.

In the most related literature, the searchable encryption [10]–[14] is a helpful technique that treats encrypted data as documents and allows a user to securely search over it through specifying single keyword or multiple keywords with Boolean relations. However, the direct application of these approaches to deploy the secure large scale cloud data utilization system would not be necessarily suitable. The keyword-based search provides much less semantics than the graph-based query since the graph could characterize more

1063-6927/11 $26.00 © 2011 IEEE
DOI 10.1109/ICDCS.2011.84

complicated relations than Boolean relation. More importantly, these searchable encryption schemes are developed as crypto primitives and cannot accommodate such high service-level requirements like system usability, user query experience, and easy information discovery in mind. Therefore, how to design an efficient encrypted query mechanism which supports graph semantics without privacy breaches still remains a challenging open problem.

In this paper, for the first time, we define and solve the problem of privacy-preserving graph query in cloud computing (PPGQ). To reduce the times of checking subgraph isomorphism, we adopt the efficient principle of "filtering-and-verification" to prune as many negative data graphs as possible before verification. A feature-based index is firstly built to provide feature-related information about every encrypted data graph. Then, we choose the efficient inner product as the pruning tool to carry out the filtering procedure. To achieve this functionality in index construction, each data graph is associated with a binary vector as a subindex where each bit represents whether the corresponding feature is subgraph isomorphic to this data graph or not. The query graph is also described as a binary vector where each bit means whether the corresponding feature is contained in this query graph or not. The inner product of the query vector and the data vector could exactly measure the number of query features contained in the data graph, which is used to filter negative data graphs that do not contain the query graph. However, directly outsourcing the data vector or the query vector will violate the index privacy or the query privacy. To meet the challenge of supporting graph semantics without privacy breaches, we propose a secure inner product computation mechanism, which is adapted from a secure k-nearest neighbor (*kNN*) technique [15], and then show our improvements on it to achieve various privacy requirements under the known-background threat model. Our contributions are summarized as follows,

1) For the first time, we explore the problem of query over encrypted graph-structured data in cloud computing, and establish a set of strict privacy requirements for such a secure cloud data utilization system to become a reality.

2) Our proposed scheme follows the principle of "filtering-and-verification" for efficiency consideration, and thorough analysis investigating privacy and efficiency guarantees of the proposed scheme is given.

3) The evaluation, which is performed with the widely-used AIDS antiviral screen dataset on the Amazon EC2 cloud infrastructure, further shows our proposed scheme introduces low computation and communication overhead.

The remainder of this paper is organized as follows. In Section II, we introduce the system model, the threat model and our design goals. Section III gives preliminaries, and section IV describes the framework and privacy requirements in PPGQ, followed by section V, which gives our proposed scheme. Section VI presents evaluation results. We discuss related work on both keyword searchable encryption and graph containment query in Section VII, and conclude the paper in Section VIII.

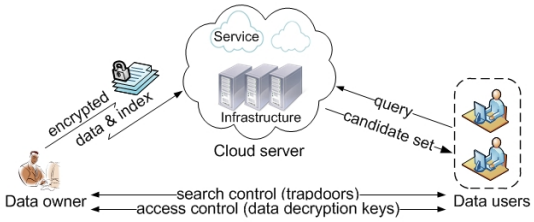


Fig. 1: Architecture of graph query over encrypted cloud data

II. Problem Formulation

A. The System Model

Considering a cloud data storage service, involving four different entities: the data owner, the data user, the storage service provider/cloud service provider, and the datacenter/cloud infrastructure provider. To take advantage of the utility computing services provided by the datacenter, e.g., computing and storage resources, the storage service provider deploys its storage service on top of the utility computing in datacenter and delivers the service to end users (including data owners and data users) through Internet. In our system model, neither cloud service provider nor cloud infrastructure provider is fully trusted by data owners or data users, so they are treated as an integrated entity, named the cloud server, as shown in Fig. 1.

The data owner has a graph-structured dataset $\mathcal{G}$ to be outsourced to the cloud server in the encrypted form $\widetilde{\mathcal{G}}$. To enable the query capability over $\widetilde{\mathcal{G}}$ for effective data utilization, the data owner will build an encrypted searchable index $\mathcal{I}$ from $\mathcal{G}$ before data outsourcing, and then both the index $\mathcal{I}$ and the encrypted graph dataset $\widetilde{\mathcal{G}}$ are outsourced to the cloud server. For every query graph Q, an authorized user acquires a corresponding trapdoor T_Q through the search control mechanism, e.g., broadcast encryption [10], and then sends it to the cloud server. Upon receiving T_Q from data users, the cloud server is responsible to perform query over the encrypted index $\mathcal{I}$ and return the encrypted candidate supergraphs. Finally, data users decrypt the candidate supergraphs through the access control mechanism, and verify each candidate by checking subgraph isomorphism.

B. The Known Background Threat Model

The cloud server is considered as "honest-but-curious" in our model, which is consistent with most related works on searchable encryption [14], [16]. Specifically, the cloud server acts in an "honest" fashion and correctly follows the designated protocol specification. However, it is "curious" to infer and analyze the data and the index in its storage and interactions during the protocol so as to learn additional information. The encrypted data $\widetilde{\mathcal{G}}$ and searchable index $\mathcal{I}$ can be easily obtained by the cloud server, because both of them are outsourced and stored on the cloud server. In addition to these encrypted information, the cloud server is supposed to know some backgrounds on the dataset, such as its subject and related statistical information. As a possible attack similar to that in [17], the cloud server could utilize the feature frequency to identify features contained in the query graph.

C. Design Goals

To enable the graph query for the effective utilization of outsourced cloud data under the aforementioned model, our design should simultaneously achieve security and performance guarantees.

- **Effectiveness:** To design a graph query scheme that introduces few false positives in the candidate supergraph set.
- **Privacy:** To prevent the cloud server from learning additional information over outsourced data and index in query interactions, and to meet privacy requirements specified in section IV-C.
- **Efficiency:** Above goals on effectiveness and privacy should be achieved with low communication and computation overhead.

D. Notations

- $\mathcal{G}$ – the graph-structured dataset, denoted as a collection of m data graphs $\mathcal{G} = (G_1, G_2, \ldots, G_m)$.
- $\widetilde{\mathcal{G}}$ – the encrypted graph-structured dataset outsourced into the cloud, denoted as $\widetilde{\mathcal{G}} = (\widetilde{G}_1, \widetilde{G}_2, \ldots, \widetilde{G}_m)$.
- $\mathrm{id}(G_i)$ – the identifier of the data graph G_i that can help uniquely locate the graph.
- $\mathcal{F}$ – the feature set mined from the graph dataset, denoted as $\mathcal{F} = (F_1, F_2, \ldots, F_n)$.
- $\mathcal{D}$ – the frequent feature dictionary, denoted as $\mathcal{D} = \{\mathcal{L}_{F_1}, \mathcal{L}_{F_2}, \ldots, \mathcal{L}_{F_n}\}$, where $\mathcal{L}_{F_j}$ is the unique canonical label of F_j;
- $\mathcal{I}$ – the searchable index associated with $\widetilde{\mathcal{G}}$, denoted as $(I_1, I_2, \ldots, I_m)$, where each subindex I_i is built from G_i.
- Q – the query graph from the data user.
- $\mathcal{F}_Q$ – the subset of $\mathcal{F}$, consisting of frequent features contained in Q, denoted as $\mathcal{F}_Q = \{F_j | F_j \subseteq Q, F_j \in \mathcal{F}\}$.
- $\mathcal{G}_{\{Q\}}$ – the subset of $\mathcal{G}$, consisting of exact supergraphs of the graph Q, denoted as $\mathcal{G}_{\{Q\}} = \{\mathrm{id}(G_i) | Q \subseteq G_i, G_i \in \mathcal{G}\}$.
- $\mathcal{G}_{\mathcal{F}_Q}$ – the subset of $\mathcal{G}$, consisting of candidate supergraphs of the graph Q, denoted as $\mathcal{G}_{\mathcal{F}_Q} = \cap \mathcal{G}_{\{F_j\}}$, where $F_j \in \mathcal{F}_Q$.
- T_Q – the trapdoor for the query graph Q.

III. Preliminaries

A. Graph Query

A labeled, undirected, and connected graph is a five-tuple as $\{V, E, \Sigma_V, \Sigma_E, L\}$, where V is the vertex set, $E \subseteq V \times V$ is the edge set, and L is a labeling function: $V \to \Sigma_V$ and $E \to \Sigma_E$. We use the number of vertices $|V(G)|$ to represent the size of the graph G.

Subgraph Isomorphism Given two graphs $G = \{V, E, \Sigma_V, \Sigma_E, L\}$ and $G' = \{V', E', \Sigma_V, \Sigma_E, L'\}$, G is subgraph isomorphic to G' if there is an injection $f : V \to V'$ such that

1. $\forall\ v \in V, L(v) = L'(f(v))$.
2. $\forall\ (u, v) \in E, (f(u), f(v)) \in E'$.
3. $\forall\ (u, v) \in E, L(u, v) = L'(f(u), f(v))$.

Graph Containment Query If G is subgraph isomorphic to G', we call G is a *subgraph* of G' or G' is a *supergraph* of G, denoted as $G \subseteq G'$. Such relation is also referred to as G is *contained by* G' or G' *contains* G. Given a graph dataset $\mathcal{G} = (G_1, G_2, \ldots, G_m)$ and a query graph Q, a graph containment query problem is to find all the supergraphs of Q from the dataset $\mathcal{G}$, denoted as $\mathcal{G}_{\{Q\}} = \{\mathrm{id}(G_i) | Q \subseteq G_i, G_i \in \mathcal{G}\}$ where $\mathrm{id}(G_i)$ is the identifier of the graph G_i. The number of supergraphs of Q, i.e., $|\mathcal{G}_{\{Q\}}|$, is called the support, or the frequency of Q.

Considering the large size of the graph dataset, it is impractical to solve the graph containment query problem by sequentially checking whether Q is subgraph isomorphic to each graph in $\mathcal{G}$ or not, because checking subgraph isomorphism has been proved to be NP-complete [18]. To reduce the times of checking subgraph isomorphism, most graph query works [3]–[8] follow the principle of "filtering-and-verification".

Filtering-and-Verification In the filtering phase, a feature-based index for the dataset $\mathcal{G}$ is utilized to prune most negative data graphs that does not contain the query graph Q, and then produce the candidate supergraph set. In the verification phase, the subgraph isomorphism is checked between the query graph and every candidate supergraph to output the exact supergraph set $\mathcal{G}_{\{Q\}}$.

The feature-based index is pre-built from the entire graph dataset, where each feature F_j is a substructure of a data graph in the dataset, such as subpath [3], subtree [5], [6], [8] and subsubgraph [4], [7]. Let $\mathcal{F} = (F_1, F_2, \ldots, F_n)$ represent the feature set. The supergraph set of every feature F_j, denoted as $\mathcal{G}_{\{F_j\}}$, is stored in the index. Let $\mathcal{F}_Q = \{F_j | F_j \subseteq Q, F_j \in \mathcal{F}\}$ denote the query feature set consisting of features contained in the query graph. Then, the candidate supergraphs of the query graph G can be obtained by the intersection operation as $\mathcal{G}_{\mathcal{F}_Q} = \cap \mathcal{G}_{\{F_k\}}$, where $F_k \in \mathcal{F}_Q$. The false positive ratio is then defined as $\frac{|\mathcal{G}_{\mathcal{F}_Q}|}{|\mathcal{G}_{\{Q\}}|}$

Frequent and Discriminative Substructure It is infeasible and unnecessary to index every possible substructure of all the graphs in a large dataset, and therefore only frequent and discriminative substructures are indexed to reduce the index size. A feature F_j is frequent if its support, or frequency is large enough, i.e., $|\mathcal{G}_{\{F_j\}}| \geq \sigma$, where σ is called the minimum support. A feature F_j is discriminative if it can provide more pruning power than its subgraph feature set, i.e., $\frac{|\cap_k \mathcal{G}_{\{F_k\}}|}{|\mathcal{G}_{\{F_j\}}|} \geq \gamma$, where $F_k \subseteq F_j$ and γ is called the discriminative threshold.

B. Secure Euclidean Distance Computation

In order to compute the inner product in a privacy-preserving method, we will adapt the secure Euclidean distance computation in the secure k-nearest neighbor (kNN) scheme [15]. In this scheme, the Euclidean distance between a database record p_i and a query vector q is used to select k nearest database records. The secret key is composed of one $(d+1)$-bit vector as S and two $(d+1) \times (d+1)$ invertible matrices as $\{M_1, M_2\}$, where d is the number of fields for each record p_i. First, every data vector p_i and the query vector q are extended to $(d+1)$-dimensional vectors as $\vec{p_i}$ and $\vec{q}$, where the $(d+1)$-th dimension is set to $-0.5||p_i^2||$ and 1, respectively.

Besides, the query vector $\vec{q}$ is scaled by a random number $r > 0$ as (rq, r). Then, $\vec{p_i}$ is split into two random vectors as $\{\vec{p_i}', \vec{p_i}''\}$, and $\vec{q}$ is also split into two random vectors as $\{\vec{q}\,', \vec{q}\,''\}$. Note here that vector S functions as a splitting indicator. Namely, if the j-th bit of S is 0, $\vec{p_i}'[j]$ and $\vec{p_i}''[j]$ are set as the same as $\vec{p_i}[j]$, while $\vec{q}\,'[j]$ and $\vec{q}\,''[j]$ are set to two random numbers so that their sum is equal to $\vec{q}[j]$; if the j-th bit of S is 1, the splitting process is similar except that $\vec{p_i}$ and $\vec{q}$ are switched. The split data vector pair $\{\vec{p_i}', \vec{p_i}''\}$ is encrypted as $\{M_1^T\vec{p_i}', M_2^T\vec{p_i}''\}$, and the split query vector pair $\{\vec{q}\,', \vec{q}\,''\}$ is encrypted as $\{M_1^{-1}\vec{q}\,', M_2^{-1}\vec{q}\,''\}$. In the query step, the product of the data vector pair and the query vector pair, i.e., $-0.5r(||p_i||^2 - 2p_i \cdot q)$, is serving as the indicator of the Euclidean distance $(||p_i||^2 - 2p_i \cdot q + ||q||^2)$ to select k nearest neighbors. Without prior knowledge of the secret key, neither the data vector nor the query vector, after such a series of processes, can be recovered by analyzing their corresponding ciphertexts. The security analysis in [15] shows that this computation technique is secure against known-plaintext attack, which is roughly equal in security to a d-bit symmetric key. Therefore, d should be no less than 80 to make the search space sufficiently large.

IV. PPGQ: The Framework and Privacy

In this section, we define the framework of query over encrypted graph-structured data in cloud computing and establish various strict system-wise privacy requirements for such a secure cloud data utilization system.

A. The Framework

Our proposed framework focuses on how the query works with the help of index which is outsourced to the cloud server. We do not illustrate how the data itself is encrypted, outsourced or accessed, as this is a complementary and orthogonal issue and has been studied elsewhere [16]. The framework of PPGQ is illustrated as follows.

- FSCon$(\mathcal{G}, \sigma)$ *Takes the graph dataset $\mathcal{G}$ and the minimum support σ as inputs, outputs a frequent feature set $\mathcal{F}$.*
- KeyGen(ξ) *Takes a secret ξ as input and outputs a symmetric key $\mathcal{K}$.*
- BuildIndex$(\mathcal{G}, \mathcal{K})$ *Takes the graph dataset $\mathcal{G}$ and the symmetric key $\mathcal{K}$ as inputs, output a searchable index $\mathcal{I}$.*
- TDGen$(Q, \mathcal{K})$ *Takes the query graph Q and the symmetric key $\mathcal{K}$ as inputs, outputs a corresponding trapdoor T_Q.*
- Query$(T_Q, \mathcal{I})$ *Takes the trapdoor T_Q and the searchable index $\mathcal{I}$ as inputs, returns $\mathcal{G}_{\mathcal{F}_Q}$, i.e., the candidate supergraphs of query graph Q.*

The first three algorithms, i.e., FSCon, BuildIndex, and BuildIndex, are run by the data owner as pre-processes. The query algorithm is run on the cloud server as a part of the cloud data storage service. According to various search control mechanisms, the trapdoor generation algorithm TDGen may be run by either the data owner or the data user. Besides, depending on some specific application scenarios, while search requests on confidential documents may be allowed for all users, the access to document contents may be forbidden for those low-priority data users. Note that neither search control or access control are within the scope of this paper.

B. Choosing Frequent Features

To build a feature-based index, there are three choices of features, i.e., subpath, subtree and subgraph, which can be extracted from the graph dataset. According to the feature comparison in [5], with the same minimum support, either subtree-based or subgraph-based feature set is larger than subpath-based one, especially when the feature size is between 5 and 20. To be consistent with the size of graph which is $|V(G)|$, the size of feature is measured by its number of vertices $|V(F_i)|$. As for the cloud server, the larger feature set will demand more index storage, and also incur larger computation cost during the query process. However, the pruning power of the subgraph-based index performs the best among all the three choices, which leads to the lowest false positive ratio and the smallest candidate supergraph set.From the perspective of the data user, the size of the candidate supergraph set has a direct and important impact on the communication and computation cost. Compared with the powerful cloud server, data users may access the cloud server through portable devices, e.g., mobile phones and netbooks, which have limited capability of communication and computation to retrieve the candidate supergraph set and check subgraph isomorphism. To this end, the subgraph-based index is more appropriate than the other two choices for our PPGQ framework that is designed for the efficient graph-structured data utilization in cloud computing. To generate the frequent feature set, there have been a lot of frequent subgraph mining algorithms over the large graph dataset, such as $gSpan$ [19], and $Gaston$ [20]. For the indexing purpose, every frequent subgraph should be represented as a unique canonical label which can be accomplished by existing graph sequentialization techniques, like CAM [21] and DFS [19]. Besides, the shrinking process on the frequent feature set is not adopted in our framework since it will weaken the pruning power of index. As the subgraph is chosen as the feature to build index in our framework, we do not distinguish between frequent feature and frequent subgraph in the rest of this paper.

C. Privacy Requirements

As described in the framework, *data privacy* is to prevent the cloud server from prying into outsourced data, and can be well protected by existing access control mechanism [16]. In related works on privacy-preserving query, like searchable encryption [10], representative privacy requirement is that the server should learn nothing but query results. With this general privacy statement, we explore and establish a set of stringent privacy requirements specifically for the PPGQ framework. While data privacy guarantees are demanded by default in the related literature, various *query privacy* requirements involved in the query procedure are more complex and difficult to tackle as follows.

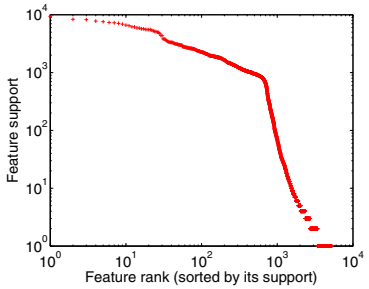


Fig. 2: Distribution of Feature Support

1) Index Privacy: With respect to the *index privacy*, if the cloud server deduces any association between frequent features and encrypted dataset from outsourced index, it may learn the major structure of a graph, or even the entire topology of a small graph. Therefore, searchable index should be constructed in such a way that prevents the cloud server from performing such kind of association attack.

2) Feature Privacy: Data users usually prefer to keep their query from being exposed to others like the cloud server, and the most important concern is to hide what they are querying, i.e., the features indicated by the corresponding trapdoor. Although trapdoor can be generated in a cryptographic way to protect the query features, the cloud server may do some statistical analysis over the search results to make an estimate. Especially, the *feature support* (i.e., the number of data graphs containing the feature), a kind of statistical information, is sufficient to identify the feature with high probability. When the cloud server knows some background information of the dataset, this feature-specific information can be utilized to reverse-engineer the feature. As presented in Fig. 2, the distribution of feature support in the AIDS antiviral screen dataset [22] provides enough information to identify most frequent features in the dataset. Such problem is similar with the keyword privacy issue in [23], where document frequency (the number of documents containing the keyword) is used as a statistical information to reverse-engineer the keyword.

3) Trapdoor Unlinkability: The trapdoor generation function should be a randomized one instead of being deterministic. In particular, the cloud server should not be able to deduce the relationship of any given trapdoors, e.g., to determine whether the two trapdoors are formed by the same search request or not. Otherwise, the deterministic trapdoor generation would give the cloud server advantage to accumulate frequencies of different search requests regarding different features, which may further violate the aforementioned feature privacy requirement. So the fundamental protection for trapdoor unlinkability is to introduce sufficient nondeterminacy into the trapdoor generation procedure.

4) Access Pattern: Access pattern is the sequence of query results where each query result is $\mathcal{G}_{\mathcal{F}_Q}$, including the id list of candidate supergraphs of the query graph. Then the access pattern is denoted as $(\mathcal{G}_{\mathcal{F}_{Q_1}}, \mathcal{G}_{\mathcal{F}_{Q_1}}, \ldots)$ which are the results of sequential queries. In related literature, although a few schemes (e.g., [24]) have been proposed to utilize private information retrieval (PIR) technique [25] to hide access pattern, our proposed schemes are not designed to protect access pattern for the efficiency concerns. This is because any PIR-based technique must "touch" the whole dataset outsourced on the server which is inefficient in the large scale cloud system. To this end, the query result of any single feature F_j, which is part of access pattern, cannot be hidden from the cloud server. Such query result $\mathcal{G}_{\{F_j\}}$ will directly expose the support of the feature, and break the feature privacy as discussed above. Therefore, we do not consider the single-feature query in our proposed schemes.

TABLE I: Analysis on inner products in two correlated queries

G	$\mathcal{F}_Q$	$\mathcal{F}_{Q'} = \mathcal{F}_Q \bigcup \{F_k\}$	${y_i}'/y_i$	$\lambda' - \lambda$
G_i	$y_i = r\lambda_i$	${\lambda_i}' = \lambda_i + 1,\ {y_i}' = r'{\lambda_i}'$	$\frac{\lambda_i+1}{\lambda_i} \cdot \frac{r'}{r}$	1
G_j	$y_j = r\lambda_j$	${\lambda_j}' = \lambda_j,\ {y_j}' = r'{\lambda_j}'$	$\frac{r'}{r}$	0

V. PPGQ: The Proposed Scheme and Analysis

In order to accomplish the filtering purpose in the graph query procedure, the data graph G_i is selected as a candidate supergraph of the query graph Q if and only if G_i contains all the frequent features in Q. Let λ_i represent the number of query features contained in the data graph G_i. For every candidate supergraph G_i, its corresponding λ_i should be equal to the size of the query feature set $\mathcal{F}_Q$, i.e., $\lambda_i = |\mathcal{F}_Q|$. To obtain the candidate supergraph set, we propose to employ the efficient inner product computation for pruning negative data graphs G_j that do not contain the query graph, i.e., $\lambda_j < \mathcal{F}_Q$. Specifically, every data graph G_i is formalized as a bit vector g_i where each bit $g_{i[j]}$ is determined by checking whether G_i contains the frequent feature F_j or not. If $F_j \subseteq G_i$, $g_{i[j]}$ is set as 1; otherwise, it is set as 0. The query graph Q is formalized as a bit vector q where each bit $q_{[j]}$ also represents the existence of the frequent feature F_j in the query feature set $\mathcal{F}_Q$. Then, λ_i can be acquired via computing the inner product of the data vector g_i and the query vector q, i.e., $g_i \cdot q$. To preserve the strict system-wise privacy, the data vector g_i and the query vector q should not be exposed to the cloud server. In this section, we first design a secure inner product computation mechanism, which is adapted from the secure Euclidean distance computation technique, and then show how to improve it to be privacy-preserving under the known-background threat model.

A. Secure Inner Product Computation

As the inner product of the data vector and the query vector is preferred to select candidate supergraphs of the query graph, the secure Euclidean distance computation technique in the secure kNN scheme [15] cannot be directly utilized here. By eliminating the extended dimension which is related to the Euclidean distance, the final inner product result changes to be $r(g_i \cdot q)$. Since the new result $r(g_i \cdot q)$ can serve as an indicator of the original inner product $g_i \cdot q$, it seems that an efficient and secure inner product computation scheme can be appropriately achieved. However, the cloud server may break the feature privacy via analyzing final inner products and figuring out some feature-specific statistical information, e.g.,

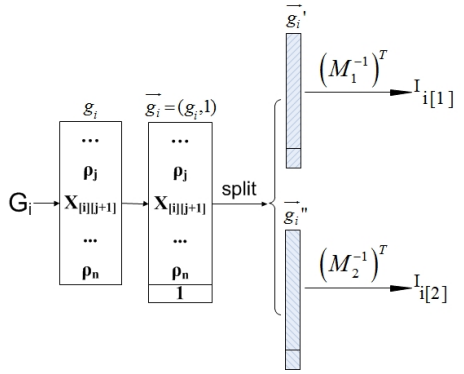


Fig. 3: Build subindex for each data graph

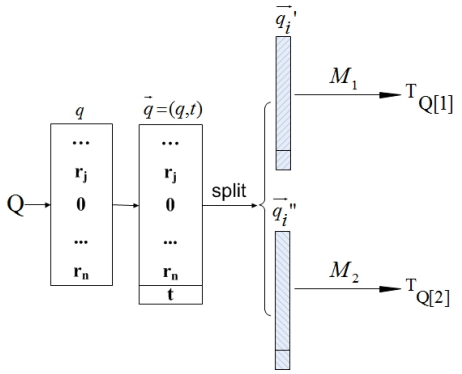


Fig. 4: Generate trapdoor for query graph

the support of feature. With the background knowledge of the outsourced graph dataset, which can be obtained by the cloud server under the known-background model, such feature-specific information could be further utilized to identify what feature is included in the query at high probability. We first demonstrate how such statistical analysis attack could break feature privacy as follows.

Whenever there exist two query graphs which have inclusion relationship, the cloud server could explore the relationship among final inner products in two queries. Assume that T_Q and $T_{Q'}$ be trapdoors for two query graphs Q and Q', and their corresponding query feature sets have the inclusion relation as $\mathcal{F}_Q \subset \mathcal{F}_{Q'}$. Especially, when the differential feature subset contains only one feature, i.e., $|\mathcal{F}_{Q''}| = 1$ where $\mathcal{F}_{Q''} = \mathcal{F}_{Q'} \backslash \mathcal{F}_Q$, the cloud server can deduce an estimate of the support of the differential feature and further identify this feature with the background knowledge of the graph dataset. As listed in Tab. I, the second query feature set $\mathcal{F}_{Q'}$ includes one more feature as F_k than the first one $\mathcal{F}_Q$. The cloud server evaluates the expression ${y_i}'/y_i$, which is equal to $({\lambda_i}'/\lambda_i)(r'/r)$ for every graph G_i, and then obtains a large number of different values. However, these values could be distinguished into two categories. If the graph G_i does not contain the feature F_k, i.e., ${\lambda_i}' = \lambda_i$, its corresponding expression evaluation ${y_i}'/y_i$ is equal to r'/r; otherwise, it is larger than r'/r and can be easily detected because of its special ratio as $\frac{\lambda_i+1}{\lambda_i}$. Therefore, the minimum values over the whole dataset indicate that corresponding data graphs do not contain the feature F_k, and other graphs with larger values contain it. In addition, by checking whether the expression y_i is equal to 0 or not, the special case where the data graph G_i contains neither feature in $\mathcal{F}_Q$ can be recognized by the cloud server. In such case, the existence of feature F_k in G_i can be determined by checking whether the expression ${y_i}'$ is equal to 0 or not. To this end, the total number of data graphs containing this feature, i.e., $|\mathcal{G}_{\{F_k\}}|$, is uncovered. Under the known-background threat model, the cloud server could break the feature privacy with both the support of single feature and the distribution of all the supports as illustrated in Fig. 2.

B. The Proposed Privacy-Preserving Graph Query Scheme

The statistical analysis attack shown above works when the final inner product y_i is a multiple of λ_i, i.e., the number of query features contained in the data graph G_i. To this end, we should break such scale relationship to make the previous statistical analysis attack infeasible. Our proposed design is to convert both the data vector and the query vector from the bit structure to more sophisticated structures. Specifically, if the frequent feature F_j is contained in the data graph G_i, the corresponding element $g_{i[j]}$ in the data vector g_i is set as $\rho_{[j]}$ instead of 1 where ρ is a n-dimensional vector; otherwise, $g_{i[j]}$ is set as $X_{[i][j]}$ where X is a $n \times n$ matrix and $X_{[i][j]}$ is a random number less than $\rho_{[j]}$. Correspondingly, if F_j is contained in the query graph Q, $q_{[j]}$ is set as a positive random number r_j instead of 1; otherwise, $q_{[j]}$ is set as 0. In addition, to hide the original inner product, we resume the dimension extending operation where the $g_{i[n+1]}$ is set as 1 and the $q_{[n+1]}$ is set as a random number t. As a result of these modifications, the final inner product of the data vector and the query vector, i.e., $g_i \cdot q + t$, should be equal to $\rho \cdot q + t$ for any candidate supergraph. Note that, the vector ρ is constant as a part of the secret key, but t and r_j in q are randomly generated for each query. Our proposed privacy-preserving graph query scheme is designed as follows with details in Fig. 5.

- $\mathsf{FSCon}(\mathcal{G}, \sigma)$ The data owner utilizes existing frequent subgraph mining algorithms to generate the frequent subgraph set $\mathcal{F}$, and then creates the frequent feature dictionary $\mathcal{D}$ and the feature-based inverted index I_{inv}.
- $\mathsf{KeyGen}(K_S, n)$ With the master key K_S, the data owner generates the secret key $\mathcal{K}$, consisting of the splitting indicator S, two invertible matrices $\{M_1, M_2\}$, and the vector ρ.
- $\mathsf{BuildIndex}(\mathcal{G}, \mathcal{F}, \mathcal{K})$ For each data graph G_i, this algorithm creates the subindex I_i as shown in Fig. 3. The data owner first creates a vector g_i with length n, in which the value of $g_{i[j]}$ is determined by whether graph G_i contains the corresponding feature F_j or not (steps 1 and 2). Subsequently, the data vector g_i is processed by applying the dimension extending where the $(n+1)$-th entry in $\vec{g_i}$ is set to 1 (step 3) and further adopting the splitting and encrypting procedures in the secure Euclidean Distance computation scheme (steps 4 and 5). Finally, a subindex

FSCon($\mathcal{G}, \sigma$)

1. Mine frequent feature set $\mathcal{F} = \{F_1, F_2, \ldots, F_n\}$ from graph dataset $\mathcal{G}$ with the minimum support threshold σ;
   i) For each frequent feature F_j, where $1 \leq j \leq n$, generate its supergraph id set $\mathcal{G}_{\{F_j\}}$;
2. Create the frequent feature dictionary $\mathcal{D} = \{\mathcal{L}_{F_1}, \mathcal{L}_{F_2}, \ldots, \mathcal{L}_{F_n}\}$, where $\mathcal{L}_{F_j}$ is the unique canonical label of F_j;
3. Build the feature-based inverted index $I_{inv} = \{\mathcal{G}_{\{F_1\}}, \mathcal{G}_{\{F_2\}}, \ldots, \mathcal{G}_{\{F_n\}}\}$.

KeyGen(K_S, n)

1. Create an $(n+1)$-bit vector S, two $(n+1) \times (n+1)$ invertible matrices M_1, M_2, and an n-dimensional vector ρ;
   i) $\{S, M_1, M_2, \rho\} \xleftarrow{\mathcal{R}} K_S$;
2. Output the secret key $\mathcal{K} = \{S, M_1, M_2, \rho\}$.

BuildIndex($I_{inv}, \mathcal{K}$)

1. Create a $n \times n$ matrix X, where $X_{[i][j]}$ is a random number less than $\mathcal{K}_{[4][j]}$;
2. For each graph G_i, where $1 \leq i \leq m$, create a n-dimensional data vector g_i;
   i) If $id(G_i) \in I_{inv[j]}$, set $g_{i[j]} = \mathcal{K}_{[4][j]}$; otherwise, set $g_{i[j]} = X_{[i][j]}$;
3. Extend every g_i to $(n+1)$-dimensional $\vec{g_i}$;
   i) For $1 \leq j \leq n$, set $\vec{g}_{i[j]} = g_{i[j]}$; Set $\vec{g}_{i[n+1]} = 1$;
4. According to the splitting indicator $\mathcal{K}_{[1]}$, split every $\vec{g_i}$ to two vectors $\vec{g_i}'$ and $\vec{g_i}''$;
   i) For $1 \leq j \leq n+1$, if $\mathcal{K}_{[1][j]} = 0$, set both $\vec{g_i}'_{[j]}$ and $\vec{g_i}''_{[j]}$ as $\vec{g}_{i[j]}$;
      otherwise, set $\vec{g_i}'_{[j]}$ and $\vec{g_i}''_{[j]}$ as two random numbers such that $\vec{g_i}'_{[j]} + \vec{g_i}''_{[j]} = \vec{g}_{i[j]}$;
5. Encrypt these two vectors by inverses of the two matrices, and combine them as the subindex I_i for G_i;
   i) $I_i = \{((\mathcal{K}_{[2]})^{-1})^T \vec{g_i}', ((\mathcal{K}_{[3]})^{-1})^T \vec{g_i}''\}$;
6. Output the encrypted index $I = \{I_1, I_2, \cdots, I_m\}$.

TDGen($Q, \mathcal{D}, \mathcal{K}$)

1. Initialize the query feature set: $\mathcal{F}_Q = \emptyset$;
2. For each frequent feature $\mathcal{D}_{[j]}$, $1 \leq j \leq n$:
   i) If $\mathcal{D}_{[j]} \subseteq Q$, $\mathcal{F}_Q = \mathcal{F}_Q \bigcup \{\mathcal{D}_{[j]}\}$;
3. Create a n-dimensional query vector q for the input query graph Q;
   i) Generate n positive random numbers as $r_1, r_2, \ldots, r_n$;
   ii) For $1 \leq j \leq n$, if $\mathcal{D}_{[j]} \in \mathcal{F}_Q$, set $q_{[j]} = r_j$; otherwise, set $q_{[j]} = 0$;
4. Extend q to $(n+1)$-dimensional $\vec{q}$, and generate a random number t;
   i) For $1 \leq j \leq n$, set $\vec{q}_{[j]} = q_{[j]}$; Set $\vec{q}_{[n+1]} = t$;
5. According to the splitting indicator $\mathcal{K}_{[1]}$, split $\vec{q}$ to two vectors as $\vec{q}'$ and $\vec{q}''$;
   i) For $1 \leq j \leq n+1$, if $\mathcal{K}_{[1][j]} = 1$, set both $\vec{q}'_{[j]}$ and $\vec{q}''_{[j]}$ as $\vec{q}_{[j]}$;
      otherwise, set $\vec{q}'_{[j]}$ and $\vec{q}''_{[j]}$ as two random numbers such that $\vec{q}'_{[j]} + \vec{q}''_{[j]} = \vec{q}_{[j]}$;
6. Encrypt these two vectors by the two invertible matrices as $\{(\mathcal{K}_{[2]})^T \vec{q}', (\mathcal{K}_{[3]})^T \vec{q}''\}$;
7. Output the trapdoor T_Q for query graph Q;
   i) $T_Q = \{(\mathcal{K}_{[2]})^T \vec{q}', (\mathcal{K}_{[3]})^T \vec{q}'', \sum \mathcal{K}_{[4][j]} q_{[j]} + t\}$;

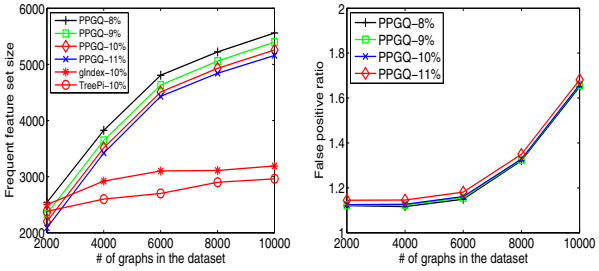
Query($\mathcal{I}, T_Q$)

1. Initialize the query result: $\mathcal{G}_{\mathcal{F}_Q} = \emptyset$;
2. For each subindex $I_{[i]}$, $1 \leq i \leq m$:
   i) Compute inner product as $I_{i[1]} \cdot T_{Q[1]} + I_{i[2]} \cdot T_{Q[2]}$;
   ii) If the inner product is equal to $T_{Q[3]}$, set $\mathcal{G}_{\mathcal{F}_Q} = \mathcal{G}_{\mathcal{F}_Q} \bigcup \{\mathrm{id}(G_i)\}$;
3. Output the query result $\mathcal{G}_{\mathcal{F}_Q}$.

Fig. 5: Privacy-Preserving Graph Query Scheme

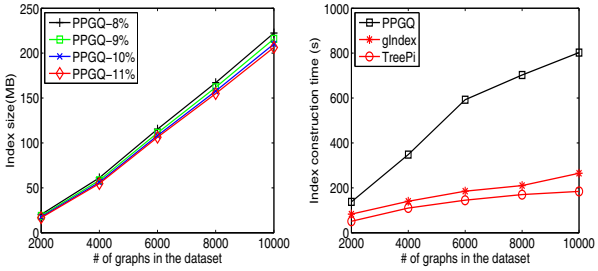
$I_i = \{(M_1^{-1})^T \vec{g_i}', (M_2^{-1})^T \vec{g_i}''\}$ is created for every data graph G_i and associated with the encrypted data graph $\widetilde{G_i}$ for outsourcing to the cloud server .

- TDGen(Q) With the query graph Q as input from the data user, this algorithm outputs the trapdoor T_Q as shown in Fig. 4. The query feature set $\mathcal{F}_Q$ is first generated through checking which features in $\mathcal{F}$ are also contained in Q (steps 1 and 2). An n-dimensional vector q is created by assigning a positive random number r_j to the element $q_{[j]}$ if $F_j \in \mathcal{F}_Q$; otherwise, $q_{[j]} = 0$ (step 3). This initial query vector q is then extended to an $(n+1)$-dimensional vector as $\vec{q} = (q, t)$, where t is a non-zero random number (step 4). After adopting the splitting and encrypting processes in the secure Euclidean distance computation technique (steps 5 and 6), the trapdoor T_Q for the query graph Q is generated as $\{M_1 \vec{q}', M_2 \vec{q}'', \sum \rho_{[j]} q_{[j]} + t\}$, where the third element is the expected final inner product of the query vector and the data vector for every candidate supergraph.
- Query($\mathcal{I}, T_Q$) With the trapdoor T_Q, the cloud server computes the inner product of $\{T_{Q[1]}, T_{Q[2]}\}$ with every subindex I_i for data graph G_i, and returns graph id list $\mathcal{G}_{\mathcal{F}_Q}$ where each graph has an inner product as exactly same as $T_{Q[3]}$. The data user can further do the graph



(a) (b)

Fig. 6: Relation between minimum support and (a) Frequent feature set size. (b) False positive ratio.



(a) (b)

Fig. 7: Index construction cost. (a) Storage cost of index. (b) Time of index construction.

verification to remove false positives from $\mathcal{G}_{\mathcal{F}_Q}$, and finally get the exact result as $\mathcal{G}_{\{Q\}}$.

C. The Analysis

Analysis of this scheme follows three aspects of design goals described in section II-C.

1) Effectiveness: Assume Q consists of ℓ query features, i.e., $\ell = |\mathcal{F}_Q|$. For any supergraph G_i of the query graph Q, it includes all the ℓ features in $\mathcal{F}_Q$ which is extracted from Q. Therefore, all the ℓ corresponding elements in the data vector are equal to those in the ρ, respectively, i.e., $g_{i[j_k]} = \rho_{[j_k]}$, where $1 \leq k \leq \ell$. Besides, each corresponding element in the query vector as $q_{i[j_k]}$ is set as r_{j_k}, and all other elements is set as 0. The final inner product $g_i \cdot q + t$ for any supergraph G_i is then equal to $\sum \rho_{[j_k]} r_{j_k} + t$, which is also the result of $\rho \cdot q + t$. The later one $\rho \cdot q + t$ has been included in the trapdoor and serves as an indicator to select candidate supergraphs. It means that our scheme does not introduce any false negative into the result $\mathcal{G}_{\mathcal{F}_Q}$, as every exact supergraph in $\mathcal{G}_{\{Q\}}$ will produce the same inner product as $\rho \cdot q + t$ with the query vector. But false positive supergraphs may be introduced into $\mathcal{G}_{\mathcal{F}_Q}$ by those data graphs that do not contain the query graph Q but contain all the features in $\mathcal{F}_Q$.

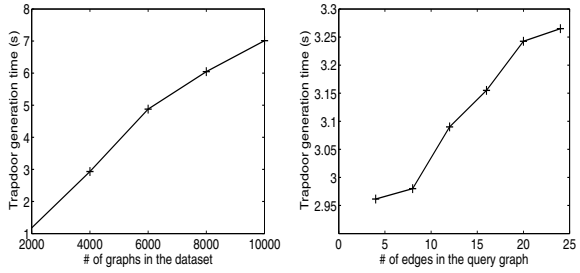
2) Efficiency: As far as the data user is concerned, the query response is well presented because the final inner product for every data graph can be efficiently computed by the cloud server via two multiplications of $(n+1)$-dimensional vectors. Although some costly computations are involved in FSCon and BuildIndex, such as graph sequentialization, they are unavoidable for building a graph index. And more importantly, they are executed for only one time during the whole scheme. Apart from these computations, the encryption of the data vector or the query vector only needs two multiplications of a $(n+1) \times (n+1)$ matrix and a $(n+1)$-dimensional vector in BuildIndex or TDGen, respectively. Besides, to avoid the high computation cost of inverting two high-dimension matrices in TDGen, every query vector is encrypted by the two matrices M_1 and M_2 themselves, instead of their inverses of M_1 and M_2 utilized in the secure Euclidean distance computation. Correspondingly, the costly inverting operation is transferred to the one-time index construction procedure.

3) Privacy: With the randomness introduced by the splitting process and the random numbers r_j and t, our scheme can generate two totally different trapdoors for the same query graph Q. This nondeterministic property of the trapdoor generation can guarantee the *trapdoor unlinkability*. Recall that the data vector encryption with matrices has been proved to be secure against known-plaintext attack in [15], the *index privacy* is protected unless the secret key $\mathcal{K}$ is disclosed.

As mentioned above, in the secure inner product computation technique, the primary reason why the statistical analysis attack works is that the final inner product y_i has the scale relationship with λ_i. And this scale relationship exists just because y_i is a multiple of the original inner product $g_i \cdot q$ which is equal to λ_i. Our proposed scheme introduces randomness in both g_i and q to break the equivalence relationship between $g_i \cdot q$ and λ_i. As a consequence, the value of $g_i \cdot q$ does not completely depend on λ_i. In the case where data graph G_i contain fewer query features than data graph G_j, it is still possible that $g_i \cdot q \geq g_j \cdot q$. Moreover, the extended dimension t is utilized to break the direct scale relationship between y_i and $g_i \cdot q$, which further eliminates the indirect scale relationship between y_i and λ_i. So the cloud server cannot deduce the special ratio as $\frac{\lambda_i+1}{\lambda_i}$ which is used to detect the inclusion relationship between two query feature sets as discussed in the previous section V-A. Without disclosing such inclusion relationship, the cloud server cannot compute the support of a single feature. In other words, the statistical analysis cannot break the *feature privacy*, and all the expected privacy requirements in section IV-C are being met by the proposed scheme.

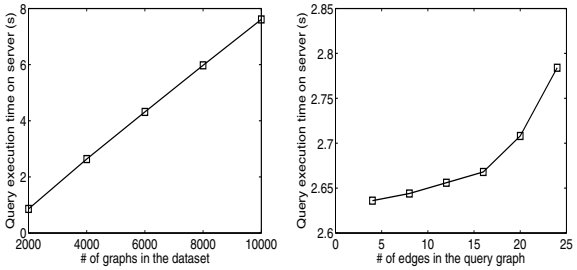
VI. Experimental Evaluations

In this section, we demonstrate a thorough experimental evaluation of the proposed scheme on the AIDS antiviral screen dataset [22] that is widely used in graph query related works [4]–[8]. It contains $42,390$ compounds with totally 62 distinct vertex labels. The 5 datasets in our experiment, as same as in [4], are $\mathcal{G}_{2000}$, $\mathcal{G}_{4000}$, $\mathcal{G}_{6000}$, $\mathcal{G}_{8000}$, and $\mathcal{G}_{10000}$, where $\mathcal{G}_N$ contains N graphs randomly chosen from the AIDS dataset. We also adopt the same 6 sets of query graphs Q_4, Q_8, Q_{12}, Q_{16}, Q_{20} and Q_{24}, where Q_i contains i query graphs with i edges. Default dataset and query graphs are set as $\mathcal{G}_{4000}$ and Q_4 in our experiment, respectively. $gSpan$ [19] is used as



(a) (b)

Fig. 8: Trapdoor generation time. (a) in different dataset with same query size as 4. (b) for different query size with same dataset size as 4000.



(a) (b)

Fig. 9: Query execution time on server. (a) in different dataset with same query size as 4. (b) for different query size with same dataset size as 4000.

the frequent subgraph mining algorithm in our scheme. The maximum size of frequent subgraph $maxL$ is set to 11, and the minimum support σ for feature F_j is defined as follows, $\sigma = 1$ if $|V(F_j)| < 5$; otherwise, $\sigma = \sqrt{\frac{|V(F_j)|}{maxL}} \cdot minsup \cdot |\mathcal{G}|$, where the default $minsup$ is set to 10% and $|\mathcal{G}|$ is the size of dataset. Graph boosting toolbox [26] is utilized to implement $gSpan$ algorithm and check subgraph isomorphism, and the public utility routines by Numerical Recipes are employed to compute the inverse of matrix. The query performance in our scheme is evaluated on the Amazon Elastic Compute Cloud (EC2) in which we deploy the basic 64-bit Linux Amazon Machine Image (AMI) with 4 CPU cores (2×1.2GHz); the performance of other procedures in our scheme, such as index construction and trapdoor generation related to data owners or data users, is evaluated on a 2.8GHz CPU with Redhat Linux. The compared schemes are gIndex [4] and TreePi [6], and their performance data are provided in [6] which are also run on a 2.8GHz CPU with RedHat Linux.

A. False Positive and Index Construction

The minimum support determines the threshold for a subgraph of being an indexed feature. Specifically, the large value of minimum support means that only very frequent subgraphs in the dataset could be treated as valid in the filtering procedure. However, such high requirement will reduce the number of features included in the index whose pruning power would be directly affected. With the decreasing number of indexed features, the query graph can only be represented by less number of query features, and therefore more and more data graphs, which does not contain the query graph Q but contain all the graphs in the smaller size query feature set $\mathcal{F}_Q$, are included in the candidate supergraph set $\mathcal{G}_{\mathcal{F}_Q}$. As demonstrated in Fig. 6 where four different minimum supports through adjusting $minsup$ from 8% to 11% are examined, the false positive ratio defined as $\frac{|\mathcal{G}_{\mathcal{F}_Q}|}{|\mathcal{G}_{\{Q\}}|}$ raises in accordance with the $minsup$. Although the minimum support should be set as small as possible to prune as many data graphs as possible, the larger one will introduce more storage cost of index due to the larger size of frequent feature set as shown in Fig. 7(a). Moreover, as shown in Fig. 6(a), the size of the frequent feature set increases in a lower speed when the dataset is larger than 600, while the minimum support $\sigma = \sqrt{\frac{|V(F_j)|}{maxL}} \cdot minsup \cdot |\mathcal{G}|$ increases linearly with the size of dataset. As a result, there will be increasing false positives in the candidate supergraph set, which is validated in Fig. 6(b).

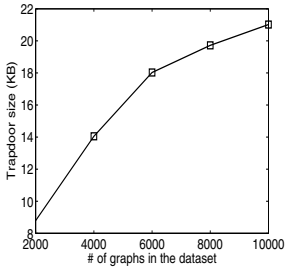


Fig. 10: Trapdoor size in different dataset

As shown in Fig. 6(a), our frequent feature set is larger than that in the other two related works since our scheme does not adopt the shrinking process on the frequent set by choosing discriminative subgraphs. Besides, the false positive ratio in our scheme is almost same as that in gSpan and a little larger than the scheme Tree+$\triangle$ [6], through the performance data provided in [6]. As shown in Fig. 7(b), because our index construction involves the encryption process on data vectors, the time cost here is about four times larger than that in other schemes which only deal with plaintext index. Note that this construction is only a one-time procedure in the whole scheme.

B. Trapdoor Generation and Query

Like index construction, every trapdoor generation incurs two multiplications of a matrix and a split query vector, whose dimensionality becomes larger with the increasing number of documents in dataset. As demonstrated in Fig. 8(a), the time to generate a trapdoor is linear with the number of data graphs in the dataset. Fig. 8(b) demonstrates the trapdoor generation cost is almost linear with the size of query graph, which is defined as the number of edges in the query graph. Such linearity is caused by the fact that the major costly operation mapping query graph to vector is roughly determined by query size since all the query features should be mapped.

In the query process in our scheme design, the cloud server

executes the filtering process by computing the inner product of trapdoor and each encrypted data vector. Fig. 9 shows that the query time is almost linear with the number of data graphs in the dataset. Although the query time in our scheme is much larger than that in gSpan, whose query time is around 100 milliseconds presented in [7], our scheme is performing query on the encrypted index. With respect to the communication cost in the query procedure, the size of trapdoor is the same as that of subindex for single data graph. As shown in Fig. 10, the size of trapdoor keeps constant in the same dataset, no matter how many features are contained in a query graph.

VII. RELATED WORK

Graph Containment Query To reduce the computation cost caused by checking subgraph isomorphism, most research on plaintext graph containment query problem follows the "filtering-and-verification" framework [3]–[8] to decrease the size of candidate supergraph set. Feature-based index has been increasingly explored by choosing different substructures as features. Shasha et al. [3] designed a path-based index approach. However, paths carry few structural information and therefore have limited filtering power. Yan et al. [4] proposed gIndex to build index from frequent and discriminative subgraphs which can carry more structure characteristics. Zhang et al. [6] utilized frequent and discriminative subtrees instead of subgraphs to build the index.

Keyword-based Searchable Encryption Traditional single keyword searchable encryption schemes [10]–[14] usually build an encrypted searchable index such that its content is hidden to the server unless it is given appropriate trapdoors generated via secret key(s) [2]. To enrich search semantics, conjunctive keyword search [27] over encrypted data have been proposed. These schemes incur large overhead caused by their fundamental primitives, such as computation cost by bilinear map [27].As a more general search approach, predicate encryption schemes [28] are recently proposed to support both conjunctive and disjunctive search. However, none of existing boolean keyword searchable encryption schemes support graph semantics as we propose to explore in this paper.

VIII. CONCLUSION

In this paper, for the first time, we define and solve the problem of query over encrypted graph-structured cloud data, and establish a variety of privacy requirements. For the efficiency consideration, we adopt the principle of "filtering-and-verification" to prune as many negative data graphs as possible before verification, where a feature-based index is pre-built to provide feature-related information for every encrypted data graph. Then, we choose the inner product as the pruning tool to carry out the filtering procedure efficiently. To meet the challenge of supporting graph semantics, we propose a secure inner product computation, and then improve it to achieve various privacy requirements under the known-background threat model. Thorough analysis investigating privacy and efficiency of our scheme is given, and the evaluation further shows our scheme introduces low overhead on computation and communication. As our future work, we will explore privacy-preserving schemes under stronger threat models.

ACKNOWLEDGMENT

This work was supported in part by the US National Science Foundation under grants CNS-0716306, CNS-0831628, CNS-0746977, and CNS-0831963.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," University of California, Berkeley, Tech. Rep. UCB-EECS-2009-28, Feb 2009.

[2] S. Kamara and K. Lauter, "Cryptographic cloud storage," in *RLCPS, January 2010, LNCS. Springer, Heidelberg*.

[3] D. Shasha, J.-L. Wang, and R. Giugno, "Algorithmics and applications of tree and graph searching," in *Proc. of PODS*, 2002.

[4] X. Yan, P. S. Yu, and J. Han, "Graph indexing: a frequent structurebased approach," in *Proc. of SIGMOD*, 2004.

[5] P. Zhao, J. X. Yu, and P. S. Yu, "Graph indexing: tree + delta >= graph," in *Proc. of VLDB*, 2007.

[6] S. Zhang, M. Hu, and J. Yang, "Treepi: A novel graph indexing method," in *Proc. of ICDE*, 2007.

[7] J. Cheng, Y. Ke, W. Ng, and A. Lu, "Fg-index: towards verification-free query processing on graph databases," in *Proc. of SIGMOD*, 2007.

[8] H. Shang, Y. Zhang, X. Lin, and J. X. Yu, "Taming verification hardness: an efficient algorithm for testing subgraph isomorphism," in *Proc. of VLDB*, 2008.

[9] S. Berreti, A. Bimbo, and E. Vicario, "Efficient matching and indexing of graph models in content-based retrieval," *IEEE Trans. Pattern Analysis and Machine Intelligence*, 2001.

[10] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proc. of ACM CCS*, 2006.

[11] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. of EUROCRYPT*, 2004.

[12] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," in *Proc. of ICDCS*, 2010.

[13] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *Proc. of IEEE INFOCOM'10 Mini-Conference*, San Diego, CA, USA, March 2010.

[14] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," in *Proc. of INFOCOM*, 2011.

[15] W. K. Wong, D. W. Cheung, B. Kao, and N. Mamoulis, "Secure knn computation on encrypted databases," in *Proc. of SIGMOD*, 2009.

[16] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *Proc. of INFOCOM*, 2010.

[17] S. Zerr, D. Olmedilla, W. Nejdl, and W. Siberski, "Zerber+r: Top-k retrieval from a confidential index," in *Proc. of EDBT*, 2009.

[18] M. R. Garey and D. S. Johnson, "Computers and intractability: A guide to the theory of np-completeness," Freeman, New York, NY, USA, 1990.

[19] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *Proc. of ICDM*, 2002.

[20] S. Nijssen and J. N. Kok, "A quickstart in frequent structure mining can make a difference," in *Proc. of SIGKDD*, 2004.

[21] J. Huan, W. Wang, and J. Prins, "Efficient mining of frequent subgraphs in the presence of isomorphism," in *Proc. of ICDM*, 2003.

[22] "Aids antiviral screen dataset," 1999, http://dtp.nci.nih.gov/docs/aids/aids_data.html.

[23] S. Zerr, E. Demidova, D. Olmedilla, W. Nejdl, M. Winslett, and S. Mitra, "Zerber: r-confidential indexing for distributed documents," in *Proc. of EDBT*, 2008, pp. 287–298.

[24] R. Brinkman, "Searching in encrypted data," in *University of Twente, PhD thesis*, 2007.

[25] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, "Cryptography from anonymity," in *Proc. of FOCS*, 2006, pp. 239–248.

[26] H. Saigo, S. Nowozin, T. Kadowaki, T. Kudo, and K. Tsuda, "Gboost: A mathematical programming approach to graph classification and regression," in *Machine Learning*, 2008.

[27] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proc. of TCC*, 2007, pp. 535–554.

[28] A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters, "Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption," in *Proc. of EUROCRYPT*, 2010.