Structure-Preserving Subgraph Query Services

Zhe Fan, Byron Choi, Qian Chen, Jianliang Xu, Haibo Hu, Sourav S Bhowmick

Abstract—A fundamental problem of graph databases is *subgraph isomorphism query* (a.k.a subgraph query): given a query graph Q and a graph database, it retrieves the graphs Gs from the database that contain Q. Due to the cost of managing massive data coupled with the computational hardness of subgraph isomorphism testing, outsourcing the computations to a third-party provider is an appealing alternative. However, confidentiality has been a critical attribute of *Quality of Service* (*QoS*) in query services. To the best of our knowledge, *subgraph query services with tunable preservation of privacy of structural information* have never been addressed. In this paper, we present the first work on *structure-preserving* subIso (SPsubIso). A crucial step of our work is to transform subIso — the seminal subgraph isomorphism algorithm (the Ullmann's algorithm) — into a series of matrix operations. We propose a novel *cyclic group based encryption* (CGBE) method for private matrix operations. We propose a *protocol* that involves the query client and *static indexes* to optimize SPsubIso. We prove that the structural information of both Q and G are preserved under CGBE and analyze the privacy preservation in the presence of the optimizations. Our extensive experiments on both real and synthetic datasets verify that SPsubIso is efficient and the optimizations are effective.

1 Introduction

Graphs are powerful tools for a wide range of real applications, from biological and chemical databases, social networks, citation networks to information networks. Large graph data repositories have been consistently found in recent applications. For example, PubChem [27] is a real database of chemical molecules, which can be freely accessed via its web interface, for its clients to query chemical compounds. Another example, namely Daylight [8], delivers chem-informatics technologies to life science companies and recently, it has provided web services to allow clients to access its technologies via a network. Further applications of graphs can be found in literature such as [2], [5], [33].

Subgraph query (via subgraph isomorphism), which is a fundamental and powerful query in various real graph applications, has actively been investigated for performance enhancements [5], [7], [14], [16], [29], [31], [33], [36], [39] recently. However, due to the high complexity of subgraph query, hosting efficient subgraph query services has been a technically challenging task, because the *owners* of graph data may not always possess the IT expertise to offer such services and hence may *outsource* to *query service providers* ($\mathcal{SP}$). $\mathcal{SP}$s are often equipped with high performance computing utilities (*e.g.*, a cloud) that offer better scalability, elasticity and IT management [13]. Unfortunately, as $\mathcal{SP}$s may not always be trusted, security (such as the confidentiality of messages exchanged) has been recognized as one of the critical attributes of *Quality of Services* (*QoS*) [25]. This directly influences the willingness of both data owners and query clients to use $\mathcal{SP}$'s services. In the past decade, there is a bloom on the research

- *Zhe Fan, Byron Choi, Qian Chen, Jianliang Xu and Haibo Hu are with the Department of Computer Science, Hong Kong Baptist University, China. E-mail: zfan, bchoi, qchen, xujl, haibo@comp.hkbu.edu.hk*
- *Sourav.S. Bhowmick is with School of Computer Engineering, Nanyang Technological University, Singapore. E-mail: assourav@ntu.edu.sg*

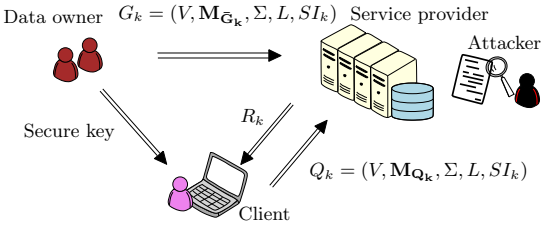


Fig. 1. Overview of the system model.

on query processing with privacy preservation[1], for example, in the context of relational databases [18], spatial databases [19] and graph databases [2]. However, up to date, private subgraph query has not yet been studied.

Motivating example: Consider a pharmaceutical company with revenue that depends mostly on the invention of health care products. The company may have discovered new compounds for a new product. To save laboratory work, it may query the compounds from proprietary biological pathway networks to check whether it is possible for the ingredient compounds to form other compounds via certain chemical reactions (a structural pattern from the network). However, on the one hand, the company is reluctant to expose the queries (the ingredients) to the $\mathcal{SP}$, as it may apply for patents for the synthesis. On the other hand, the owner of the pathway networks may not only lack the expertise to host query services but may also be reluctant to release the networks to the public. The owner is willing to release it to paid users only. Hence, it is crucial to protect *both* the queries and the network from the $\mathcal{SP}$. Such privacy concerns also arise from social networks and biological networks, among many other applications.

In this paper, we investigate that the query client may prefer not to expose the structure of query graphs to the $\mathcal{SP}$, and meanwhile, the data owner may not want the $\mathcal{SP}$ to be able to infer the structure of their graph data. The fundamental problem being studied is to *evaluate subgraph query at the*

1. In addition to privacy protection via legal means, this stream of research has aimed to offer technological solutions for such protection.

$\mathcal{SP}$ with a preservation of the structures of both the query graphs and graph data in the paradigm of the query services. This paper, in particular, aims to protect the adjacency matrices of the data graph and queries from the $\mathcal{SP}$. To our knowledge, such a problem has never been addressed before.

In our recent work [10], we have addressed the authenticity of the answers of subgraph query, but not their confidentiality. A host of related work is also on privacy-preserving graph query [1], [2], [11], [17], [20], [22], [26], [34], [35]. However, none of these studies can support subgraph query with the structure preservation of the query and graph data. Another category of related research is on the study of privacy-preserving graph publication [3], [4], [24], [37], [38]. As the published data are modified in a non-trivial manner (*e.g.*, by sanitization), it is not clear how subgraph query can be supported.

The intrinsic difficulty of this research is that the $\mathcal{SP}$ cannot optimize query processing by directly using the structures of the graph, since such information cannot be exposed. However, most of the existing subgraph isomorphism algorithms (*e.g.*, VF2 [7], QuickSI [29] and Turbo$_{\text{iso}}$ [14]) for the query services must *traverse* the graph, which by definition leaks structural information. A naïve method is to transfer the entire database to the client for query processing. However, it is inefficient when the database is large.

Our techniques for a *structure-preserving* subIso (denoted as SPsubIso) are derived from the Ullmann's algorithm [30], a seminal algorithm for subgraph isomorphism. We revise the Ullmann's algorithm into *three steps* that form the foundation of our techniques. (1) Enum enumerates all *possible subgraph isomorphism mappings* $\mathbf{M}_i$s from query graph Q to data graph G; (2) Match verifies if the mapping $\mathbf{M}_i$ is *valid* or not; and (3) Refine reduces the search space of $\mathbf{M}_i$s by degree and neighborhood constraints. The benefits of adopting the Ullmann's algorithm are twofold: (1) the query evaluation between Q and G is mostly a series of matrix operations between their adjacency matrices $\mathbf{M_Q}$ and $\mathbf{M_G}$. It does not require traversals on structures; and (2) its query evaluation requires simple structures. This makes the privacy analysis simpler.

Specifically, to facilitate structure-preserving computations, we first transform subIso into a series of mathematical computations, denoted as TsubIso. TsubIso comprises three steps, corresponding to subIso: (1) TEnum enumerates all $\mathbf{M}_i$s; (2) TMatch verifies the validity of $\mathbf{M}_i$ by *additions and multiplications* using $\mathbf{M_Q}$ and $\mathbf{M}_{\bar{\mathbf{G}}}$, where $\mathbf{M}_{\bar{\mathbf{G}}}$ is the complement of $\mathbf{M_G}$; and (3) TRefine reduces the search space of $\mathbf{M}_i$s by *inner product*s on our proposed *static indexes* $\mathsf{SI_Q}$ and $\mathsf{SI_G}$ of Q and G, where $\mathsf{SI_Q}$ ($\mathsf{SI_G}$) is an ensemble of h-*hop information* of each vertex of Q ($\mathsf{SI_G}$) represented by a *bit vector*.

The major benefit of these three steps of TsubIso is that *only* mathematical operations are involved, which allows an adoption of private computations in encrypted domains. Based on TsubIso, we present our novel *structure-preserving* subIso (SPsubIso). In particular, we first propose a new *private-key encryption scheme*, namely *cyclic group based encryption scheme* (CGBE), to encrypt $\mathbf{M_Q}$ and $\mathbf{M}_{\bar{\mathbf{G}}}$ as $\mathbf{M}_{\mathbf{Q}k}$ and $\mathbf{M}_{\bar{\mathbf{G}}k}$. Then, we propose SPMatch involving the additions and multiplications under CGBE to check the validity of each mapping $\mathbf{M}_i$, with negligible false positives. Further, the computation results under CGBE can be *aggregated* to reduce communication overheads between the client and the $\mathcal{SP}$. We prove that CGBE is perfectly secure under *chosen plaintext attack* and the $\mathcal{SP}$ cannot learn any structures from SPMatch.

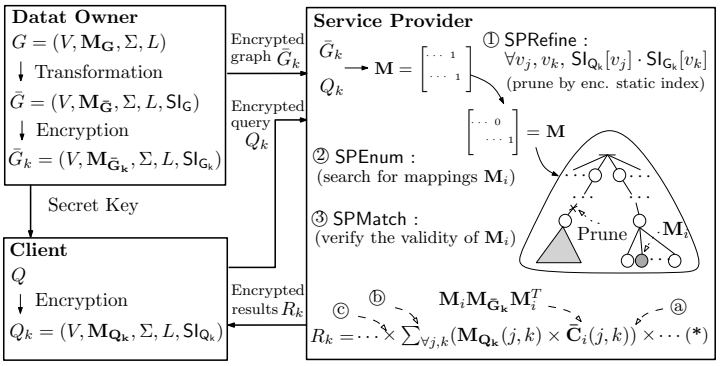


Fig. 2. Overview of our techniques.

Next, we propose SPEnum which optimizes the mapping enumeration by introducing a *protocol* that involves the client's participation, who informs the $\mathcal{SP}$ useless enumerations. In addition, to optimize SPsubIso, we develop SPRefine which exploits private inner products on the static indexes to derive a refinement that reduces the number of possible mappings. The indexes of the graphs are computed and encrypted offline, whereas those of the queries are computed once by the clients online. We analyze the effects of these optimizations on the probabilities that the $\mathcal{SP}$ may correctly determine graph structures. Therefore, the clients may tune the trade-off between performances and privacy requirements.

To summarize, the contributions of this paper are as follows:

- We transform the Ullmann's algorithm subIso as TsubIso. It only involves a few mathematical computations, such that its private version can be proposed and analyzed;
- We propose a *structure-preserving* subIso (SPsubIso) based on TsubIso, consisting of SPMatch, SPEnum and SPRefine. Specifically, we propose CGBE for SPMatch, which supports efficient encryption and decryption, *partial* additions and multiplications, and aggregation of computation results. We propose a protocol for SPEnum that involves the client to eliminate useless mappings. We propose SPRefine that exploits private inner products of static indexes to further optimization;
- We analyze the privacies of SPMatch, SPEnum and SPRefine; and
- We conduct detailed experiments to verify that SPsubIso is efficient and our optimizations are effective.

The remaining of this paper is organized as follows. We first give the problem definition in Sec. 2. We then study the preliminary of subgraph isomorphism subIso in Sec. 3. Sec. 4 presents the transformed algorithm TsubIso. We propose the SPsubIso in Sec. 5. We give the privacy analysis in Sec. 6. We present the experimental results in Sec. 7. Sec. 8 discusses related work, and we finally conclude this paper in Sec. 9.

2 Problem Formulation

This section presents a formulation of the problem studied in this paper. More specifically, we present the system model, privacy target, attack model, and problem statement.

System model. We follow the system model that has been well received in the literature of database outsourcing (shown in Fig. 1), and known to be suitable for many applications. It consists of three parties:

(1) *Data owner*: The owner owns and encrypts the graph data G. He/she then outsources the encrypted graph to the service provider and delivers the secret keys to clients for encryption of the query graphs and decryption of the encrypted result;
(2) *Service provider* ($\mathcal{SP}$): The $\mathcal{SP}$ may be equipped with powerful computing utilities such as a cloud. The $\mathcal{SP}$ evaluates a client's query over the encrypted data, on behalf of the data owner, and returns the encrypted result to the client; and
(3) *Client*: A client encrypts the query graph Q using the secret keys, submits it to the $\mathcal{SP}$, and decrypts the returned encrypted result to obtain the final answer.

Attack model. We assume the dominating semi-honest adversary model [1], [2], [19], [21] from literature, where the attackers are *honest-but-curious* and the $\mathcal{SP}$ may also be the attacker. For presentation simplicity, we often *term the attackers as the* $\mathcal{SP}$. We assume that the attackers are the *eavesdroppers* and adopt the *chosen plaintext attack* [21]. We assume that the $\mathcal{SP}$ and clients are not allowed to collude.

Privacy target. To facilitate a technical discussion, we assume that the privacy target is to protect the *structures* of a query graph Q and a graph data G from the $\mathcal{SP}$ under the attack model defined above. The *structural information* of Q and G considered is the adjacency matrices of Q and G, respectively. More specifically, the probability that the $\mathcal{SP}$ correctly determines the values of the adjacency matrix of the graph is guaranteed to be lower than a threshold with reference to that of *random guess*.

The *problem statement* of this paper can be stated as follows: *Given the above system and attack model, we seek an efficient approach to facilitate the subgraph isomorphism query services with preserving the above defined privacy target.*

3 Preliminaries

In this section, we first discuss the background for the subgraph query and revise the classical Ullmann's algorithm.

3.1 Subgraph Query

This paper assumes a graph database is a large collection of graphs of modest sizes. We consider *undirected labeled connected graphs*. A graph is denoted as $G = (V, E, \Sigma, L)$, where $V(G)$, $E(G)$, $\Sigma(G)$ and L are the set of vertices, edges, vertex labels and the function that maps a vertex to its label, respectively. We use $\mathsf{Deg}(v_i, G)$ to denote the degree of the vertex v_i in graph G. In this paper, we focus on the graph with only vertex labels. Our proposed techniques can be extended to support the graph with edge labels with minor modifications.

Definition 3.1: Given two graphs $G = (V, E, \Sigma, L)$ and $G' = (V', E', \Sigma', L')$, a *subgraph isomorphism mapping* from G to G' is an injective function $f : V(G) \rightarrow V(G')$ such that

- $\forall u \in V(G), f(u) \in V(G'), L(u) = L'(f(u))$; and
- $\forall (u, v) \in E(G)$, $(f(u), f(v)) \in E(G')$. □

We say a graph G is a subgraph of another graph G' if and only if there exists a subgraph isomorphism mapping (in short mapping) from G to G', denoted as $G \subseteq G'$ or $\mathsf{subIso}(G, G') = \mathtt{true}$. It is known that deciding whether G is the subgraph of G' is NP-hard. *Subgraph isomorphism query* or simply *subgraph query* can be described as follows.

Definition 3.2: Given a query graph Q and a graph database, the *subgraph query* is to retrieve the graphs from the database where Q is a subgraph of the graphs. □

3.2 Revised Ullmann's Algorithm

Subgraph query has been a classical query and many algorithms, *e.g.*, [7], [14], [29], [30], have been proposed in the literature. As motivated in Sec. 1, the Ullmann's algorithm [30] is simple for privacy preservation. In this subsection, we revise the Ullmann's algorithm into three interleaving steps, namely enumeration, matching and refinement. These form a foundation of our discussions, as we propose our structure preservation techniques for them.

Prior to the algorithmic details, we present some notations used in this paper. We use subIso to refer to as the Ullmann's algorithm. We denote a query as $Q = (V, \mathbf{M_Q}, \Sigma, L)$ and graph as $G = (V, \mathbf{M_G}, \Sigma, L)$, $m = |V(Q)|$ and $n = |V(G)|$, $\mathbf{M_Q}$ and $\mathbf{M_G}$ are the adjacency matrices of Q and G, respectively. $\mathbf{M_Q}(j, k)$ is a *binary* value, where $\mathbf{M_Q}(j, k) = 1$ if $(v_j, v_k) \in E(Q)$, and otherwise 0. The values of the entries of $\mathbf{M_G}$ are defined, similarly. Both adjacency matrices $\mathbf{M_Q}$ and $\mathbf{M_G}$ carry the most fundamental *structural information*, *i.e.*, the edge information. We use a $m \times n$ binary matrix $\mathbf{M}$ to represent the *vertex label mapping* between Q and G. Specifically, $\forall j, k$, $\mathbf{M}(j, k) = 1$ if $L(v_j) = L(v_k)$, where $v_j \in V(Q)$ and $v_k \in V(G)$; and otherwise 0.

The revised Ullmann's algorithm (subIso) is detailed in Algo. 1. subIso takes Q and G as input and returns true if Q is the subgraph of G. Initially, it determines the vertex label mapping $\mathbf{M}$ (Lines 1-2). Then, subIso checks from $\mathbf{M}$ if there is a subgraph isomorphism mapping from Q to G by using three steps: (1) Enum; (2) Match; and (3) Refine. Next, we highlight some details of each step.

Enumeration (Lines 8-17). Enum *enumerates* all *possible subgraph isomorphism mappings* from Q to G by $\mathbf{M}$. Each possible mapping is denoted as $\mathbf{M}_i$. *Each column of* $\mathbf{M}_i$ *contains at most one 1* and *each row of* $\mathbf{M}_i$ *has only one 1* (Lines 12-13). $\mathbf{M}_i$ is enumerated from $\mathbf{M}$ row by row (Line 14). When an $\mathbf{M}_i$ is obtained (Line 8), Match checks if $\mathbf{M}_i$ is a subgraph isomorphism mapping (Line 9). It is easy to see that the number of possible $\mathbf{M}_i$s enumerated is $O(n^m)$.

Matching (Lines 18-21). For each $\mathbf{M}_i$ enumerated from $\mathbf{M}$, if there exists a matrix $\mathbf{C}_i$, $\mathbf{C}_i = \mathbf{M}_i\mathbf{M_G}\mathbf{M}_i^T$, such that $\exists j, k$,

$$\mathbf{M_Q}(j, k) = 1 \ \wedge \ \mathbf{C}_i(j, k) = 0 \quad (1)$$

then such an $\mathbf{M}_i$ cannot be an subgraph isomorphism mapping from Q to G. Note that $\mathbf{C}_i$ *intuitively* represents the adjacency

Algorithm 1 Revised Ullmann's algorithm subIso (Q, G)

Input: The query graph Q and the data graph G.
Output: **True** if Q is a subgraph of G, **False** otherwise.

1: Initialize $\mathbf{M}_i := \mathbf{0}$
2: Generate $\mathbf{M}$ from (V, Σ, L) of Q and G
3: **if** !Refine($\mathbf{M}$, Q, G) /* Refinement */
4: **return False**
5: **if** !Enum(0, $\mathbf{M}_i$, $\mathbf{M}$, Q, G) /* Enumeration */
6: **return False**
7: **return True**

Procedure 1.1 Enum (d, $\mathbf{M}_i$, $\mathbf{M}$, Q, G)
8: **if** $d = m$
9: **return** Match($\mathbf{M}_i$, Q, G) /* Matching */
10: **if** !Refine($\mathbf{M}$, Q, G) /* Refinement */
11: **return False**
12: **for each** c, where $c < n$, $\mathbf{M}(d, c) = 1$, and $\forall d' < d$ $\mathbf{M}_i(d', c) = 0$
13: $\mathbf{M}_i(d, c) := 1$
14: **if** Enum($d + 1$, $\mathbf{M}_i$, $\mathbf{M}$, Q, G)
15: **return True**
16: $\mathbf{M}_i(d, c) := 0$
17: **return False**

Procedure 1.2 Match($\mathbf{M}_i, Q, G$)
18: $\mathbf{C}_i = \mathbf{M}_i\mathbf{M_G}\mathbf{M}_i^T$
/* violation */
19: **if** $\exists j, k$, $\mathbf{M_Q}(j, k) = 1 \wedge \mathbf{C}_i(j, k) = 0$
20: **return False**
21: **return True**

Procedure 1.3 Refine($\mathbf{M}, Q, G$)
22: **do** $\forall j, k$, $\mathbf{M}(j, k) = 1$
23: **if** *degree constraint* or *neighborhood constraint* fails
24: $\mathbf{M}(j, k) := 0$
25: **while** $\mathbf{M}$ is not changed
26: **if** $\exists j$, *s.t.*, $\forall k, \mathbf{M}(j, k) = 0$
27: **return False**
28: **return True**

matrix of a subgraph of G, that Q may be isomorphic to through $\mathbf{M}_i$. Formula 1 states that there is an edge between vertices j and k in Q but no corresponding edge in the subgraph of G, represented by $\mathbf{C}_i$. Such an $\mathbf{M}_i$ is definitely *not* a mapping. We term the case in Formula 1 as a *violation of subgraph isomorphism* (or simply *violation*). $\mathbf{M}_i$ without violation is called a *valid* mapping. That is, Q is a subgraph of G through $\mathbf{M}_i$.

Refinement (Lines 22-28). The number of 1's in $\mathbf{M}$ significantly increases the number of $\mathbf{M}_i$ to be enumerated in worst case. In the Ullmann's algorithm, there are two optimizations, called *refinements*, to reduce the number of 1's in $\mathbf{M}$. Intuitively, the first refinement exploits the *degree constraint*, whereas the second refinement relies on the *neighborhood constraint*: $\forall j, k$, $\mathbf{M}(j, k) = 1 \Rightarrow$

(1) $\mathsf{Deg}(v_j, Q) \leq \mathsf{Deg}(v_k, G)$; and
(2) $\forall x, \mathbf{M_Q}(j, x) = 1 \Rightarrow \exists y, \mathbf{M}(x, y)\mathbf{M_G}(k, y) = 1$.

Refinement is performed when (1) $\mathbf{M}$ is determined (Line 3) and (2) $\mathbf{M}_i$s are enumerated (Line 10). For any pair of j and k, $\mathbf{M}(j, k) = 1$, if either one of the constraints is not satisfied, the algorithm then flips $\mathbf{M}(j, k)$, *i.e.*, sets $\mathbf{M}(j, k) = 0$ (Lines 22-24). If any row of $\mathbf{M}$ contains only 0s, it reports there is no valid mapping (Lines 26-27).

Example 3.1: Fig. 3 shows an example for Algo. 1. The LHS shows the query graph Q and the data graph G and their adjacency matrices (below the graphs). The RHS shows the enumeration of $\mathbf{M}_i$s. $\mathbf{C}_1$ is computed by $\mathbf{M}_1$, which is a valid mapping from Q to G. Suppose we do not perform Refine, $\mathbf{M}_2$ will be enumerated. Match determines that $\mathbf{M}_2$ contains violations, as shown. However, when Refine is performed, $\mathbf{M}(1, 4)$ is flipped to 0 as v_4 of G does not connect to v_2 and $\mathsf{Deg}(v_1, Q) > \mathsf{Deg}(v_4, G)$. $\mathbf{M}_2$ is not enumerated at all.

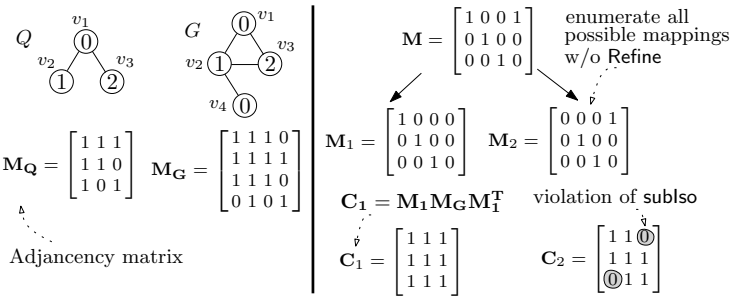


Fig. 3. Adjacency matrices of Q and G; Two possible mappings ($\mathbf{M}_1$ and $\mathbf{M}_2$) and a violation in $\mathbf{C}_2$ (Formula 1).

4 SUBISO WITH MATRIX OPERATIONS

From subIso in Algo. 1, it can be noted that the violation defined by Formula 1 in Match (Line 19) is determined by processing of the entries between $\mathbf{M_Q}$ and $\mathbf{C}_i$, and the neighborhood constraint (Line 23) precisely exploits edge information. Hence, as motivated in Sec. 1, we cast subIso into an algorithm that uses a series of mathematical computations, denoted as TsubIso. This enables us to derive private versions of such operations in later sections.

Foremost, we extend the definition of the query and data graph (Q and G), defined in Def. 4.1. Def. 4.1 only differs from the one presented in Sec. 3 that the entries in the adjacency matrix $\mathbf{M_G}$ are *flipped*, *i.e.*, 0s (resp. 1s) are set to 1s (resp. 0s), for the transformed subIso (to be detailed soon). Moreover, Q and G are extended with precomputed indexes, called static indexes (to be detailed in SubSec. 5.3), to enhance performances. Since our subsequent discussions always assume the extended queries/graphs, we omit the term "extended" for brevity.

Definition 4.1: The *extended data graph* of G is denoted as $\bar{G} = (V, \mathbf{M}_{\bar{\mathbf{G}}}, \Sigma, L, \mathsf{SI_G})$ and the *query graph* is extended as $Q = (V, \mathbf{M_Q}, \Sigma, L, \mathsf{SI_Q})$, where $\mathbf{M}_{\bar{\mathbf{G}}}$ are *flipped*, *i.e.*, $\forall j, k$,

$$\mathbf{M}_{\bar{\mathbf{G}}}(j, k) = \neg\mathbf{M_G}(j, k),$$

and $\mathsf{SI_G}$ and $\mathsf{SI_Q}$ (called *static indexes*) are sets of bit vectors, for optimization purposes. □

Based on Def. 4.1, we rewrite subIso into transformed subIso called TsubIso in Algo. 2. The inputs are the query graph Q and data graph $\bar{G}$. It returns 0 if Q is a subgraph of G, and non zero otherwise. The corresponding three main steps of Algo. 1 in Algo. 2 are highlighted below.

Transformed enumeration. The main difference in TEnum is that Refine (Lines 10-11 of Algo. 1) is removed. The reason is that Refine exploits structural information, which is required to keep private. Another difference is that TEnum is invoked with an input message R that *aggregates* the subgraph isomorphism information from Q to G during the enumeration of $\mathbf{M}_i$s.

Transformed matching. In Match, the violation of Formula 1 (Line 19 of Algo 1) is checked by a condition defined on each entry of $\mathbf{M_Q}$ and $\mathbf{C}_i$, which leaks structural information. In comparison, with Def. 4.1, the presence of a violation is detected from the product of the matrices $\mathbf{M_Q}$ and $\bar{\mathbf{C}}_i$ (Lines 14-15) in TMatch. Further, the violation due to $\mathbf{M}_i$ is preserved under aggregations, *i.e.*, the result of $\mathbf{M}_i$ (denoted as R_i) is aggregated into one message R (Lines 16-17).

Algorithm 2 TsubIso (Q, $\bar{G}$)

Input: The query graph Q and the transformed data graph G.
Output: $R = 0$ if Q is a subgraph of G, $R = 1$ otherwise.

1: Initialize $R := 1$, $\mathbf{M}_i := \mathbf{0}$
2: Generate $\mathbf{M}$ from (V, Σ, L) of Q and $\bar{G}$
3: **if** !TRefine($M, Q, \bar{G}$) /* TRefinement */
4: **return** R
5: TEnum(0, $\mathbf{M}_i$, $\mathbf{M}$, Q, $\bar{G}$, R) /* TEnumeration */
6: **return** R

Procedure 2.1 TEnum($d, \mathbf{M}_i, \mathbf{M}, Q, \bar{G}, R$)
7: **if** $d = m$
8: TMatch($\mathbf{M}_i$, Q, $\bar{G}$, R) /* TMatching */
9: **for each** c, where $c < n$, $\mathbf{M}(d, c) = 1$ and $\forall d' < d$ $\mathbf{M}_i(d', c) = 0$
10: $\mathbf{M}_i(d, c) := 1$
11: TEnum($d + 1$, $\mathbf{M}_i$, $\mathbf{M}$, Q, $\bar{G}$, R)
12: $\mathbf{M}_i(d, c) := 0$

Procedure 2.2 TMatch($\mathbf{M}_i, Q, \bar{G}, R$)
13: Initialize $R_i := 0$, $\mathbf{MC}_i := \mathbf{0}$
14: $\bar{\mathbf{C}}_i := \mathbf{M}_i \mathbf{M}_{\bar{\mathbf{G}}} \mathbf{M}_i^T$
15: $\forall j, k, \mathbf{MC}_i(j, k) := \mathbf{M_Q}(j, k) \times \bar{\mathbf{C}}_i(j, k)$ /* Multiplication */
16: $R_i := \sum_{\forall j,k} \mathbf{MC}_i(j, k)$ /* Addition */
17: $R \times= R_i$ /* Multiplication */

Procedure 2.3 TRefine($\mathbf{M}, Q, G$)
18: **for each** j, k, $\mathbf{M}(j, k) = 1$
19: if $\mathsf{SI_Q}[v_j] \cdot \mathsf{SI_Q}[v_j] \neq \mathsf{SI_Q}[v_j] \cdot \mathsf{SI_G}[v_k]$
20: $\mathbf{M}(j, k) = 0$
21: **if** $\exists j$, *s.t.*, $\forall k, \mathbf{M}(j, k) = 0$
22: **return False**
23: **return True**

The detection of a violation in TMatch is illustrated with Fig. 4. Similar to Match, TMatch computes the "subgraph" $\bar{\mathbf{C}}_i$ that Q may be isomorphic to. With the data graph, $\bar{\mathbf{C}}_i$ is computed in Line 14. There are four possible cases of the entries of $\mathbf{M_Q}$ and $\bar{\mathbf{C}}_i$ and Fig. 4 a) highlights the case of the violation of Formula 1. That is, $\exists j, k$, $\mathbf{M_Q}(j, k) = 1$ and $\mathbf{C}_i(j, k) = 0$ (thus, $\bar{\mathbf{C}}_i(j, k) = 1$), then

$$\mathbf{M_Q}(j, k)\bar{\mathbf{C}}_i(j, k) = 1, \tag{2}$$

For the other three cases, the product is 0. Therefore, by Formula 2, TMatch detects the violation and aggregates the results as follows:

1. *Multiplication* (Line 15). For each pair of (j, k), TMatch computes $\mathbf{MC}_i(j, k) = \mathbf{M_Q}(j, k) \times \bar{\mathbf{C}}_i(j, k)$;
2. *Addition* (Line 16). TMatch sums up the entries of the product $\mathbf{MC}_i$, *i.e.*, $R_i = \sum_{\forall j,k} \mathbf{MC}_i(j, k)$. Note that R_i intuitively represents the validity of the mapping $\mathbf{M}_i$, *i.e.*, if $\mathbf{M}_i$ is valid, no violation is found and the value of R_i is 0, by Formula 2; and
3. *Multiplication* (Line 17). TMatch then aggregates R_i into R by a multiplication, *i.e.*, $R = R \times R_i$. If there is at least a valid $\mathbf{M}_i$, the value of R equals 0, and non zero otherwise.

It is worth highlighting that if there exists a subgraph isomorphism mapping $\mathbf{M}_i$ from Q to G, then $\mathbf{M}_i$ contains no violation, $R_i = 0$ and $R = 0$. Thus, $R = 0$ implies that Q is a subgraph of G. Otherwise, R is non zero, which implies all R_is are not zero and there must be some 1's in the entries of $\mathbf{MC}_i$, for all i. By Formula 2, there is a violation in each $\mathbf{M}_i$ and thus, Q is not a subgraph of G.

Example 4.1: We illustrate TMatch with the example shown in Figs. 4 b) and c). The query and graph are those shown in Fig. 3. Fig. 4 b) presents $\mathbf{M_Q}$ and $\mathbf{M}_{\bar{\mathbf{G}}}$. Fig. 4 c) reports the intermediate results of TMatch of two possible mappings $\mathbf{M}_1$ and $\mathbf{M}_2$ (Fig. 3). $\mathbf{M}_1$ is a valid mapping as R_1 computed

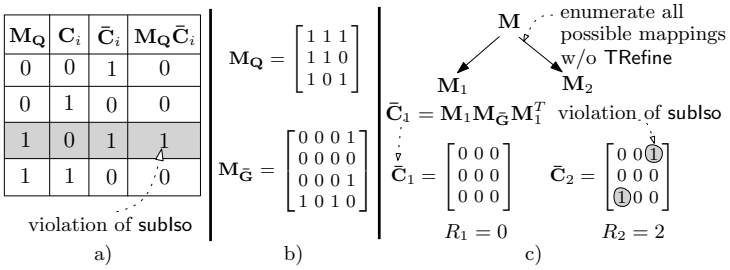


Fig. 4. (a) The truth table of $\mathbf{M_Q}\bar{\mathbf{C}}_i$; (b) Illustration of $\mathbf{M_Q}$ and $\mathbf{M}_{\bar{\mathbf{G}}}$; and (c) TMatch.

using $\mathbf{M_Q}$ and $\bar{\mathbf{C}}_1$ (in Lines 15-16) is 0. In comparison, R_2 computed using $\mathbf{M_Q}$ and $\bar{\mathbf{C}}_2$ is 2. Hence, $\mathbf{M}_2$ is an invalid mapping. $R = R_1 \times R_2 = 0$ indicates that there is a valid mapping and thus Q is a subgraph of G.

Transformed refinement. As the neighborhood constraint of Refine precisely exploits the edge information, it cannot be directly adopted. We transform Refine as TRefine that inner products (Line 19) between our proposed *static index* (SI, in the form of bit vector) are used for refinements. The index is called static as the indexes of the data graphs are precomputed and those of query graphs are computed by the client prior to TsubIso.

It is worth noting that TsubIso is mainly a series of mathematical operations, *i.e.*, additions, multiplications and inner products. This enables us to establish a connection to private query processing.

5 Structure-Preserving subIso

In this section, we propose structure-preserving subIso, denoted as SPsubIso. SPsubIso contains three steps: (1) structure-preserving Match (SPMatch) in SubSec. 5.1; (2) structure-preserving Enum (SPEnum) in SubSec. 5.2; and (3) structure-preserving Refine (SPRefine) in SubSec. 5.3.

Before presenting the details, we first give the definition of the *encrypted* query graph Q_k and the transformed graph $\bar{G}_k$, which are shared by SPMatch, SPEnum and SPRefine.

Definition 5.1: The *encrypted* Q and $\bar{G}$ are denoted as Q_k and $\bar{G}_k$, respectively, where $Q_k = (V, \mathbf{M_Q}_k, \Sigma, L, \mathsf{SI_{Q_k}})$ and $\bar{G}_k = (V, \mathbf{M}_{\bar{\mathbf{G}}k}, \Sigma, L, \mathsf{SI_{G_k}})$. $\mathbf{M_Q}_k$ ($\mathbf{M}_{\bar{\mathbf{G}}k}$) and $\mathsf{SI_{Q_k}}$ ($\mathsf{SI_Q}$) are the encrypted $\mathbf{M_Q}$ ($\mathbf{M}_{\bar{\mathbf{G}}}$) and $\mathsf{SI_Q}$ ($\mathsf{SI_G}$), respectively. □

It is worth remarking that we only protect $\mathbf{M_Q}$ (resp. $\mathbf{M}_{\bar{\mathbf{G}}}$) and $\mathsf{SI_Q}$ (resp. $\mathsf{SI_G}$) in Q (resp. $\bar{G}$), by using encryption, since (V, Σ, L) does not expose the structural information.

5.1 Structure-Preserving Matching

In this subsection, we adopt *cyclic group* and propose a novel *private-key encryption scheme* to encrypt $\mathbf{M_Q}$ and $\mathbf{M}_{\bar{\mathbf{G}}}$. We then propose SPMatch to compute the operations of TMatch in encrypted domain, where the mapping ($\mathbf{M}_i$) has been enumerated by SPEnum (to be discussed in SubSec. 5.2).

5.1.1 Cyclic Group Based Encryption

Recall that TMatch involves both additions and multiplications. Hence, the state-of-the-art *partially homomorphic encryption schemes* (*e.g.*, Paillier and ElGamal) [21] cannot be adopted to our problem. On the other hand, due to the

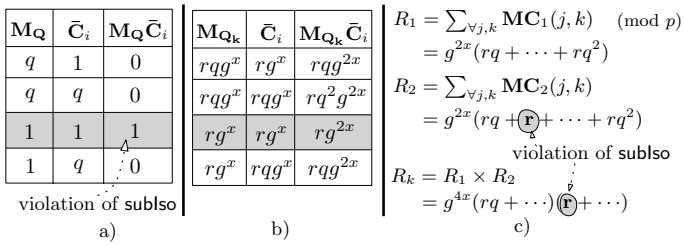


Fig. 5. (a) The encoding of the truth table shown in Fig. 4(a); (b) Encryption by CGBE; and (c) Illustration of SPMatch with $\mathbf{M}_1$ and $\mathbf{M}_2$.

known performance concerns of *fully homomorphic encryption scheme* (FHE) [12], we may not directly adopt FHE either.

Therefore, we propose a private-key encryption scheme, namely *cyclic graph based encryption scheme* (CGBE). CGBE not only supports both partial additions and multiplications, but also allows efficient encryption and decryption. Importantly, it is secure against CPA. However, the trade-off of using CGBE in SPMatch is that (1) it introduces *negligible false positives*; and (2) it requires multiple encrypted messages for aggregating a query result, which are sent to the client.

Before the detailed discussion, we first present the preliminary about cyclic group [21]. Let $\mathbb{G}$ be a group. $p = |\mathbb{G}|$ is denoted as the *order* of $\mathbb{G}$. In particular, $\forall g \in \mathbb{G}$, the order of $\mathbb{G}$ is the smallest positive integer p such that $g^p = 1$. Let $\langle g \rangle = \{g^i : i \in \mathbb{Z}_p, g^i \in \mathbb{Z}_n\} = \{g^0, g^1, \cdots, g^{p-1}\}$ denote the set of group elements generated by g. The group $\mathbb{G}$ is called *cyclic* if there exists an element $g \in \mathbb{G}$ such that $\langle g \rangle = \mathbb{G}$. In this case, the *order* of $\mathbb{G}$ is $p = |\mathbb{G}|$ and g is called a *generator* of $\mathbb{G}$. Next we propose the cyclic group based encryption scheme as follows.

Definition 5.2: The *cyclic group based encryption* scheme is a *private-key* encryption scheme, denoted as CGBE = (Gen, Enc, Dec), where

- Gen is a *key generation function*, which generates a secrete key $x \in [0, p-1]$ uniformly at random, a cyclic group $\langle g \rangle = \{g^i : i \in \mathbb{Z}_p, g^i \in \mathbb{Z}_n\}$. It outputs the private keys as $(x,\ g)$ and the value p which is known to the public.
- Enc is an *encryption function*, which takes as input a message m and the secrete key $(x,\ g)$. It chooses a random value r, and outputs the ciphertext
$$c = mrg^x \pmod{p}$$
- Dec is a *decryption function*, which takes as input a ciphertext c, and the secrete key $(x,\ g)$. It outputs
$$mr = cg^{-x} \pmod{p}$$

□

Note that the Dec function of CGBE only decrypts the ciphertext c as *the product of the message m and random value r*. This is because SPMatch does not require the exact value of m.

5.1.2 Encryption of $\mathbf{M_Q}$ and $\mathbf{M}_{\bar{\mathbf{G}}}$

To encrypt $\mathbf{M_Q}$ and $\mathbf{M}_{\bar{\mathbf{G}}}$, we first present an *encoding* for each entry of $\mathbf{M_Q}$ and $\mathbf{M}_{\bar{\mathbf{G}}}$.

Definition 5.3: The *encoding* of the entries of $\mathbf{M_Q}$ and $\mathbf{M}_{\bar{\mathbf{G}}}$ are: $\forall j, k$,

if $\mathbf{M_Q}(j,k) = 0$, set $\mathbf{M_Q}(j,k)$ as q; and
if $\mathbf{M}_{\bar{\mathbf{G}}}(j,k) = 0$, set $\mathbf{M}_{\bar{\mathbf{G}}}(j,k)$ as q,

where q is a large prime number. □

Algorithm 3 SPMatch ($\mathbf{M}_i$, Q_k, $\bar{G}_k$, R_k)

1: $\bar{\mathbf{C}}_i := \mathbf{M}_i \mathbf{M}_{\bar{\mathbf{G}}k} \mathbf{M}_i^T$
/* Multiplication */
2: $\forall j,k, \mathbf{MC}_i(j,k) := \mathbf{M_Q}_k(j,k) \times \bar{\mathbf{C}}_i(j,k) \pmod{p}$
/* Addition */
3: $R_i := \sum_{\forall j,k} \mathbf{MC}_i(j,k) \pmod{p}$
4: **if** $i \neq 0$, $i \bmod \omega \neq 0$
/* Multiplication */
5: $R_k \times= R_i \pmod{p}$
6: **else** Send R_k to client, $R_k := R_i$

In relation to Def. 5.3, we have the following Formula 3 that similar to Formula 2 to detect the violation. We note that *only* in case of $\mathbf{M_Q}(j,k) = 1$ and $\bar{\mathbf{C}}_i(j,k) = 1$,

$$\mathbf{M_Q}(j,k) \times \bar{\mathbf{C}}_i(j,k) = 1 \pmod{q}, \tag{3}$$

where $\bar{\mathbf{C}}_i = \mathbf{M}_i \mathbf{M}_{\bar{\mathbf{G}}} \mathbf{M}_i^T$, the product will be 0 otherwise. Fig. 5 a) shows the encoding of four possible combinations between entries, we can see that only if $\mathbf{M_Q}(j,k) = 1$ and $\bar{\mathbf{C}}_i(j,k) = 1$, the product becomes 1. Otherwise it is 0.

Under the encryption scheme CGBE in Def. 5.2 and the encoding in Def. 5.3, we are ready to define the encryption of the encoding of $\mathbf{M_Q}$ and $\mathbf{M}_{\bar{\mathbf{G}}}$ (in short, the encryption of $\mathbf{M_Q}$ and $\mathbf{M}_{\bar{\mathbf{G}}}$) as follows.

Definition 5.4: The *encryption* of $\mathbf{M_Q}$ and $\mathbf{M}_{\bar{\mathbf{G}}}$ are denoted as $\mathbf{M_Q}_k$ and $\mathbf{M}_{\bar{\mathbf{G}}k}$, respectively, where $\forall j, k$,

$$\begin{aligned} \mathbf{M_Q}_k(j,k) &= \mathsf{Enc}(\mathbf{M_Q}(j,k), x, g) \\ \mathbf{M}_{\bar{\mathbf{G}}k}(j,k) &= \mathsf{Enc}(\mathbf{M}_{\bar{\mathbf{G}}}(j,k), x, g) \end{aligned} \tag{4}$$

□

Example 5.1: We use Fig. 5 b) to illustrate an example of the encryption of $\mathbf{M_Q}$ by CGBE. $\forall j, k$, if $\mathbf{M_Q}(j,k) = 1$, $\mathbf{M_Q}_k(j,k) = \mathsf{Enc}(1, x, g) = rg^x \pmod{p}$; and if $\mathbf{M_Q}(j,k) = q$, $\mathbf{M_Q}_k(j,k) = \mathsf{Enc}(q, x, g) = qrg^x \pmod{p}$.

Finally, we remark that the large prime number q for the encoding (Def. 5.3) must be kept secret. Since CGBE is a symmetric encryption scheme, both the $\mathcal{DO}$ and the client hold the same keys (x, g, p), whereas $\mathcal{SP}$ keeps p only.

5.1.3 SPMatching

Based on Def. 5.4, we propose a *cyclic group based* matching (in short, SPMatch) derived from TMatch (in Algo. 2), shown in Algo. 3. In particular, the input value R_k is the encrypted message that aggregates the violation. SPMatch first generates $\bar{\mathbf{C}}_i$ (Line 1), which is computed from $\mathbf{M}_i$ and $\mathbf{M}_{\bar{\mathbf{G}}k}$. Then the following three steps are invoked.

1. *Multiplication* (Line 2). For each pair of (j,k), SPMatch computes $\mathbf{MC}_i(j,k) = \mathbf{M_Q}_k(j,k) \times \bar{\mathbf{C}}_i(j,k) \pmod{p}$;
2. *Addition* (Line 3). SPMatch sums up the entries in the product, *i.e.*, $R_i := \sum_{\forall j,k} \mathbf{MC}_i(j,k) \pmod{p}$. If $\mathbf{M}_i$ is valid, *i.e.*, no violation is found, the decryption of the sum is exactly 0, by Formula 3; and
3. *Multiplication* (Lines 4-6). SPMatch then aggregates R_i into R_k by multiplication (Line 5). If there is at least one valid mapping from Q to G, the decryption of a R_k equals 0. Otherwise, the decryption value is non zero. We remark that CGBE leads to *errors* if the number of R_is in R_k is larger than a predetermined value ω. We thereby propose

a *decomposition scheme* (discussed later) that sends to the client a sequence of R_ks, where each R_k aggregates ω R_i (Line 4).

Example 5.2: Fig. 5 b) shows an example to illustrate the multiplication of the four possible cases of combinations between $\mathbf{M}_{\mathbf{Q}_k}$ and $\bar{\mathbf{C}}_i$. We observe that only under the violation (shown in grey shadow), the product of $\mathbf{M}_{\mathbf{Q}_k}$ and $\bar{\mathbf{C}}_i$ does *not* contain q. Fig. 5 c) illustrates an example of SPMatch following Fig. 4 c). R_1 and R_2 are computed by the summations of $\mathbf{MC}_1$ and $\mathbf{MC}_2$, respectively. Note that R_2 contains violation as M_2 is not a valid mapping. R_k is produced.

Decryption at the client. After receiving all the encrypted results R_k, the client performs the decryption, which mainly contains two steps as belows.

1. For each message R_k aggregated with ω R_is, the client computes *the message encoded in* R_k as $R'_k = \mathsf{Dec}(R_k, x, g)^{2\omega}$; and
2. For each encoded message R'_k, the client computes the final result by $R = R'_k \bmod q$.

If any of R equals to 0, there is at least one valid isomorphic mapping $\mathbf{M}_i$ that contributes a 0 (Line 3) to the product R_k (Lines 4-5). Thus $\mathsf{subIso}(Q, G) = \texttt{true}$.

Example 5.3: We show the decryption at client following Fig. 5 c). The encrypted message R_k client receives aggregates two R_is. The client first generates $(g^{-x})^{2\times 2}$, computes $R'_k = R_k \times g^{-4x} \pmod p$, and finally computes $R = R'_k \bmod q$. The result is 0 that indicates Q is a subgraph of G.

Decomposition scheme. Once the number of R_i aggregated by R_k exceeds a predetermined value, SPMatch will result in incorrect answer. The reason leading to this problem is the multiplications when aggregating R_i into R_k in Line 5 of Algo. 3. Recall that in the decryption, the client needs to compute the encoded message R'_k after receiving R_k, once R'_k exceeds p, the client can *never* recover the final result R by modular q correctly. We can overcome this limitation by determining the maximum number of R_is that can be aggregated in R_k, denoted as ω. We have the following formula:

$$\begin{array}{rlll} & \mathsf{Len}(R'_i) & = & 2 \times (\mathsf{Len}(q) + \mathsf{Len}(r)) + \log(m^2) \\ & \mathsf{Len}(p) & \geq & \omega \times \mathsf{Len}(R'_i) \\ \Leftrightarrow & \omega & \leq & \frac{\mathsf{Len}(p)}{\mathsf{Len}(R'_i)} \end{array} \quad (5)$$

where $m = |V(Q)|$, $\mathsf{Len}(x)$ is the size of the value x, and R'_i is the message encoded in R_i, *i.e.*, $R'_i = \mathsf{Dec}(R_i, x, g)^2$. In particular, with reference to Algo. 3, $(\mathsf{Len}(q) + \mathsf{Len}(r))$ is the largest size of the message encoded in each entry of $\mathbf{M}_{\mathbf{Q}_k}$ and $\bar{\mathbf{C}}_i$. The size of their product (Line 2) is $2(\mathsf{Len}(q) + \mathsf{Len}(r))$. There are m^2 additions of such products (Line 3), hence, Algo. 3 requires at most $\log(m^2)$ carry bits. This gives us the largest size of an R'_i. Then, the size of ω R'_i values must be smaller than that of p, and we obtain the inequality in Formula 5. Having computed ω, the $\mathcal{SP}$ *decomposes* R_k *into a number of aggregated messages, each of which is a product of at most* ω R_i*s*.

False positive. When performing SPMatch, we find that two operations introduce false positive: (1) additions with computing R_i (Line 3); and (2) multiplications with computing R_k in each decomposition (Line 5). We prove that the probabilities of the above two false positive are negligible. Next, we first analyze the probability of false positive from the additions with computing R_i.

Proposition 5.1: The probability of false positive in R_i is negligible. □

Proof: The probability of false positive in R_i is

$$\begin{array}{rcl} \Pr(\text{false positive in } R_i) & = & \Pr(r_1 + \cdots + r_{m^2} = 0 (\bmod q)) \\ & = & \frac{1}{q}, \end{array} \quad (6)$$

where $m = V(Q)$, and q is a large prime number, *e.g.*, 32bits. Thus, the probability is negligible in practice. □

Based on Prop. 5.1, we are able to analyze the probability of false positive with computing the R_k in each decomposition.

Proposition 5.2: The probability of false positive in R_k is negligible in each decomposition. □

Proof: The probability of false positive in each R_k is

$$\begin{array}{rcl} \Pr(\text{false positive in } R_k) & = & \Pr(\text{false positive in all its } R_i) \\ & = & 1 - (1 - \frac{1}{q})^{\omega} \\ & \approx & 1 - e^{-\frac{\omega}{q}}, \end{array} \quad (7)$$

where ω is the size of the decomposition. Since $\omega \ll q$, the probability is negligible in practice. □

5.2 Structure-Preserving Enumeration

The mappings ($\mathbf{M}_i$s) processed by SPMatch are enumerated by SPEnum. Since the worst case number of all possible mappings $\mathbf{M}_i$s from $\mathbf{M}$ (Lines 7-12, Algo. 2) is $O(n^m)$, it has been a crucial task of SPsubIso to prune the search of *useless* $\mathbf{M}_i$s. For instance, we show a scenario of *useless* enumerations by using the LHS of Fig. 6. There are four subgraphs of G in grey, which are disconnected from each other. In the example, only 4 mappings out of 4^6 are possible and the remaining enumerated mappings are useless. However, since both G and Q are encrypted, the $\mathcal{SP}$ can only blindly enumerates those mappings even they may appear "certainly" invalid.

Therefore, in this subsection, we propose SPEnum that consists of a *protocol* between the $\mathcal{SP}$ and the client to prune some *useless partial* mappings. However, due to the pruned enumerations, a little non-trivial structural information may be leaked. Such information leakage can be well controlled by determining how often the client informs the pruning (to be analyzed in Sec. 6.2).

5.2.1 Mapping Enumeration as a Search Tree

To facilitate the discussions on pruning, we view the search of possible subgraph isomorphic mappings from Q to G (in the LHS of Fig. 6) as a search tree, as in the literature of optimizations. A sketch is shown in the RHS of Fig. 6. Each internal node in the d-th level represents a *partial* mapping $\mathbf{M}_i$, denoted as $\mathbf{M}'_i$, whose enumeration is *only* up to the first d rows of $\mathbf{M}$. We denoted Q' as the *induced subgraph* of Q from the first d vertices of Q and G' as the subgraph that Q' maps to, under $\mathbf{M}'_i$. In the example, the query size is 6, thus the height of the search tree is 6. The fanout of each internal node in d-th level equals to the number of 1s in the $(d+1)$-th row of $\mathbf{M}$. Each leaf node of the search tree represents a

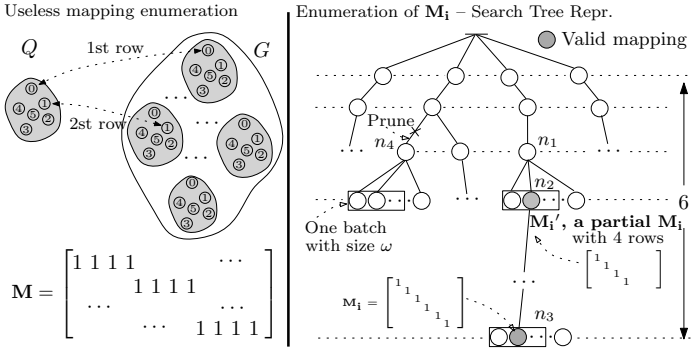


Fig. 6. Illustration of mapping enumerations and SPEnum.

complete $\mathbf{M}_i$. Without pruning, TEnum enumerates all 4^6 leaf nodes.

5.2.2 SPEnumeration

We next present SPEnum. It adopts TEnum and SPMatch, and determines if a partial mapping $\mathbf{M}'_i$ between Q'_k and $\bar{G}'_k$ is a valid to be expanded or not. The inputs of SPMatch are (1) $\mathbf{M}'_i$, which is the current partial mapping in d-th level; (2) Q'_k and $\bar{G}'_k$, which are the induced subgraph of Q_k and $\bar{G}_k$ with the first d vertices, respectively; and (3) R_k, which is the same as before. Then, a *naive* protocol involving the client is described as follows:

1. $\mathcal{SP}$ enumerates an $\mathbf{M}'_i$ and performs SPMatch. Its result R_k is transmitted to the client for decryption *immediately*.
2. The client decrypts R_k. If it is 0, Q'_k is subgraph isomorphic to $\bar{G}'_k$ and $\mathbf{M}'_i$ is valid. The client then notifies the $\mathcal{SP}$ to repeat Step 1 to continue the enumeration from $\mathbf{M}'_i$. Otherwise, the search subtree at $\mathbf{M}'_i$ can be pruned.
3. $\mathcal{SP}$ terminates SPEnum when a valid $\mathbf{M}_i$ is found or no more $\mathbf{M}_i$s can be enumerated.

Protocol with aggregated messages. The naive protocol introduces additional network communications between the client and the $\mathcal{SP}$. To control such communication overhead, we adopt a *breath first search* (BFS) to TEnum and exploit the aggregation supported by CGBE. Specifically, suppose SPEnum is at the d-th level of the search tree, it conducts two rounds of communications.

1. In the first round, for each internal nodes n with a valid partial mapping at the d-th level (*e.g.*, n_1 and n_4 at the 3rd level in Fig. 6), the $\mathcal{SP}$ processes its child nodes of n in *batches* of the size ω. For each batch, the $\mathcal{SP}$ performs SPMatch at each node and aggregates ω R_is into R_k. Recall that ω is the number of messages that CGBE correctly aggregates (see SubSec. 5.1). The $\mathcal{SP}$ sends all the R_ks to the client.
   The client decrypts all R_ks. (i) If it is 0, there exists at least one child node n' of n such that its partial mapping is valid (*e.g.*, n_1). (ii) Otherwise, there is no valid partial mapping of the n's child nodes. The search subtree of n can be safely pruned (*e.g.*, n_4).
2. In the second round, the $\mathcal{SP}$ then sends all R_is of the child nodes of each batch that contains valid partial mappings. to the client, and determines at which node(s) the search proceeds (*e.g.*, n_2). Step 1 is then repeated at the $(d+1)$-th level (*e.g.*, at the 4th level).
3. $\mathcal{SP}$ terminates SPEnum as in the naive protocol.

Remarks. It is worth noting that CGBE limits ω R_i to be aggregated into R_k. If SPMatch utilizes other encryption scheme, such a limitation may not be necessary. For example, FHE does not limit the number of aggregated messages, but its computation cost is significant. Moreover, the number of communication rounds between the $\mathcal{SP}$ and the client is $O(2|V(Q)|)$. In practice, most of the partial mappings of internal nodes are invalid, which result in the size of messages for each round small, *e.g.*, 16KB in worst case for our experiments with default setting.

Vertex ordering. Finally, we remark that vertex ordering (the row ordering of $\mathbf{M}$) significantly affects subgraph query performance, consistent to the literature (*e.g.*, [14]). However, SPEnum cannot exploit structural information to choose an ordering by heuristics. We thereby order the rows of $\mathbf{M}$ by the number of 1s in ascending order. Suppose that partial mappings are mostly useless, such an ordering prune useless enumerations with fewer communications in practice.

5.3 Structure-Preserving Refinement

In order to further optimize the search, *i.e.*, to reduce the number of possible mappings, in this subsection, we propose a *static index* $\mathsf{SI_Q}$ ($\mathsf{SI_G}$) of a Q (G) that indexes h-*hop information*. We modify Refine into inner products between $\mathsf{SI_G}$ and $\mathsf{SI_Q}$ as TRefine. SPRefine achieves privacies by using *asymmetric scalar product preserving encryption* ASPE [32].

5.3.1 Static Index SI

The classical refinement reduces the number of 1s in $\mathbf{M}$ (*a.k.a* flips the 1s to 0s) to minimize the large search space of subgraph isomorphism mappings. Further to reducing the number of 1s in $\mathbf{M}$, SPRefine cannot expose structural information. To address these, we propose to index vertices with their h-hop information. The design requirements of such index are that they can be computed before a query is run and hence, no knowledge is gained by executing the query algorithm. Moreover, they should be efficiently computed, as the client may not have powerful machines. Since the index flips the 1s in $\mathbf{M}$ by information of the vertices that are h hop away. The $\mathcal{SP}$ cannot precisely determine the cause of the flips. Next, we define the *h-hop vertices of a vertex* v as follows.

Definition 5.5: The *h-hop vertices of* v ($\mathsf{h\text{-}Hop}(v)$) is the set of vertices that are reachable from v by *exactly* h hops. The *h-hop ℓ-labeled vertices* of v ($\mathsf{h\text{-}Hop}_\ell(v)$) is

$$\{v' \mid L(v') = \ell \text{ and } v' \in \mathsf{h\text{-}Hop}(v)\ \}.$$

□

Fig. 7 a) illustrates the notations of h-hop vertices and h-hop ℓ-labeled vertices of v. We assume a canonical ordering of possible labels and hence, present labels as integers. Given a graph G, suppose $h = 2$, $\mathsf{2\text{-}Hop}(v) = \{v_2, v_3\}$ and, $\mathsf{2\text{-}Hop}_0(v) = \{v_3\}$.

For illustration purposes, we choose four structural information for indexing used in SPRefine. We can determine if two vertices can be mapped or not from them. It is worth remarking that one can further extend other information for indexing. We define such information as an attribute of the set $\mathsf{h\text{-}Hop}_\ell(v)$ as follows:

- h-Hop$_\ell(v)$.MaxDeg is the maximum degree of v', $v' \in$ h-Hop$_\ell(v)$;
- h-Hop$_\ell(v)$.Occur is $|$h-Hop$_\ell(v)|$;
- h-Hop$_\ell(v)$.PreLabel is a set of labels of the parents of occurred h-Hop$_\ell(v)$; and
- h-Hop$_\ell(v)$.Sup is the number of different paths that can reach from v to v', where $v' \in$ h-Hop$_\ell(v)$.

Example 5.4: We continue to discuss the example in Fig. 7 a). Suppose $h = 2$. Recall that 2-Hop$_0(v)$ = $\{v_3\}$. We list some 2-hop information as follows: (1) 2-Hop$_0(v)$.MaxDeg $= 2$, since Deg$(v_3, G) = 2$; (2) 2-Hop$_0(v)$.Occur $= 1$, since only one label with 0 in 2-Hop$_0(v)$; (3) 2-Hop$_0(v)$.PreLabel $= \{0\}$ as 0 is the only label of the parents of $\{v_3\}$; and (4) 2-Hop$_0(v)$.Sup $= 1$ because there is only one path that can reach from v to v_3.

Encoding h-hop information in static index. The *static index* of G is denoted as SI$_G$. For all v, h, and ℓ, $h \le$ maxH, maxH is a user-specified maximum hop size, SI$_G[v][h][\ell]$ is a bit vector. In the four h-hop information defined above, we identify two types. They are encoded in SI$_G$ as follows.

(1) *Label set* (*e.g.*, PreLabel): for each $\ell' \in$ h-Hop$_\ell(v)$.PreLabel $\Rightarrow$ SI$_G[v][h][\ell]$.PreLabel$[\ell'] = 1$, otherwise 0; and
(2) *Numerical data* (*e.g.* MaxDeg, Occur and Sup): We present the encoding of MaxDeg for illustration. Those of Occur and Sup are similar. We denote the maximum value for MaxDeg as MaxDeg$_{max}$. For each $i \le$ MaxDeg$_{max}$ and $i \le$ h-Hop$_\ell(v)$.MaxDeg $\Rightarrow$ SI$_G[v][h][\ell]$.MaxDeg$[i] = 1$, otherwise 0.

The bit vector SI$_G[v][h][\ell]$ is then simply a concatenation of SI$_G[v][h][\ell]$.MaxDeg, SI$_G[v][h][\ell]$.Occur, SI$_G[v][h][\ell]$.PreLabel and SI$_G[v][h][\ell]$.Sup. The bit vector SI$_G[v]$ is accordingly a concatenation of all SI$_G[v][h][\ell]$s for all v, $h \le$ maxH and ℓ.

Example 5.5: Fig. 7 a) shows a simple example of the partial SI$_G[v][h][\ell]$ for v in G, where $h = 2$, $\ell = 0$. We preset the default maximum value for MaxDeg, Occur and Sup to 3. We assume that the possible labels are 0 and 1. (1) For PreLabel, since 2-Hop$_0(v)$.PreLabel $= \{0\}$, then SI$_G[v][2][0]$.PreLabel$[0] = 1$, and SI$_G[v][2][0]$.PreLabel$[1] = 0$; and (2) For MaxDeg, as 2-Hop$_0(v)$.MaxDeg $= 2$, thereby SI$_G[v][2][0]$.MaxDeg$[1] =$ SI$_G[v][2][0]$.MaxDeg$[2] = 1$.

The h-hop information abovementioned can be generated by a simple depth first traversal starting at each vertex on the data graph offline and on the query by the client on the fly. Due to space restrictions, we omit the verbose algorithm.

5.3.2 Inner Products of Static Indexes

With the static index SI, we establish the refinement of possible subgraph isomorphism mappings by the following proposition:

Proposition 5.3: Given a user-specified maxH, $\forall\ v_j \in V(Q)$ and $v_k \in V(G)$, $\mathbf{M}(j,k) = 1$, *iff* the following of the h-hop information of v_j and v_k hold: $\forall \ell \in \Sigma(G)$, $h \le$ maxH,

- h-Hop$_\ell(v_j)$.MaxDeg $\le$ h-Hop$_\ell(v_k)$.MaxDeg;
- h-Hop$_\ell(v_j)$.Occur $\le$ h-Hop$_\ell(v_k)$.Occur;
- h-Hop$_\ell(v_j)$.PreLabel $\subseteq$ h-Hop$_\ell(v_k)$.PreLabel; and

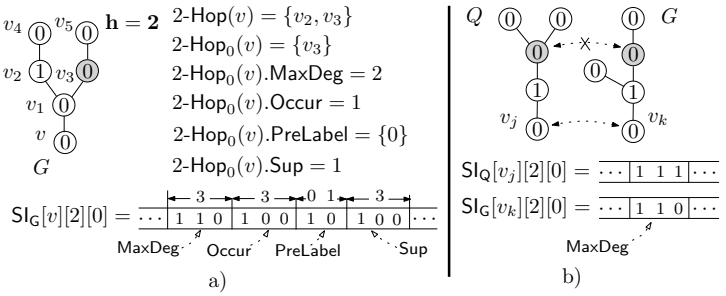


Fig. 7. (a) Illustration of the h-hop vertices and static index; and (b) an refinement by the index

- h-Hop$_\ell(v_j)$.Sup $\le$ h-Hop$_\ell(v_k)$.Sup.

□

Prop. 5.3 can be obtained from a proof by contradiction.

Example 5.6: We use Fig. 7 b) to illustrate the underlying idea of Prop. 5.3. For simplicity, we only show the effect of MaxDeg. Before the refinement, $\mathbf{M}(j,k) = 1$ since $L(v_j) = L(v_k)$. Since 2-Hop$_0(v)$.MaxDeg of Q and G are 3 and 2, respectively. Hence, 2-Hop$_0(v_j)$.MaxDeg $\not\le$ 2-Hop$_0(v_k)$.MaxDeg. By Prop. 5.3, v_j cannot be mapped to v_k and $\mathbf{M}(j,k)$ is flipped to 0.

Therefore, TRefine further transforms Prop. 5.3 into the inner product as follows.

Proposition 5.4: Given a user-specified maxH, $\mathbf{M}(j,k) = 1$, $v_j \in V(Q)$ and $v_k \in V(G)$, *iff* the following of SI of v_j and v_k hold: $\forall \ell \in \Sigma(G)$, $h \le$ maxH,

SI$_Q[v_j][h][\ell] \cdot$ SI$_Q[v_j][h][\ell] =$ SI$_Q[v_j][h][\ell] \cdot$ SI$_G[v_k][h][\ell]$. □

Example 5.7: We illustrate the Prop. 5.4 with the Example 5.6 in Fig. 7 b), the partial SI of both Q and G are shown. Since SI$_Q[v_j][2][0] \cdot$ SI$_Q[v_j][2][0] \neq$ SI$_Q[v_j][2][0] \cdot$ SI$_G[v_k][2][0]$, then $\mathbf{M}(j,k)$ is flipped to 0.

Note that we can further simplify the inner product in Prop. 5.4 to SI$_Q[v_j] \cdot$ SI$_Q[v_j] =$ SI$_Q[v_j] \cdot$ SI$_G[v_k]$, where SI$_Q[v_j]$ is the concatenation for all SI$_Q[v_j][h][\ell]$s. Therefore, Line 19 of TRefine is mainly one inner product between SI$_Q[v_j]$ and SI$_G[v_k]$, using Prop. 5.4 for pruning the 1s in $\mathbf{M}$.

For SPRefine, we encrypt SIs as: $\forall v_j \in V(Q)$ and $\forall v_k \in V(G)$, SI$_{Q_k}[v_j] =$ ASPE(SI$_Q[v_j]$) and SI$_{G_k}[v_k] =$ ASPE(SI$_G[v_k]$). The secret keys held by $\mathcal{SP}$ and the client are the same to that of [32]. Finally, SPRefine is TRefine after replacing Line 19 with a *private inner product* between encrypted bit vectors (SI$_{Q_k}$ and SI$_G$), supported by ASPE.

We close this section with a remark that SPEnum and SPRefine may expose little non-trivial information in the sense that the probability of guessing the structure of a graph is not that of a random guess anymore. We shall analyze their probabilities in Sec. 6.2.

6 Privacy Analysis

In this section, we prove the privacy of the encryption method and then the query algorithm SPsubIso. The attack model is defined in Sec. 2 that we assume the attackers or $\mathcal{SP}$s are the eavesdroppers and can adopt the chosen plaintext attack (CPA) [21].

6.1 Privacy of the Encryption Method

Two encryption methods are used in this paper. (1) CGBE scheme is proposed to encrypt $\mathbf{M_Q}$ and $\mathbf{M_G}$, and (2) ASPE [32] is adopted to encrypt $\mathsf{SI_Q}$ and $\mathsf{SI_G}$. We first state that both the CGBE and ASPE schemes are secure against CPA and then establishe that the structures of the query and the graph are protected against our attack model. Denote $\hat{g}$ to be *an arbitrary chosen from* $\mathbb{G}$.

Lemma 6.1: [21] Let $\mathbb{G}$ be a finite group, and let $m \in \mathbb{G}$ be arbitrary. Then, choosing random $g \in \mathbb{G}$ and setting $g' = m \cdot g$ gives the same distribution for g' as choosing random $g' \in \mathbb{G}$. *I.e.,* for any $\hat{g} \in \mathbb{G}$

$$\Pr[m \cdot g = \hat{g}] = 1/|\mathbb{G}|,$$

where the probability is taken over random choice of g. □

Lemma 6.2: Let $\mathbb{G}$ be a finite group, and let $g \in \mathbb{G}$ be arbitrary. Then choosing random $r \in [0, |\mathbb{G}|]$ and setting $g' = g^r$ gives the same distribution for g' as choosing g' from $\mathbb{G}$. *I.e.,* for any $\hat{g} \in \mathbb{G}$

$$\Pr[g^r = \hat{g}] = 1/|\mathbb{G}|,$$

where the probability is taken over random choice of r. □

Proof: We prove the lemma in a similar style of the proof [21] of Lemma 6.1. Let $\hat{g} \in \mathbb{G}$ be arbitrary. Then

$$\Pr[g^r = \hat{g}] = \Pr[r = \log_g \hat{g}]$$

Since r is chosen uniformly at random, the probability that r is equal to the fixed element $\log_g \hat{g}$ is exactly $1/|\mathbb{G}|$. □

Lemma 6.3: CGBE is secure against CPA. □

Proof: We prove that the proposed CGBE scheme has indistinguishable encryptions in the presence of the eavesdroppers, which is implied by the definition of CPA secure [21].

Specifically, choosing a random value r, and letting $r' \in \mathbb{G}$ such that $g^{r'} = r$, we have $\mathsf{Enc}(m, g, x) = mrg^x = mg^{x+r'}$. First, by Lemma 6.2, $\Pr[g^{x+r'} = \hat{g}] = 1/|\mathbb{G}|$, where $\hat{g}$ is arbitrary chosen from $\mathbb{G}$. Then, by Lemma 6.1, $\Pr[mg^{x+r'} = \hat{g}] = 1/|\mathbb{G}|$. Therefore, the ciphertext in the CGBE scheme is a *uniformly distributed* group element and, in particular, is independent of the message m being encrypted, *i.e.,* $\Pr[mrg^x = \hat{g}] = 1/|\mathbb{G}|$. That means the entire ciphertext contains no information about m. Given the above, CGBE is secure against chosen plaintext attack. □

Since CGBE is a secure encryption scheme against CPA, $\mathcal{SP}$ can never attack the $\mathbf{M}_{\mathbf{Q}_k}$ and $\mathbf{M}_{\bar{\mathbf{G}}k}$ without possessing the secret key against our attack model.

Lemma 6.4: $\mathbf{M}_{\mathbf{Q}_k}$ and $\mathbf{M}_{\bar{\mathbf{G}}k}$ are preserved from $\mathcal{SP}$ against the attack model under CGBE. □

Proof: The proof is a direct application of Lemma. 6.3. Since CGBE is secure against CPA, $\mathbf{M}_{\mathbf{Q}_k}$ and $\mathbf{M}_{\bar{\mathbf{G}}k}$ are secure against the attack model under CGBE. □

Next, we state that $\mathsf{SI}_{\mathsf{Q}_k}$ and $\mathsf{SI}_{\mathsf{G}_k}$ are preserved from $\mathcal{SP}$.

Lemma 6.5: $\mathsf{SI}_{\mathsf{Q}_k}$ and $\mathsf{SI}_{\mathsf{G}_k}$ are preserved from $\mathcal{SP}$ against the attack model under ASPE. □

$\mathsf{SI_Q}[v_j]$ and $\mathsf{SI_G}[v_k]$ are encrypted by ASPE, where $v_j \in V(Q)$ and $v_k \in V(G)$. Since ASPE is secure against CPA [32], it is immediate that Lemma 6.5 is true.

Theorem 6.1: The structure of both Q and G are preserved from $\mathcal{SP}$ against our attack model under CGBE and ASPE. □

Proof: The proof can be deduced from Lemmas 6.4 and 6.5. Recall that $Q_k = (V, \mathbf{M}_{\mathbf{Q}_k}, \Sigma, L, \mathsf{SI}_{\mathsf{Q}_k})$ and $\bar{G}_k = (V, \mathbf{M}_{\bar{\mathbf{G}}k}, \Sigma, L, \mathsf{SI}_{\mathsf{G}_k})$. By Lemmas 6.4 and 6.5, the $\mathcal{SP}$ cannot break Q_k and $\bar{G}_k$ since the structures of Q_k and $\bar{G}_k$ (*i.e.,* $\mathbf{M}_{\mathbf{Q}_k}$, $\mathsf{SI}_{\mathsf{Q}_k}$, $\mathbf{M}_{\bar{\mathbf{G}}k}$ and $\mathsf{SI}_{\mathsf{G}_k}$) are secure against CPA. □

6.2 Privacy of SPsubIso

As presented in Sec 5, SPsubIso contains three main steps. We analyze the privacy of each of these steps in this subsection. Before we present the analysis, we clarify some notations. Given Q and G, $m = |V(Q)|$ and $n = |V(G)|$. The function $P(n)$ returns the number of all possible graphs generated by n vertices, *i.e.,* $P(n) = 2^{n^2}$. The function $\mathcal{A}(G)$ returns 1 if $\mathcal{SP}$ can determine the exact structure of G, and 0 otherwise. *The probability that the $\mathcal{SP}$ can determine the structure of the graph G is denoted as Pr*$[\mathcal{A}(G) = 1]$. Given a graph G with n vertices, the probability to determine the graph structure by a *random guess* is $\Pr[\mathcal{A}(G) = 1] = \frac{1}{P(n)}$.

Proposition 6.1: Under SPMatch, $\Pr[\mathcal{A}(Q)=1] = \frac{1}{P(m)}$, and $\Pr[\mathcal{A}(G) = 1] = \frac{1}{P(n)}$, which are equivalent to random guess. □

Proof: (1) First we prove that the $\mathcal{SP}$ can never determine any structural information from the computations in each step of SPMatch. Recall that each SPMatch comprises a *constant* number of mathematical operations in the encrypted domain in Algo. 3:

- Line 2 invokes a *constant* number m^2 of multiplications of $\mathbf{M}_{\mathbf{Q}_k}$ and $\mathbf{C}_i$;
- Line 3 requires a *constant* number m^2 of additions in $\mathbf{MC}_i$; and
- Line 4 conducts *one* multiplication R_i and R_k.

Further, by Lemma 6.3, all the intermediate computation results are securely protected against the attack model. Thus, $\mathcal{SP}$ cannot learn any structural information from these steps.

(2) Next, given any two SPMatchs, the $\mathcal{SP}$ only knows that each SPMatch aggregates its R_i into R_k by one multiplication. Similarly, by Lemma 6.3, no other information can be learned from the R_i or R_k by the $\mathcal{SP}$.

Putting the above together, the $\mathcal{SP}$ does not learn the structures of Q or G by invoking SPMatchs and the probability of determining a structure is equivalent to that of random guess. □

Proposition 6.2: Under SPEnum, the following holds:

- If Q_d is subgraph isomorphic to G_d, there is no information leakage, *i.e.*,
  $\Pr[\mathcal{A}(Q_d) = 1] = \Pr[\mathcal{A}(G_d) = 1] = \frac{1}{P(d)}$; and
- Otherwise,
  $\Pr[\mathcal{A}(Q_d) = 1] = \Pr[\mathcal{A}(G_d) = 1] = \frac{1}{(P(d) - P(d-1))}$,

where Q_d (resp., G_d) is the induced subgraph of Q (resp., G) that contains the mapped d vertices specified by the partial mapping $\mathbf{M}'_i$ enumerated up to the level d. □

Proof: Recall that $\mathbf{M}_{\mathbf{Q}k}$ and $\mathbf{M}_{\bar{\mathbf{G}}k}$ are preserved, by Lemma 6.4 and Prop. 6.1. Hence, we only consider the information that the $\mathcal{SP}$ can gain from the protocol in SPEnum. Only Q_d and G_d are analyzed as the remaining subgraphs ($Q - Q_d$ and $G - G_d$) are not yet processed by the $\mathcal{SP}$. By the protocol of SPEnum, the client informs the $\mathcal{SP}$ at the d-th level of the search tree, the $\mathcal{SP}$ knows that the nodes at the d-th level, say v_j and v_k in Q and G, cause a violation is detected or not. We thereby consider these two exhaustive cases as follows:

Case 1: If Q_d is subgraph isomorphic to G_d, there is *no* violation between Q_d and G_d. Recall Formula 1, a violation occurs when v_j is connected to some vertices (under $\mathbf{M}_{\mathbf{Q}}$) but v_k does not have corresponding edges (under $\mathbf{C}_i$). When there is no violation, v_j *may or may not* be connected to other vertices in Q_d. The $\mathcal{SP}$ cannot distinguish this because the edges of v_j (in $\mathbf{M}_{\mathbf{Q}k}$) is preserved. Similarly, the $\mathcal{SP}$ does not learn any information about the edges of v_k of G_d neither. Hence, there is no information leakage; and

Case 2: If Q_d is *not* subgraph isomorphic to G_d, there is a violation between Q_d and G_d. Hence, the $\mathcal{SP}$ knows Q_d and G_d do not falsify Formula 1. However, if v_j is isolated in Q_d, the first predicate of Formula 1 is *always* false; and if v_k is connected to all other vertices in G_d, the second predicate of Formula 1 is *always* false. Contrarily, other than the above two scenarios, the $\mathcal{SP}$ cannot be certain the cause of the violation, as both $\mathbf{M}_{\mathbf{Q}k}$ and $\mathbf{M}_{\bar{\mathbf{G}}k}$ are protected. The above scenarios affect the probabilities as follows.

- v_j is *isolated* in Q_d, *i.e.*, $\forall v'_j \in V(Q_d)$, $v'_j \neq v_j$, $(v_j, v'_j) \notin E(Q_d)$. Then, the possible number of Q_d with isolated v_j is $P(d-1)$. Thus, the probability that the $\mathcal{SP}$ determines Q_d is $\Pr[\mathcal{A}(Q_d) = 1] = \frac{1}{(P(d)-P(d-1))}$; and
- v_k is connected to all other vertices in G_d, *i.e.*, $\forall v'_k \in V(G_d)$, $v'_k \neq v_k$, $(v_k, v'_k) \in E(G_d)$. Then, the possible number of G_d with v_k connecting to all other vertices is $P(d-1)$. Therefore, the probability that $\mathcal{SP}$ determines G_d is $\Pr[\mathcal{A}(G_d) = 1] = \frac{1}{(P(d)-P(d-1))}$.

Consider multiple SPEnum calls. *Case 1* does not leak information, whereas the enumerations beyond *Case 2* are pruned. In either case, an SPEnum call will not affect another. □

Proposition 6.3: Under SPRefine, the following holds:

- If $\mathbf{M}(j,k)$ is not flipped, there is no information leakage; and
- Otherwise,

$$\begin{aligned} \Pr[\mathcal{A}(Q) = 1] &= \frac{P(a+1)}{P(m)(P(a+1)-1)}, \text{and} \\ \Pr[\mathcal{A}(G) = 1] &= \frac{P(b+1)}{P(n)(P(b+1)-1)}, \end{aligned} \tag{8}$$

where $a = |\mathsf{MaxDeg}(Q)|^{\mathsf{maxH}}$, $b = |\mathsf{MaxDeg}(G)|^{\mathsf{maxH}}$, and $\mathsf{MaxDeg}(G)$ is the maximum degree of the vertices of G. □

Proof: (Sketch) Due to space limitation, we present the proof idea here and the detailed derivation in Appendix B. The proof of Proposition 6.3 is again established by a case analysis. The SI are protected by the encryption and its operations. However, when $\mathbf{M}(j,k)$ is flipped, the $\mathcal{SP}$ is certain that there is a violation because of v_j and v_k. Hence, the rest of the analysis is similar to that of *Case 2* of Proposition 6.2. □

Finally, we remark that Props. 6.2 and 6.3 state that the client may tune the privacy offered by SPsubIso by varying the variables maxH and d of SPEnum and SPRefine. Further, the values of MaxDeg and maxH (and therefore a and b) are not known to the $\mathcal{SP}$. We use these values in Prop. 6.2 to simply quantify the privacy. In our experiment, we confirmed that SPEnum and SPRefine are effective optimizations and we may set these variables to balance privacy and performances.

7 Experimental Evaluation

In this section, we present a detailed experimental evaluation to investigate the performance of our techniques on both real world and synthetic datasets. Due to space restriction, we elaborate our findings with real world datasets and report the result from synthetic datasets in Appendix A.

7.1 Experimental Setup

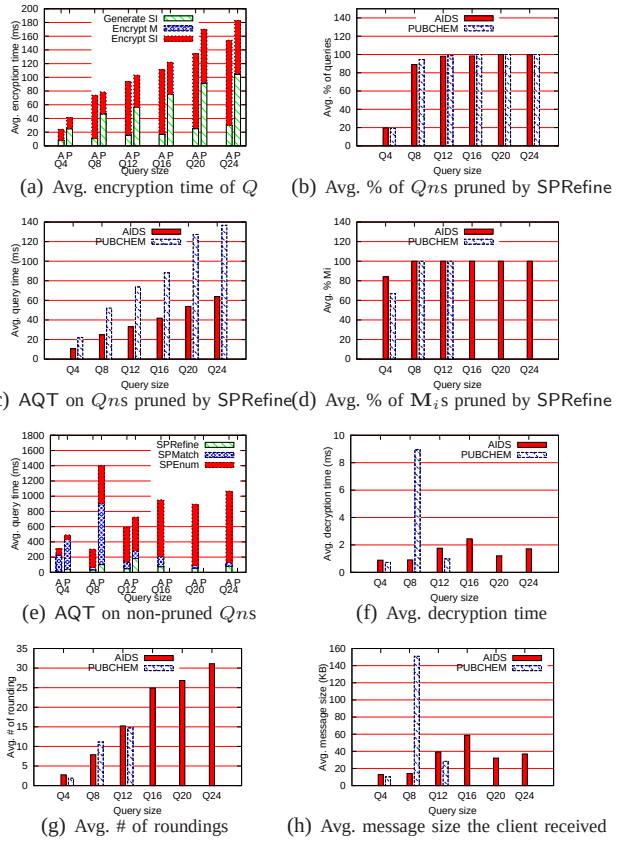
The platform. We set up the $\mathcal{SP}$ as a server at Amazon EC2, equipped with a 2.8GHz CPU and 16GB memory running Ubuntu 14.04 OS. The client is a local machine with a 3.4GHz CPU and 16GB memory running Win 7 OS. For ease of exposition, we assume the $\mathcal{DO}$ has a machine with the same setting, to encrypt data graphs. The client is connected to an Ethernet. All techniques were implemented on the GMP library (C++). By default, our CGBE uses 2048 bits; the sizes of the prime number q and the random number r are both set to 32bits. The decomposition size ω is 15. Our ASPE implementation is set accordingly to [32]. We have implemented a FHE-based solution. Its performance is always at least one order of magnitude slower than CGBE's. Thus, we do not report their numbers here.

Datasets. We used two real-world benchmark datasets namely Aids (A) and PubChem (P) from [15], which are widely used in [5], [14], [16], [23], [29], [33], [36], [39]. As our discussions focused on vertex labels, without loss of generality, we remove the edge labels. Aids consists of 10,000 graphs, which are drawn from a real antiviral dataset [28]. On average, each graph in Aids has 25.42 vertices and 27.40 edges. The number of distinct vertex labels is 51. PubChem consists of 1 million graphs, which are drawn from a real chemical database [27]. Each graph in PubChem has 23.98 vertices and 25.76 edges, on average. The number of distinct vertex labels is 81.

Query sets. For each of the aforementioned datasets, we used its existing query sets $Q4$, $Q8$, $Q12$, $Q16$, $Q20$ and $Q24$, which can be downloaded from [15]. Each Qn contains 1,000 query graphs, where n is the number of edges for each query.

Test runs. The queries were generated from random sampling of the above datasets and their associated query sets. For each dataset and query set Qn, we randomly sampled 1,000 graphs and 10 query graphs, *i.e.*, for each Qn, we performed 10,000 subgraph isomorphism testings. In addition, the average densities of the sample graphs and queries are the same as those of the original data and query sets, respectively. We report the average of the test runs by default. We use the abbreviation AQT for average query time.

Default values of parameters. The parameters used in SPRefine and SPEnum are set as follows. We set the default



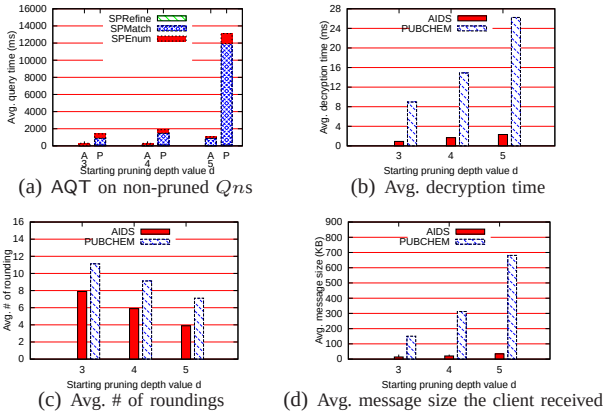
(a) Avg. encryption time of Q (b) Avg. % of Qns pruned by SPRefine

(c) AQT on Qns pruned by SPRefine (d) Avg. % of $\mathbf{M}_i$s pruned by SPRefine

(e) AQT on non-pruned Qns (f) Avg. decryption time

(g) Avg. # of roundings (h) Avg. message size the client received

Fig. 8. Performance on varying query sizes.



(a) AQT on non-pruned Qns (b) Avg. decryption time

(c) Avg. # of roundings (d) Avg. message size the client received

Fig. 9. Effectiveness of the starting pruning depth d.

maxH, and maximum values for MaxDeg, Occur, and Sup to 6. We set the starting pruning depth d of the protocol of SPEnum to 3.

7.2 Experiments on Real Datasets

7.2.1 *Performance by Varying Query Sizes*

We first show the performance of various query sizes in Fig. 8.

Encryption time by varying query sizes. We report the average encryption times in Fig. 8(a). The encryption time of a query Q involves (1) the time for generating SI_Q; (2) the time of encryption of $\mathbf{M}_\mathbf{Q}$ by CGBE; and (3) the time of encryption of SI_Q by ASPE. We observe that the average encryption times are around 100ms and 150ms for Aids and PubChem, respectively. The encryption of $\mathbf{M}_\mathbf{Q}$ by our proposed CGBE is efficient, which only costs several milliseconds on a commodity machine. Further, the query is encrypted *only once*.

Performance at the $\mathcal{SP}$. There are two types of queries in the processing of SPsubIso. The first type of the queries are those *pruned* by SPRefine. Fig. 8(b) reports the percentage of such queries. In particular, we note that the PubChem queries $Q16$-24 are completely pruned. Fig. 8(c) shows the average query time of those pruned queries, which is largely occupied by the private inner product. It is unsurprising that the time increases with the query size. They are smaller than 65ms and 140ms on Aids and PubChem, respectively.

The second type is the *non-pruned* queries that pass SPRefine. For these queries, we report the percentage of pruned possible mappings in Fig. 8(d), which can be calculated by the number of flipped 1s by SPRefine. The average query times are shown in Fig. 8(e). For most queries, the query times are smaller than 1s. The query time of $Q8$ is the longest but it is still smaller than 1.4s.

Performance at the client side. We report the performance at the client side in Fig. 8(f). The times required are tiny, for instance, about 9ms from $Q8$ of PubChem and clearly smaller than 2ms for other queries. The average number of rounds between $\mathcal{SP}$ and client is usually small (Fig. 8(g)). Since many invalid partial mappings are pruned, the total message size sent to the client (Fig. 8(h)) is small (around 150KB in worst case). In each round, at most 16KB of messages are sent.

Comparison with the naïve method. Assume that the whole database was transferred to the client. We run one of the most popular non-indexing subgraph isomorphism algorithms VF2 [7]. The *total* AQT for all query sets on Aids and PubChem at the client side are up to 20ms and 30ms, respectively. In comparison, after the encryption for each query, the computation of our techniques at the client side requires only a few milliseconds on average (Fig. 8(f)). That is, we save most of the computations at the client.

7.2.2 *Effectiveness of* SPEnum

In Fig. 9, we verify the effectiveness of SPEnum by varying the starting pruning depth d to (3, 4, 5). The query set is $Q8$.

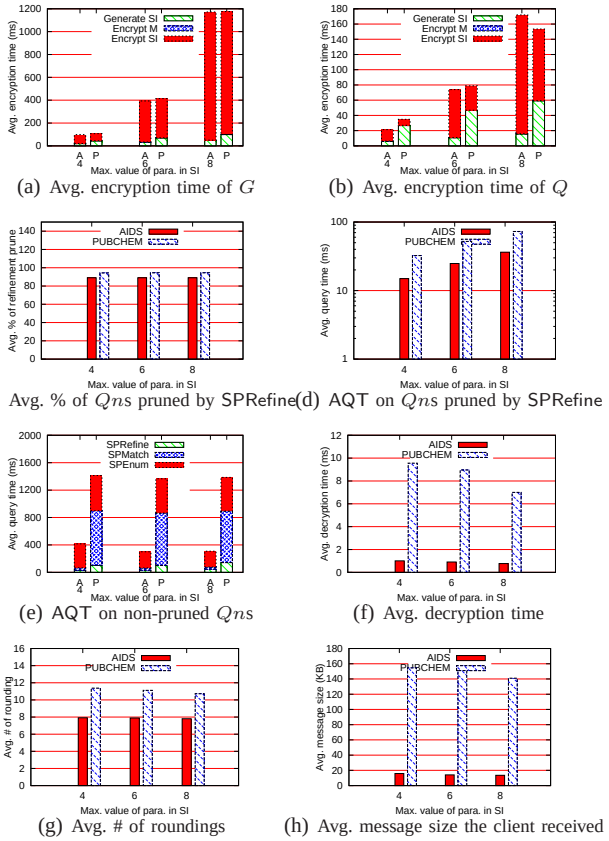
Performance at the $\mathcal{SP}$. Fig. 9(a) shows the query time at $\mathcal{SP}$. It is obvious that as the value d increases, the search space increases, the query time increases.

Performance at the client side. Fig. 9(b) shows the decryption time at the client side increases with d and its trend closely follows that of the query times. The average number of rounds between $\mathcal{SP}$ and client (Fig. 9(c)) decreases as the value d increases because the protocol in SPEnum is a BFS. The message size increases according to d, as shown in Fig. 9(d). However, importantly, by Prop. 6.2, the probabilities that $\mathcal{SP}$ can determine the structures decrease with d increases.

7.2.3 *Effectiveness of* SPRefine

We verify the effectiveness of SPRefine by varying SI. We ranged maxH, and the maximum values for MaxDeg, Occur and Sup from 4 to 8. In this experiment, the query set is $Q8$, and the starting pruning depth d of SPEnum is 3.

Encryption time. Figs. 10 (a) and (b) show the encryption times of G and Q, respectively. As the maximum values increase, the encryption times of both G and Q increase.



(a) Avg. encryption time of G (b) Avg. encryption time of Q

(c) Avg. % of Qns pruned by SPRefine (d) AQT on Qns pruned by SPRefine

(e) AQT on non-pruned Qns (f) Avg. decryption time

(g) Avg. # of roundings (h) Avg. message size the client received

Fig. 10. Effectiveness of SI.

Effectiveness of SPRefine. Fig. 10(c) shows the average percentage of queries that are pruned by SPRefine with different maximum values in SI. We note that the pruning effectiveness on different maximum values are similar to each other, which are almost 96% for queries on both `Aids` and `PubChem`. That means for each $v_j \in V(Q)$, $v_k \in V(G)$, h-Hop$_\ell(v)$ may differ with each other within 4 hops with very high probabilities if $\mathbf{M}(j,k)$ is flipped to 0. However, the $\mathcal{SP}$ has no precise knowledge about the encrypted SIs. Further, by Prop. 6.3, the probability that the $\mathcal{SP}$ can determine the structures decreases as maxH increases.

Performance at the $\mathcal{SP}$. Fig. 10(d) shows the average query time of queries pruned by SPRefine, which mainly involves the time for private inner products. As expected, the times are small. Since the pruning of SI is very similar under different maximum values (by Fig. 10(c)), the query times for those non-pruned queries (the queries pass SPRefine) are similar, shown in Fig. 10(e). The times are around 400ms and 1.4s for `Aids` and `PubChem`, respectively.

Performance at the client side. Since the query times are similar to different maximum values on SI, the decryption times at the client side shown in Fig. 10(f) are also very similar. The average number of rounds between the $\mathcal{SP}$ and the client are shown in Fig. 10(g), which are around 8 and 11 for `Aids` and `PubChem` respectively. The size of the received messages at client is shown in Fig. 10(h), which are around 17KB and 145KB, respectively.

8 Related Work

Privacy-preserving graph query. Cao et al. [2] propose to support subgraph query over an encrypted database with a number of small graphs. Their work protects the privacy of the query, index and data features. This work does not solve the problem of the subgraph isomorphism testings of the candidate graphs. Such testings are required to be performed at the client side. Cao et al. [1] study tree pattern queries over encrypted XML documents. The traversal order for each query is predetermined. In the context of graphs, the order cannot be predetermined. In our recent work [34], [35], we propose privacy-preserving reachability query services over encrypted index and data to preserve both of the query and graph structure, under the same system model of this work. He et al. [17] analyze the vertex reachability of the graph data, with the preservation of edge privacy. Kundu et al. [22] propose a series of methods to verify the authenticity of a *given portion of data* without any leakage of extraneous information about the data (tree/graph/forest). Gao et al. [11] propose neighborhood-privacy protected shortest distance in the paradigm of cloud computing. This method aims to p-reserve all of the neighborhood connections and the shortest distances between each two vertices in outsourced graph data. However, it allows some connection and distance information between vertices to be exposed. Mouratidis et al. [26] propose the shortest path computation with no information leakage by using the PIR [6] protocol. The high computational cost of PIR is known to be a concern. Karwa et al. [20] present some novel algorithms for releasing *subgraph counts* of a graph by satisfying the differential privacy of edges, which requires that the presence or absence of a certain edge be hidden. Fan et al. [10] propose efficient authenticated subgraph query services under the same setting of data outsourcing. In [9], Fan et al. propose an asymmetric structure-preserving subgraph query, where the privacy of the data graphs has been relaxed.

Subgraph isomorphism query. Ullmann [30] is a seminal algorithm for subgraph isomorphism. Its basic idea is *the search with backtracking with respect to the matrix that represents possible isomorphic relationships*. In the recent decade, several algorithms (*e.g.*, VF2 [7], QuickSI [29] and Turbo$_{\text{iso}}$ [14]) have been proposed to enhance the Ullmann's algorithm. They all require to *traverse* the query on graph data. For instance, VF2 [7] relies on a set of state transitions and traversals on the graph and query. QuickSI [29] optimizes the ordering in traversals of graphs. Turbo$_{\text{iso}}$ [14] exploits neighborhood information and local regions of vertices to further optimize query performance. Turbo$_{\text{iso}}$ involves determining an optimal traversal in query processing. However, the traversals themselves carry structural information, which makes privacy preservation complicated if it is possible at all.

9 Conclusion

This paper presents the first work on query services for *structure-preserving subgraph isomorphism* (SPsubIso). SPsubIso comprises three major steps: (1) *Structure-preserving matching* (SPMatch) involves a novel *cyclic group based encryption* (CGBE) scheme to compute whether a mapping between Q and G is valid, in an encrypted domain. (2) *Structure-preserving enumeration* (SPEnum) comprises a *protocol* that involves the client for further pruning. (3) *Structure-preserving refinement* (SPRefine) exploits a *static*

index for pruning the search space of possible mappings. Our analysis shows that the structural information is preserved under SPMatch and presents the privacy preservation due to optimizations. Our experiments on both real and synthetic datasets confirm that SPsubIso is efficient. In future work, we will investigate relaxations of privacy requirements (*e.g.*, [9]).

REFERENCES

[1] J. Cao, F.-Y. Rao, M. Kuzu, E. Bertino, and M. Kantarcioglu. Efficient tree pattern queries on encrypted xml documents. In *EDBT*, 2013.

[2] N. Cao, Z. Yang, C. Wang, K. Ren, and W. Lou. Privacy-preserving query over encrypted graph-structured data in cloud computing. In *ICDCS*, 2011.

[3] R. Chen, B. Fung, P. Yu, and B. Desai. Correlated network data publication via differential privacy. *The VLDB Journal*, in press.

[4] J. Cheng, A. W.-c. Fu, and J. Liu. K-isomorphism: privacy preserving network publication against structural attacks. In *SIGMOD*, 2010.

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

[6] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *J. ACM*, 45:965–981, 1998.

[7] L. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *PAMI, IEEE*, 26(10):1367–1372, 2004.

[8] Daylight Chemical Information Systems, Inc. Daylight web services manual. *http://www.daylight.com/dayhtml/doc/webservices/index.html*, 2011.

[9] Z. Fan, B. Choi, J. Xu, and S. S. Bhowmick. Asymmetric structure-preserving subgraph query for large graphs. In *ICDE*, 2015.

[10] Z. Fan, Y. Peng, B. Choi, J. Xu, and S. S. Bhowmick. Towards efficient authenticated subgraph query service in outsourced graph databases. *IEEE Trans. on Services Computing*, 7(4):696–713, 2014.

[11] J. Gao, J. X. Yu, R. Jin, J. Zhou, T. Wang, and D. Yang. Neighborhood-privacy protected shortest distance computing in cloud. In *SIGMOD*, 2011.

[12] C. Gentry. A fully homomorphic encryption scheme. 2009.

[13] H. Hacigumus, B. Iyer, and S. Mehrotra. Providing database as a service. In *ICDE*, 2002.

[14] W.-S. Han, J. Lee, and J.-H. Lee. Turbo$_{iso}$: towards ultrafast and robust subgraph isomorphism search in large graph databases. In *SIGMOD*, 2013.

[15] W.-S. Han, J. Lee, M.-D. Pham, and J. X. Yu. igraph: a framework for comparisons of disk-based graph indexing techniques. In *PVLDB*, 2010.

[16] H. He and A. K. Singh. Closure-tree: An index structure for graph queries. In *ICDE*, 2006.

[17] X. He, J. Vaidya, B. Shafiq, N. Adam, and X. Lin. Reachability analysis in privacy-preserving perturbed graphs. In *WI-IAT*, 2010.

[18] B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In *VLDB*, 2004.

[19] H. Hu, J. Xu, Q. Chen, and Z. Yang. Authenticating location-based services without compromising location privacy. In *SIGMOD*, 2012.

[20] V. Karwa, S. Raskhodnikova, A. Smith, and G. Yaroslavtsev. Private analysis of graph structure. In *VLDB*, 2011.

[21] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2007.

[22] A. Kundu, M. J. Atallah, and E. Bertino. Efficient leakage-free authentication of trees, graphs and forests. *IACR Cryptology ePrint Archive*, 2012:36, 2012.

[23] J. Lee, W.-S. Han, R. Kasperovics, and J.-H. Lee. An in-depth comparison of subgraph isomorphism algorithms in graph databases. In *PVLDB*, 2013.

[24] K. Liu and E. Terzi. Towards identity anonymization on graphs. In *SIGMOD*, 2008.

[25] D. A. Menascé. Qos issues in web services. *IEEE Internet Computing*, 2002.

[26] K. Mouratidis and M. L. Yiu. Shortest path computation with no information leakage. 2012.

[27] NCBI. PubChem. *http://pubchem.ncbi.nlm.nih.gov/*, 2014.

[28] NIC. AIDS. *http://dtp.nci.nih.gov/docs/aids/aids_data.html*, 2004.

[29] H. Shang, Y. Zhang, X. Lin, and J. X. Yu. Taming verification hardness: an efficient algo. for testing subgraph isomorphism. In *PVLDB*, 2008.

[30] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23:31–42, 1976.

[31] H. Wang, J. Li, J. Luo, and H. Gao. Hash-base subgraph query processing method for graph-structured xml documents. In *PVLDB*, 2008.

[32] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis. Secure knn computation on encrypted databases. In *SIGMOD*, 2009.

[33] X. Yan, P. S. Yu, and J. Han. Graph indexing: a frequent structure-based approach. In *SIGMOD*, 2004.

[34] P. Yi, Z. Fan, and S. Yin. Privacy-preserving reachability query services for sparse graph. *ICDE GDM Workshop*, 2014.

[35] S. Yin, Z. Fan, P. Yi, B. Choi, J. Xu, and S. Zhou. Privacy-preserving reachability query services. In *DASFAA*, 2014.

[36] D. Yuan and P. Mitra. Lindex: a lattice-based index for graph databases. *The VLDB Journal*, 22:229–252, 2012.

[37] B. Zhou and J. Pei. Preserving privacy in social networks against neighborhood attacks. In *ICDE*, 2008.

[38] L. Zou, L. Chen, and M. T. Özsu. k-automorphism: a general framework for privacy preserving network publication. In *VLDB*, 2009.

[39] L. Zou, L. Chen, J. X. Yu, and Y. Lu. A novel spectral coding in a large graph database. In *EDBT*, 2008.

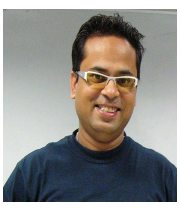
Zhe Fan is a PhD student in the Department of Computer Science, Hong Kong Baptist University. He received his BEng degree in Computer Science from Sourth China University of Technology in 2011. His research interests include graph-structured databases. He is a member of the Database Group at Hong Kong Baptist University. (http://www.comp.hkbu.edu.hk/∼db/).

Byron Choi is an associate professor in the Department of Computer Science at the Hong Kong Baptist University. He received the bachelor of engineering degree in computer engineering from the Hong Kong University of Science and Technology (HKUST) in 1999 and the MSE and PhD degrees in computer and information science from the University of Pennsylvania in 2002 and 2006, respectively.

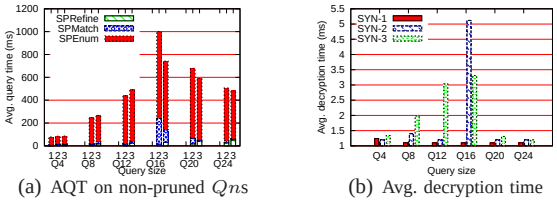
Qian Chen is a PhD student in the Department of Computer Science, Hong Kong Baptist University. He received his BEng degree in Computer Science from East China Normal University in 2011. His research interests include privacy-aware computing. He is a member of the Database Group at Hong Kong Baptist University. (http://www.comp.hkbu.edu.hk/∼db/).

Jianliang Xu is a professor in the Department of Computer Science, Hong Kong Baptist University. He received his BEng degree in computer science and engineering from Zhejiang University, Hangzhou, China, in 1998 and his PhD degree in computer science from Hong Kong University of Science and Technology in 2002. He held visiting positions at Pennsylvania State University and Fudan University.

Haibo Hu is a research assistant professor in the Department of Computer Science, Hong Kong Baptist University. Prior to this, he held several research and teaching posts at HKUST and HKBU. He received his PhD degree in Computer Science from the HKUST in 2005. His research interests include mobile and wireless data management, location-based services, and privacy-aware computing.



Sourav S Bhowmick is an Associate Professor in the School of Computer Engineering, Nanyang Technological University. He is a Visiting Associate Professor at the Biological Engineering Division, Massachusetts Institute of Technology. He held the position of Singapore-MIT Alliance Fellow in Computation and Systems Biology program (05'-12'). He received his Ph.D. in computer engineering in 2001.



(a) AQT on non-pruned Qns (b) Avg. decryption time

Fig. 11. Performance on synthetic dataset.

APPENDIX A
EXPERIMENTS ON SYNTHETIC DATASETS

In this appendix, we highlight the performance of SPsubIso on synthetic datasets. For the synthetic datasets, we selected three synthetic datasets (denoted as SYN-1, SYN-2 and SYN-3) from [15]. We note that the number of distinct vertex labels significantly affects the performance. We varied the number of distinct vertex labels of these three datasets to 20, 50, and 80, respectively. The average size and density of each graph are 30 and 0.5 respectively.

We note that their experimental results are similar to those of real datasets (presented in Sec 7). Hence, we only report two major performance results, shown in Fig. 11. Fig. 11 (a) reports the average query times on non-pruned queries. All of them are no larger than 1s. Fig. 11 (b) shows the decryption time at the client side. The decryption time on $Q16$ under SYN-2 is the largest, which is only 5ms. In general, the average time spent at the client side is very small.

APPENDIX B
THE PROOF OF PROPOSITION 6.3

Proposition 6.3 Under SPRefine, the following holds:

- If $\mathbf{M}(j,k)$ is not flipped, there is no information leakage; and
- Otherwise,

$$\begin{array}{rcl} \Pr[\mathcal{A}(Q)=1] & = & \frac{P(a+1)}{P(m)(P(a+1)-1)}, \text{and} \\ \Pr[\mathcal{A}(G)=1] & = & \frac{P(b+1)}{P(n)(P(b+1)-1)}, \end{array} \quad (9)$$

where $a = |\mathsf{MaxDeg}(Q)|^{\mathsf{maxH}}$, $b = |\mathsf{MaxDeg}(G)|^{\mathsf{maxH}}$, and $\mathsf{MaxDeg}(G)$ is the maximum degree of the vertices of G. □

Proof: Recall that for any $v_j \in V(Q), v_k \in V(G)$, $\mathsf{SI}_{\mathsf{Q_k}}[v_j]$ or $\mathsf{SI}_{\mathsf{G_k}}[v_k]$ themselves do not leak any structural information against CPA by Lemma. 6.5. Therefore, we only consider the private inner product between $\mathsf{SI}_{\mathsf{Q_k}}[v_j]$ and $\mathsf{SI}_{\mathsf{G_k}}[v_k]$. For each $\mathbf{M}(j,k) = 1$, we divide it into two exhaustive cases as follows:

Case 1: If $\mathbf{M}(j,k)$ is *not* flipped, $\mathsf{SI}_{\mathsf{Q}}[v_j] \cdot \mathsf{SI}_{\mathsf{G}}[v_k] = \mathsf{SI}_{\mathsf{Q}}[v_j] \cdot \mathsf{SI}_{\mathsf{Q}}[v_j]$ by Prop. 5.4. By Lemma 6.5, $\mathcal{SP}$ cannot learn any structural information from $\mathsf{SI}_{\mathsf{Q_k}}[v_j]$ and $\mathsf{SI}_{\mathsf{G_k}}[v_k]$. The only information the $\mathcal{SP}$ can deduce is that the (four) conditions listed in Prop. 5.3 hold. Since all the values of MaxDeg, Occur, PreLabel and Sup are encrypted, the $\mathcal{SP}$ does not learn any structural information (*i.e.*, Q and G) of v_j and v_k. Hence, there is no information leakage; and

Case 2: If $\mathbf{M}(j,k)$ is flipped, $\mathsf{SI}_{\mathsf{Q}}[v_j] \cdot \mathsf{SI}_{\mathsf{G}}[v_k] \neq \mathsf{SI}_{\mathsf{Q}}[v_j] \cdot \mathsf{SI}_{\mathsf{Q}}[v_j]$. Similar to *Case* 1, the $\mathcal{SP}$ cannot deduce structural information from this, due to the encrypted operations. However, the flip of $\mathbf{M}(j,k)$ implies that there is a violation caused by v_j and v_k between the subgraphs Q_a and G_b, where Q_a (resp., G_b) is the induced subgraph of Q (resp., G), containing at most a (resp., b) vertices that are reachable from v_j (resp., v_k) within maxH hops. This affects the probabilities similar to that in the proof of Prop. 6.2 as follows:

- Vertices in $V(Q_a)$ are all *isolated*. The number of the possible Q containing such a Q_a is $2^{m^2-(a+1)^2} = P(m)/P(a+1)$; and
- Vertices in $V(G_b)$ are connected to all other vertices. The number of the possible G containing such G_b is $2^{n^2-(b+1)^2} = P(n)/P(b+1)$.

We obtain the probabilities as follows (similar to the derivations of Prop. 6.2's proof): $\Pr[\mathcal{A}(Q) = 1] = \frac{1}{P(m)-P(m)/P(a+1)} = \frac{P(a+1)}{P(m)(P(a+1)-1)}$, and $\Pr[\mathcal{A}(G) = 1] = \frac{1}{P(n)-P(n)/P(b+1)} = \frac{P(b+1)}{P(n)(P(b+1)-1)}$, respectively.

Finally, each flip is independent because the subgraph of Q_a and G_b of each SPRefine can be arbitrarily different. □