Privacy-Preserving Reachability Query Services

Shuxiang Yin[1], Zhe Fan[2], Peipei Yi[2], Byron Choi[2], Jianliang Xu[2], Shuigeng Zhou[1]

[1]Shanghai Key Lab of Intelligent Information Processing, Fudan University, China
{sxyin,sgzhou}@fudan.edu.cn
[2]Computer Science Department, Hong Kong Baptist University, Hong Kong, China
{zfan, csppyi, choi, xujl}@comp.hkbu.edu.hk

Abstract. Due to the massive volume of graph data from a wide range of recent applications and resources required to process numerous queries at large scale, it is becoming economically appealing to outsource graph data to a third-party service provider ($\mathcal{SP}$), to provide query services. However, $\mathcal{SP}$ cannot always be trusted. Hence, data owners and query clients may prefer not to expose their data graphs and queries. This paper studies privacy-preserving query services for a fundamental query for graphs namely the reachability query where *both* clients' queries and the structural information of the owner's data are protected. We propose *privacy-preserving 2-hop labeling* (`pp-2-hop`) where the queries are computed in an encrypted domain and the input and output sizes of any queries are indistinguishable. We analyze the security of `pp-2-hop` with respect to ciphertext only and size based attacks. We verify the performance of `pp-2-hop` with an experimental study on both synthetic and real-world datasets.

1 Introduction

There is a wide range of emerging applications of graph-structured data, *e.g.*, bioinformatics, communication networks, social networks, web topology and semi-structured data. Many graph queries have been proposed to retrieve such graph data. However, as the volume of graph data is increasing at an unprecedented rate, hosting efficient query services has become a technically challenging task. The *owners* of graph data may not always be equipped with the expertise required to provide such services and therefore may employ *query service providers* ($\mathcal{SP}$s) to host *query services*, which are often supported by high performance computing. Security (such as the confidentiality of messages exchanged) has been stated as one of the attributes of Quality of Services (QoS) [23], as $\mathcal{SP}$ s cannot always be trusted. This attribute may influence the willingness of both data owners and query clients to use $\mathcal{SP}$'s services.

Take the reachability query — one of the most *fundamental and popular* graph queries [3, 4, 7, 8, 10, 16–18, 25–29, 31] — as an example: *Given two nodes u and v of a graph, the reachability query is used to test if v is reachable from u or not*. A query client may prefer not to expose his/her reachability query to an $\mathcal{SP}$. On the other hand, the data owner may prefer that the $\mathcal{SP}$ not be able to infer the structure of their graph data. Therefore, the query results must be protected as malicious $\mathcal{SP}$s can exploit the results from multiple queries to infer the graphs' structures.

Motivating example. Consider a pharmaceutical company whose revenue depends mostly on the invention of Health Care Products, illustrated in Fig. 1. The company may have discovered new compounds for a new product. To save laboratory experiments, it

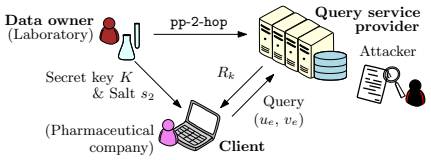
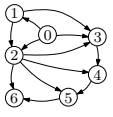


Fig. 1. Overview of the system model



v	Lin(v)	Lout(v)
0	0	0, 1, 2, 3, 5, 6
1	1	1, 2, 3, 5, 6
2	2	2, 3, 5, 6
3	3	3, 4
4	0, 1, 2, 4	4
5	4, 5	5, 6
6	4, 6	6

Fig. 2. A (partial) schematic of a biological network (LHS) and its 2-hop labeling (RHS)

may often query the compounds from web-accessible biological pathway networks of massive size (such as [2]) to understand the compounds' characteristics, such as whether it is possible for the compounds to form other compounds via any chemical reactions (reachability in the network). However, on the one hand, the company does not want the $\mathcal{SP}$ to know about the queries (the compounds), as it may apply for patents for the synthesis. On the other hand, the owner of the pathway networks may not only lack the experience to host query services (which is best supported by IT companies, *e.g.*, [15]) but also be reluctant to release the networks to the public. Alternatively, the owner is willing to release a license only to paid users. Therefore, it is crucial to protect *both* the queries and the network from the $\mathcal{SP}$.

This paper studies the problem of *evaluating the reachability query at an $\mathcal{SP}$ without compromising the privacy of the reachability of the query nodes and the graph structure under cipertext-only and size-based attacks, in the paradigm of query services* (to be detailed with Fig. 1 in Sec. 3). To our knowledge, this has not been addressed before.

There have been many recent studies on efficient reachability query (*e.g.*, [3, 4, 6–8, 17, 18, 25–28, 31]). Jin et al. [16] show that these studies can be roughly categorized into *transitive closure compressions*, *refined online search* and *hop labeling*. The first two categories suffer from high storage costs and require online searches, respectively (to be discussed in Sec. 2). In this paper, we propose our techniques based on hop labeling [10], in particular 2-hop labeling. The benefits of adopting 2-hop labeling are threefold. Firstly, the structures of 2-hop labels are simple, where each node is associated with two sets of nodes called Lin and Lout. Secondly, the query evaluation with 2-hop labeling is an intersection between an Lin and an Lout. Such simple structure and algorithm make privacy preservation plausible. Thirdly, 2-hop labeling is an active research topic. The recent works on large graph partitioning (*e.g.*, [8]), compression (*e.g.*, [7]) and maintenance (*e.g.*, [3, 25]) of 2-hop labeling can be readily adopted.

This paper proposes pp-2-hop (*privacy-preserving* 2-hop), which adopts 2-hop labeling. Firstly, the evaluation of each query on 2-hop labeling only involves an intersection between two sets of center nodes. Hence, we minimize the size of the maximum cardinality of the intersection results and add *minimal* artificial nodes (called surrogate nodes) to Lins and Louts such that the intersection results for all possible queries are of the *same* size. We unify each of the Lin and Lout labels such that the difference of the label set sizes are within a user-defined parameter. Secondly, we encrypt the 2-hop labels, after adding surrogate nodes and evaluate queries in the encrypted domain. We analyze the privacy of pp-2-hop.

The contributions of this paper are summarized as follows.

- We propose algorithms to unify the sizes of 2-hop labels and the query result sizes;
- We propose private query processing over the encrypted 2-hop labels;

– We propose a new heuristic 2-hop construction that yields 2-hop labeling that minimizes the intermediate results in private query processing; and
– We conduct an empirical study to confirm that our techniques are efficient.

The rest of this paper is organized as follows. We introduce some related work in Sec. 2. We then present the background, the problem and the overview of our solution in Sec. 3. We propose pp-2-hop, the index construction, optimization and query processing in Sec. 4. We conduct a privacy analysis in Sec. 5. We present an experimental evaluation in Sec. 6. We end this paper with the conclusion in Sec. 7.

2 Related Work

In this section, we present some related works on security of graph queries.

Authentication of graph queries. Query authentication is a security problem where the $\mathcal{SP}$ cannot be trusted. It requires the client to verify the correctness of the data graphs returned. Kundu et al. [21, 22] focus on the verification of the *authenticity* of a *given portion of data* (subtree/subgraph that users' have the right to access to) without leakage of extraneous information of the data (tree/graph/forest). They optimize the signature needed [22]. Zhe et al. [12] propose an efficient authenticated subgraph query services framework under the outsourced graph databases system. In comparison, the input of our problem is a reachability query not a subgraph for verification. Our problem focuses on *protecting both the query and its answer from $\mathcal{SP}$s*.

Privacy-preserving graph query. He et al. [14] analyze the node reachability of the graph data, with the preservation of edge privacy. Unfortunately, the method reveals the reachability of the query nodes and partial structure of the graph data to the $\mathcal{SP}$. Gao et al. [13] propose neighborhood-privacy protected shortest distance in the paradigm of cloud computing. This aims to preserve all the neighborhood connections and the shortest distances between each pair of nodes in outsourced graph data. When this work is directly applied to reachability queries, some information about the graph structure and the reachability of the query nodes are still exposed to the $\mathcal{SP}$.

Mouratidis et al. [24] determine the shortest path of the query nodes with no information leakage by using the PIR [9] protocol. Firstly, the high computational cost of PIR has been well known. Secondly, the PIR approach requires the transfer of the same amount of data for every query, which can be large. Reachability queries are simple (yet fundamental) queries and do not require the use of the costly PIR method. Cao et al. [5] propose to support subgraph query over an encrypted database of small graphs. Their work protects the query privacy, index privacy and feature privacy. However, reachability queries cannot be expressed as subgraph queries. Karwa et al. [19] present an efficient algorithm for outsourcing useful statistics of graph data by protecting the edge *differential privacy* and they support counting queries but not reachability queries.

Approaches for reachability query. Numerous approaches for reachability query have been proposed recently in the literature. Jin et al. [16] recently discuss the approaches based on three main categories: *transitive closure compressions*, *refined online search* and *hop labeling*. (i) Transitive closure compressions (*e.g.*, [1, 17, 28]) offer the best query performance. It is known that their storage costs are the highest. (ii) Refined online search (*e.g.*, [26,27,31]) relies on online searches (*e.g.*, DFS and BFS) by definition leaks graph structure information and it is not clear how they can be adopted here. (iii)

The seminal 2-hop labeling is proposed by Cohen et al. [10] and further optimized by many studies [7, 8, 25]. Due to the simple structure of the 2-hop labeling approach and the simple query evaluation (as motivated in Sec. 1), in this paper, we propose our techniques based on the 2-hop labeling approach.

3 Problem Formulation and Overview

This section formulates the problem studied and gives an overview of our solution.

Data model. We consider *directed node-labeled graphs*. A graph is denoted as G, and $V(G)$ and $E(G)$ are the node and edge sets of G, respectively. Since the reachability information of nodes in a strongly connected component is identical, we assume directed acyclic graphs (DAG), for presentation brevity. A reachability query takes two nodes u and v as input, denoted as $u \rightsquigarrow v$, and returns true *iff* v is reachable from u.

System model. We follow the system model that is commonly used in the literature of database outsourcing, presented in Fig. 1. The model consists of three parties:

- *Data owner*: An owner owns the graph data and computes the privacy-preserving 2-hop labeling offline *once*. It outsources them to the service provider and delivers query clients a salt s_2 to encrypt queries and the secret key K to decrypt results;
- *Service provider* ($\mathcal{SP}$): The $\mathcal{SP}$ has high computational utility such as cloud computing. The $\mathcal{SP}$ handles massive query requests over the encrypted data on behalf of the data owner and returns the encrypted results to clients; and
- *Client*: A client encrypts a query using s_2, sends the encrypted query to the $\mathcal{SP}$ and decrypts the result with K. We assume that the clients and $\mathcal{SP}$ do not collude.

Privacy target. Our privacy target is required to keep the following two pieces of information private from *attackers* using the two attacks defined in the attack model:

- *Reachability of the query nodes* a.k.a the query result. In particular, given a reachability query $u \rightsquigarrow v$, attackers cannot infer whether u can reach to v; and
- *Graph structure* a.k.a. the topology of the data graph, *e.g.*, the existence of an edge.

Attack model. As other outsourcing work, we assume the $\mathcal{SP}$s are *honest-but-curious* [20]. The attackers may be the $\mathcal{SP}$ or another adversary hacking the $\mathcal{SP}$. For simplicity, we *term the attackers as the* $\mathcal{SP}$. The $\mathcal{SP}$ can adopt the following popular attacks.[1]

- *Ciphertext only attack*, where the $\mathcal{SP}$ can access only the ciphertext (encrypted graph data) and does not know what their original graph is; and
- *Size based attack*, where the $\mathcal{SP}$ attempts to infer the two pieces of private information from the sizes of the data and the query results.

2-hop labeling. As motivated in Sec 1, we undertake the 2-hop approach to address the problem. In 2-hop labeling, each node $u \in V(G)$ is associated with two sets of nodes, denoted as $\texttt{Lout}(u)$ and $\texttt{Lin}(u)$, called 2-hop *labels*. Nodes in $\texttt{Lout}(u)$ (respectively, $\texttt{Lin}(u)$) can be reachable from u (respectively, can reach u), which are also called *center nodes*. Given two nodes u and v, $u \rightsquigarrow v$ *iff* $\texttt{Lout}(u) \cap \texttt{Lin}(v) \neq \emptyset$.

[1] There are admittedly many other attacks in the literature. We do not claim them in this paper since their theoretical privacy guarantees have yet to be established.

Example 1. Consider the simplified biological pathway network shown in the LHS of Fig. 2. The node ID represents the chemical ID and the edge denotes a chemical reaction. The nodes' labels of the graph data are omitted for simplicity of presentation. The original 2-hop labeling (RHS) of the graph is shown in the RHS of Fig. 2. Consider two nodes 1 and 5. Node 1 can reach Node 5, *i.e.*, $1 \rightsquigarrow 5$. $\texttt{Lout}(1) \cap \texttt{Lin}(5) = \{5\}$. However, Node 0 is not reachable from Node 6 as $\texttt{Lout}(6) \cap \texttt{Lin}(0) = \emptyset$.

2-hop construction. To ensure that the 2-hop labels of $V(G)$ contain all the reachability (a.k.a connectivity) information of G, the labels must cover all the elements in the transitive closure $T(G)$ of G. 2-hop labels (covers) are known to be costly to construct. In practice, they are constructed offline. In addition, there are various works that significantly optimize the construction time (*e.g.*, [8]).

The majority of previous 2-hop constructions focus on minimizing the size of 2-hop labeling, defined as $\sum_{u \in V(G)}(|\texttt{Lout}(u)| + |\texttt{Lin}(u)|)$. As we shall present our heuristic in Sec 4.1, we briefly outline the heuristic construction approach [10] for 2-hop construction. Initially, a variable T' is defined to represent the uncovered elements of $T(G)$, *i.e.*, $T' = T(G)$. Elements of $T(G)$ are iteratively covered and removed from T'. For each node $w \in G$, an undirected bipartite graph (*a.k.a* center graph) $G_w(L_w, R_w, E_w)$ is built, where L_w are nodes that can reach w and R_w are those w can reach. $(u, v) \in E_w$ *iff* (u, v) is in T'. The heuristic algorithm *selects* the center w whose induced subgraph $G_i(L_i, R_i, E_i)$ of G_w has the largest ratio maxDensCover defined below.

$$\texttt{maxDensCover} = \frac{|E_i \cap T'|}{|L_i \cup R_i|} \tag{1}$$

In each iteration, the node w with the largest maxDensCover is selected as a center. For all $u \in L_i$ and $v \in R_i$, the algorithm adds w into $\texttt{Lout}(u)$ and $\texttt{Lin}(v)$, and removes (u, w), (w, v) and (u, v) from T'. The iterations terminate when T' is empty.

4 Privacy-preserving 2-hop Labeling

This section presents the details of privacy-preserving 2-hop labeling. Sec. 4.1 presents a heuristic, to replace maxDensCover, that minimizes *the maximum cardinalities of the intersection results* $I_{\texttt{max}}$ of 2-hop labels. In Sec 4.2, we propose a greedy algorithm that introduces a minimal number of surrogate nodes to Lins and Louts of the 2-hop labels derived from Sec. 4.1. In Sec. 4.3, we encrypt the 2-hop labels. In Sec. 4.4, we present their private query processing.

4.1 Imax-aware 2hop construction

In practice, $I_{\texttt{max}}$ constructed by using maxDensCover (outlined in Sec. 3) are very often large. In our experiments, the average $I_{\texttt{max}}$ of the 2-hop labels using maxDensCover of our real datasets is 378 (Table 7). Furthermore, large $I_{\texttt{max}}$s lead to large surrogate labels and thus high query costs.

In this section, we propose a heuristic to minimize $I_{\texttt{max}}$ as 2-hop labels are constructed. The idea is that the heuristic includes the intersection information in 2-hop construction. Specifically, the objectives are to minimize the following two quantities:

1. the center node w that covers the most uncovered elements in T': As in previous work, such a heuristic leads to few center nodes and small 2-hop labels; and

2. the maximum cardinality of the intersection between $\texttt{Lout}(u)$ and $\texttt{Lin}(v)$, *i.e.*, $max(|\texttt{Lout}(u) \cap \texttt{Lin}(v)|)$.

To achieve the objectives, we propose the heuristic ratio `maxISCover` as follows:

$$\texttt{maxISCover} = \frac{|E_w \cap T'|}{max(|\texttt{Lout}(u) \cap \texttt{Lin}(v)|)}, \tag{2}$$

where $u \in L_w$, $v \in R_w$, T' is the uncovered elements in $T(G)$. Eqn. 2 contains two main parts. The details are as follows:

1. $E_w \cap T'$ is the uncovered elements in $T(G)$ covered by selecting w; and
2. $max(|\texttt{Lout}(u) \cap \texttt{Lin}(v)|)$ is the largest $|\texttt{Lout}(u) \cap \texttt{Lin}(v)|$ of all $u, v \in V(G)$.

Putting these together, we use the 2-hop construction presented at the end of Sec. 3, replacing `maxDensCover` with `maxISCover`.

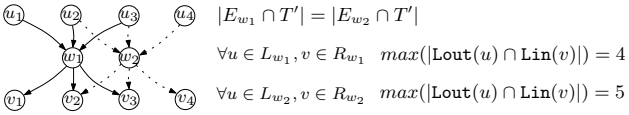


Fig. 3. Illustration of selection of center node by `maxISCover`

Example 2. We compare `maxDensCover` and `maxISCover` with a schematic shown in Fig. 3. The graph data is on the LHS. Suppose there are only two center nodes (w_1 and w_2) checked in an iteration respectively. $L_{w_1} = \{u_1, u_2, u_3\}$, $R_{w_1} = \{v_1, v_2, v_3\}$, and $L_{w_2} = \{u_2, u_3, u_4\}$, $R_{w_2} = \{v_2, v_3, v_4\}$. We assume $|E_{w_1} \cap T'| = |E_{w_2} \cap T'|$. `maxDensCover` may select either w_1 or w_2. Further assume that $\forall u \in L_{w_1}, v \in R_{w_1}$, $max(|\texttt{Lout}(u) \cap \texttt{Lin}(v)|) = 4$ and $\forall u \in L_{w_2}, v \in R_{w_2}$, $max(|\texttt{Lout}(u) \cap \texttt{Lin}(v)|) = 5$. `maxISCover` selects w_1 as it results in smaller intersections.

4.2 Addition of Surrogate Nodes

Next, we add surrogate nodes to 2-hop labeling to unify the sizes of intersection results and the `Lins` and `Louts`.

Unification of Intersection Results. The main idea of adding surrogate nodes into 2-hop labels (`Lins` and `Louts`) is to achieve that for each u and v, the intersection between surrogate labels $\texttt{Lout}^{\texttt{s}}(u)$ and $\texttt{Lin}^{\texttt{s}}(v)$ always equals to $I_{\texttt{max}}$.[2] Our objective is thus to introduce the smallest number of surrogate nodes (*i.e.*, the 2-hop label size is minimized). As the cardinality for each intersection is the same, the $\mathcal{SP}$ gains no connectivity knowledge about specific information of each pair of nodes.[3] We call the problem the *minimum addition of surrogate nodes* (`MASN`), defined below.

Definition 1. *The problem of minimum addition of surrogate nodes (*`MASN`*) is that given the* 2-hop *labels of a graph G, introduce surrogate nodes in both* `Lin`*s and* `Lout`*s to obtain* $\texttt{Lin}^{\texttt{s}}$*s and* $\texttt{Lout}^{\texttt{s}}$*s, such that*

- $\forall u, v \in V(G)$, $|\texttt{Lout}^{\texttt{s}}(u) \cap \texttt{Lin}^{\texttt{s}}(v)| = I_{\texttt{max}}$*;*
- *the largest* $\texttt{Lin}^{\texttt{s}}$ *and the largest* $\texttt{Lout}^{\texttt{s}}$ *are minimized; and*
- $\sum_{u \in V(G)}(|\texttt{Lin}^{\texttt{s}}(u)| + |\texttt{Lout}^{\texttt{s}}(u)|)$ *is minimized.*

Algorithm 1 Unify Intersection Sizes `unifyIS`(`Lout`, `Lin`)

Input: 2-hop labeling `Lout` and `Lin`
Output: $\texttt{Lout}^{\texttt{s}}$ and $\texttt{Lin}^{\texttt{s}}$

1: Initialize surrogate node set $D_w = [\,]$
2: create a priority queue Q_v for $V(G)$, where the rank of $v \in V(G)$ is defined by:
   $max_{u \in V}(I_{\texttt{max}}$ - $|\texttt{Lout}(u) \cap \texttt{Lin}(v)|)$
3: **while** $v \neq$ `null`, where $v \leftarrow Q_v$.`getNext()` //scan `Lins`
4: D_w.`movetofront()` //move to the front of D_w
5: **while** $d_i \leftarrow D_w$.`getNext()`
6: **if** d_i = `null` and $\exists u', v'\ |\texttt{Lout}(u') \cap \texttt{Lin}(v')| < I_{\texttt{max}}$
   $d_i \leftarrow$ `new node()` and D_w.`push`(d_i)
   else `break`
7: **for each** $u \in V(G)$, where $|\texttt{Lout}(u) \cap \texttt{Lin}(v)| < I_{\texttt{max}}$ //scan `Louts`
   //if the *intersection constraint* is satisfied.
8: **if** ψ(2-hop, u, v, d_i) is true
9: $\texttt{Lout}(u) \leftarrow \texttt{Lout}(u) \cup \{d_i\}$; $\texttt{Lin}(v) \leftarrow \texttt{Lin}(v) \cup \{d_i\}$
10: **return** (`Lout`, `Lin`) **as** ($\texttt{Lout}^{\texttt{s}}$, $\texttt{Lin}^{\texttt{s}}$)

Proposition 1. *The problem of* `MASN` *is NP-hard.*

It is not surprising that the problem of `MASN` is NP-hard, as presented in Prop. 1. Due to space constraints, we provide its proof in the Appendix. We propose a greedy algorithm called `unifyIS` (shown in Algo. 1) to solve the problem of `MASN`.

The input of `unifyIS` is 2-hop labels (*i.e.*, `Louts` and `Lins`). The output is the 2-hop labels with surrogate nodes, *i.e.*, $\texttt{Lout}^{\texttt{s}}$ and $\texttt{Lin}^{\texttt{s}}$. The main idea is that for any surrogate node d, we add d to as many `Lins` and `Louts` as possible, in $O(|V|)$, and hence more entries in $T(G)$, in $O(|V|^2)$, can be covered. Specifically, we maintain a list of surrogate nodes D_w as we add them to the 2-hop labels (Line 1). `unifyIS` processes the `Lins` whose intersection with some other `Louts` has the largest gap between $I_{\texttt{max}}$ earlier (Lines 2-3).[4] As `unifyIS` may use new surrogate nodes to reduce such a gap, the surrogate nodes can be used to reduce the gaps between the `Lin` and many other `Louts`. In Lines 5-9, `unifyIS` processes the `Louts` iteratively. Line 6 checks if there is a `Lout` to be processed (*i.e.*, $|\texttt{Lout}(u) \cap \texttt{Lin}(v)| < I_{\texttt{max}}$). If so, and all existing surrogate nodes have been used in previous iterations, then a new surrogate node is created (Line 6). Otherwise, `unifyIS` terminates. The surrogate node is added to both $\texttt{Lout}(u)$ and $\texttt{Lin}(v)$ (Line 9), if the *intersection constraint* ψ (defined below) in Line 8 is satisfied:

$$\begin{aligned}\psi(\texttt{2-hop}, u, v, d_i) : \forall v', |(\texttt{Lout}(u) \cup \{d_i\}) \cap \texttt{Lin}(v')| \leq I_{\texttt{max}} \wedge \\ \forall u', |\texttt{Lout}(u') \cap (\texttt{Lin}(v) \cup \{d_i\})| \leq I_{\texttt{max}},\end{aligned} \tag{3}$$

where $u', v' \in V(G)$. In other words, there is no $|\texttt{Lout}(u) \cap \texttt{Lin}(v)| > I_{\texttt{max}}$ after adding d_i to both $\texttt{Lout}(u)$ and $\texttt{Lin}(v)$. Thus, if ψ is true, we can add d_i into both $\texttt{Lout}(u)$ and $\texttt{Lin}(v)$. The algorithm terminates when for all pairs of nodes u and v, $|\texttt{Lout}(u) \cap \texttt{Lin}(v)| = I_{\texttt{max}}$.

[2] For brevity, we assume the data graph does not contain highly disconnected nodes where its $I_{\texttt{max}}$ equals 0 or 1. To cater for such special cases, we may further set a minimum $I_{\texttt{max}}$ (*e.g.*, 4) so that the $\mathcal{SP}$ cannot infer whether the graph is highly disconnected or not.

[3] In 2-hop labeling, the sizes of the intersection results of a query indicate the reachability of nodes. Only unreachable query nodes have a value of 0.

[4] Given `Lins` and `Louts`, the value of $I_{\texttt{max}}$ can be easily computed.

v	$\texttt{Lin}^s(v)$	$\texttt{Lout}^s(v)$
0	$\mathbf{0}, 7, 8, 9$	$\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{5}, \mathbf{6}, 7, 8$
1	$\mathbf{1}, 7, 8, 10$	$\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{5}, \mathbf{6}, 7, 8, 9$
2	$\mathbf{2}, 7, 8, 11$	$\mathbf{2}, \mathbf{3}, \mathbf{5}, \mathbf{6}, 7, 8, 9, 10$
3	$\mathbf{3}, 7, 8, 12$	$\mathbf{3}, \mathbf{4}, 7, 8, 9, 10, 11$
4	$\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{4}, 9, 10, 13$	$\mathbf{4}, 7, 8, 9, 10, 11, 12$
5	$\mathbf{4}, \mathbf{5}, 7, 8, 14$	$\mathbf{5}, \mathbf{6}, 7, 8, 9, 10, 11, 12, 13$
6	$\mathbf{4}, \mathbf{6}, 7, 8$	$\mathbf{6}, 7, 8, 9, 10, 11, 12, 13, 14$

Fig. 4. The 2-hop labels of the network in Figure 2 after the addition of surrogate nodes

v	$\texttt{Lin}^s(v)$	$\texttt{Lout}^s(v)$
⋮	⋮	⋮
4	$\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{4}, 9, 10, 13$	$\mathbf{4}, 7, 7_1, 7_3, , 8, 9, 10, 11, 12$
5	$\mathbf{4}, \mathbf{5}, 7_1, 7_2, 8, 14$	$\mathbf{5}, \mathbf{6}, 7, 7_2, 8, 9, 10, 11, 12, 13$
6	$\mathbf{4}, \mathbf{6}, 7_2, 7_3, 8$	$\mathbf{6}, 7, 7_2, 8, 9, 10, 11, 12, 13, 14$

Encryption of $\texttt{Lin}^s(6)$:
$\texttt{Lin}^e(h_{s_2}(6)) = \{(h_{s_1}(4), E(0)), \cdots, (h_{s_1}(8), E(1))\}$

Fig. 5. One iteration of split on 2-hop labels shown in Figure 4 and the encrypted $\texttt{Lin}^s(6)$

Discussion. The number of distinct surrogate nodes used by unifyIS is $I_{\max}|V|$ in the worst case and $I_{\max}$ in the best case. Therefore, the total number of surrogate nodes added by unifyIS is $I_{\max}|V|(1 + |V|)$ in the worst case and $2I_{\max}|V|$ in the best case, where $|V|$ is the number of nodes in the graph.

Example 3. Figure 4 shows the 2-hop labeling after the addition of surrogate nodes to that in Figure 2. The IDs of the real centers are highlighted in bold in $\texttt{Lin}^s$s and $\texttt{Lout}^s$s. Other IDs represent surrogate nodes. The $I_{\max}$ of the 2-hop labels is 3.

Consider the first iteration of unifyIS. Since $I_{\max}$ - |Lin(0)| ∩ |Lout(1)| = $I_{\max}$, Node 0 is one of the nodes with the highest rank in Q_u (Lines 2 and 3). There are obviously some intersection results of Lin and Lout smaller than $I_{\max}$. Moreover, d_i is null as D_w is empty. A new surrogate (Node 7) is created (Line 6). Node 7 is introduced to both Lin(0) and Lout(1) as this does not violate ψ (Lines 8-9). Next, Node 7 is added to Louts of Nodes 0 and 2-6 as the addition does not violate ψ (Lines 7-9). In the next few iterations (Lines 5-9), unifyIS adds new surrogate nodes 8 and 9 to Lin(0) and the corresponding Louts until all intersection results of Lin(0) and other Louts are of the size $I_{\max}$.

Then, unifyIS processes the next node in Q_u (Line 3), for example, Node 1. unifyIS uses Node 7 (Lines 4-5) and adds it to Lin(1) as it does not violate ψ (Lines 7-9). Similarly, Node 8 is added to Lin(1). However, Node 9 cannot be added to Lin(1) as |Lin(1) ∩ Lout(1)| = |{**1**, 7, 8}| = $I_{\max}$.

Consider the surrogate node 7 from Figure 4. It has been added 13 places. However, it covers 6 × 7 elements in $T(G)$. For any query u and v, the intersection size is the same. Following the same queries of Example. 1, $\texttt{Lout}^s(1) \cap \texttt{Lin}^s(5) = \{\mathbf{5}, 7, 8\}$ and $\texttt{Lout}^s(6) \cap \texttt{Lin}^s(0) = \{7, 8, 9\}$. The result size is always 3.

Unification of Labeling Sizes In order to unify the sizes of both $\texttt{Lin}^s$s and $\texttt{Lout}^s$s, we introduce a (postprocessing) unifyLin operation of surrogate nodes. Since the task of unifying $\texttt{Lin}^s$ and $\texttt{Lout}^s$ are symmetric, we only discuss $\texttt{Lin}^s$ for a concise presentation. We first denote the size of the largest $\texttt{Lin}^s$ (respectively, $\texttt{Lout}^s$) as $\texttt{Lin}_{\max}$ (respectively, $\texttt{Lout}_{\max}$). The intuitions of our approach can be described as follows: (1) for each node u in the graph where $|\texttt{Lin}^s(u)| < \texttt{Lin}_{\max}$, we split its surrogate nodes in $\texttt{Lin}^s(u)$ such that the size of $\texttt{Lin}^s(u)$ is closer to $\texttt{Lin}_{\max}$; and (2) we never increase the $I_{\max}$ during unification. We formally define the problem, namely *unification of the labeling size* (ULS) as follows.

Definition 2. *The problem of the unification of the labeling size (*ULS*) is that given* $\texttt{Lin}^s$*s and* $\texttt{Lout}^s$*s, unify both of their sizes, s.t.,* $\forall u, v$*,*

$$\frac{||\texttt{Lin}^s(u)| - \texttt{Lin}_{\max}|}{\texttt{Lin}_{\max}} \leq \delta \text{ and } \frac{||\texttt{Lout}^s(u)| - \texttt{Lout}_{\max}|}{\texttt{Lout}_{\max}} \leq \delta,$$

where δ is a user-specified parameter for the allowable difference in the label sizes.

Algorithm 2 Unify $\texttt{Lin}^s$ unifyLin($\texttt{Lout}^s$, $\texttt{Lin}^s$, δ)

Input: 2-hop labeling $\texttt{Lout}^s$ and $\texttt{Lin}^s$ after addition of surrogate nodes, and the user-specified allowable differences δ in the sizes of $\texttt{Lin}^s$ and $\texttt{Lout}^s$.

Output: $\texttt{Lout}^s$ and $\texttt{Lin}^s$

1: **for each** u, $|\texttt{Lin}^s(u)| + \delta \leq \texttt{Lin}_{\texttt{max}}$
2: choose a surrogate node w from $\texttt{Lin}^s(u)$
3: $W = \{w_1, \cdots, w_n, w_{n+1}\}$, $n \leq \texttt{Lin}_{\texttt{max}} - |\texttt{Lin}^s(u)| - 1$
4: $\texttt{Lin}^s(u) \leftarrow (\texttt{Lin}^s(u)/\{w\}) \cup \{w_1, \cdots, w_{n+1}\}$
5: **for each** u', where $|\texttt{Lin}^s(u')| < \texttt{Lin}_{\texttt{max}} \wedge w \in \texttt{Lin}^s(u')$
6: $\texttt{Lin}^s(u') \leftarrow (\texttt{Lin}^s(u')/\{w\}) \cup \{w_{n+1}, w_{n+2}\}$
7: **for each** v, where $|\texttt{Lout}^s(v)| = \texttt{Lout}_{\texttt{max}} \wedge w \in \texttt{Lout}^s(v)$
8: $\texttt{Lout}^s(v) \leftarrow \texttt{Lout}^s(v) \cup \{w_{n+1}\}$
9: **for each** v, where $|\texttt{Lout}^s(v)| < \texttt{Lout}_{\texttt{max}} \wedge w \in \texttt{Lout}^s(v)$
10: $\texttt{Lout}^s(v) \leftarrow \texttt{Lout}^s(v) \cup \{w_i, w_{n+2}\}$,
where $i \in$ [1,n] and each i is added to some $\texttt{Lout}^s$ at least once.

We propose unifyLin algorithm presented in Algo. 2 to solve the problem of ULS. Suppose that we have sorted the $\texttt{Lin}^s$s by their sizes in descending order and we apply unifyLin to each $\texttt{Lin}^s$ accordingly. We first choose a surrogate node w from $\texttt{Lin}^s(u)$ (Lines 1-2). We declare new surrogate nodes $\{w_1, \cdots, w_n, w_{n+1}\}$, where $n \leq \texttt{Lin}_{\texttt{max}}$ - $|\texttt{Lin}^s(u)|$ - 1 (Line 3). We replace w by $\{w_1, \cdots, w_{n+1}\}$ in $\texttt{Lin}^s(u)$ (Line 4). For each u' where $w \in \texttt{Lin}^s(u')$ and $|\texttt{Lin}^s(u')| < \texttt{Lin}_{\texttt{max}}$, we replace w with w_{n+1} and w_{n+2} in $\texttt{Lin}^s(u')$ (Lines 5-6). For each v where $|\texttt{Lout}^s(v)| = \texttt{Lout}_{\texttt{max}}$ and $w \in \texttt{Lout}^s(v)$, we add w_{n+1} (Lines 7-8). For each v where $|\texttt{Lout}^s(v)| < \texttt{Lout}_{\texttt{max}}$ and $w \in \texttt{Lout}^s(v)$, we add $\{w_i, w_{n+2}\}$, where $i \in$ [1,n] (Lines 9-10).

Example 4. We illustrate Algo. 2 with reference to the 2-hop labels shown in Figure 4. The result after one unifyLin operation is shown in Figure 5. Suppose we choose Node 7 from $\texttt{Lin}^s(5)$ (Line 2). $\texttt{Lin}_{\texttt{max}} = 7$ and $|\texttt{Lin}^s(5)| = 5$. Therefore, n+1 can be 2. We replace Node 7 with $\{7_1, 7_2\}$ in $\texttt{Lin}^s(5)$ (Line 4). Since Node 7 appears in some other $\texttt{Lin}^s$, we add $\{7_2, 7_3\}$ to $\texttt{Lin}^s(i)$, $i \in \{0,1,2,3,6\}$ (Lines 5-6). $\texttt{Lout}_{\texttt{max}} = |\texttt{Lout}^s(5)| = |\texttt{Lout}^s(6)|$. We add Node 7_2 into $\texttt{Lout}^s(5)$ and $\texttt{Lout}^s(6)$ (Lines 7-8). We add Node 7_3 into $\texttt{Lout}^s(j)$ (Lines 9-10), where $j = \{0,1,2,3,4\}$.

One may verify that the sizes of intersection results do not change with a simple case analysis. In the meantime, all the surrogate nodes due to unifyLin appear in some intersection results. The increase in the sizes of $\texttt{Lin}^s$ and $\texttt{Lout}^s$ due to a unifyLin operation can be listed as follows: (i) the size of $\texttt{Lin}^s(u)$ is increased by n+1 (Line 4); (ii) the size of $\texttt{Lin}^s$ that contains w is increase by two (Lines 5-6); (iii) the sizes of $\texttt{Lout}^s$ with w whose sizes are $\texttt{Lout}_{\texttt{max}}$ are increased by one (Lines 7-8); and (iv) the $\texttt{Lout}^s$s that contains w whose sizes are smaller than $\texttt{Lout}_{\texttt{max}}$ increase by two (Lines 9-10). We then alternately apply unifyLin operations on $\texttt{Lin}^s$ and $\texttt{Lout}^s$ until the sizes do not differ from the largest labels by δ (as stated in Def. 2).

4.3 Index Encryption

After adding the surrogate nodes, the remaining task is to encrypt the labels . In order to distinguish the real nodes and surrogate nodes, we implement the nodes with flag values (see Def. 3). The flag values of real nodes are 0, and 1 otherwise. We will present how

to use the flags to encode the query result in Sec. 4.4. Below is the new definition of *center*s for $\texttt{Lout}^{\texttt{s}}$s or $\texttt{Lin}^{\texttt{s}}$s.

Definition 3. *Each* center *of* $\texttt{Lout}^{\texttt{s}}(u)$ *or* $\texttt{Lin}^{\texttt{s}}(v)$ *is a binary tuple* (w,f)*, where* $f = 0$ *if* w *is a real center, and* 1 *otherwise.*

Based on Def. 3, we encrypt the surrogate labels in order to protect both the reachability of the query nodes and the graph structure. (i) To hide any association between the nodes and the center nodes, we hash the w in (w, f) and the u in $\texttt{Lout}^{\texttt{s}}(u)$ and $\texttt{Lin}^{\texttt{s}}(u)$ with a *one-way collision-resistant hash function* with different salts, denoted as $h_{s_1}(w)$ and $h_{s_2}(u)$, to hash them respectively. Recall that that $w = u$ does not imply $h_{s_1}(w) = h_{s_2}(u)$, where $s_1 \neq s_2$. (ii) Regarding the encryption of the flag value, we use Elgamal $\texttt{E}(\cdot)$ [11], which is a *multiplicative homomorphic encryption* method. The benefits of Elgamal are twofold: (1) since the flag has binary values, Elgamal ensures randomness in the encrypted flags; (2) Elgamal allows one decryption at the client side. To sum up, the definition of the privacy-preserving 2-hop labeling is given as follows.

Definition 4. *Each* encrypted center *is a binary tuple* (w_e, f_e)*, where* $w_e = h_{s_1}(w)$ *and* $f_e = \texttt{E}(f)$*. The* privacy-preserving 2-hop *(*pp-2-hop*) is a* 2-hop *labeling where each encrypted node* u_e*, where* $u_e = h_{s_2}(u)$*, is associated with two sets of encrypted* $\texttt{Lout}^{\texttt{s}}(u)$ *and* $\texttt{Lin}^{\texttt{s}}(u)$*, denoted as* $\texttt{Lout}^{\texttt{e}}(u)$ *and* $\texttt{Lin}^{\texttt{e}}(u)$*.*

Example 5. Fig. 5 illustrates an example of the encryption of $\texttt{Lin}^{\texttt{s}}(6)$ for node 6. The encryption of $\texttt{Lin}^{\texttt{s}}(6)$ is denoted as $\texttt{Lin}^{\texttt{e}}(h_{s_2}(6))$, where $h_{s_2}(6)$ is the encryption of node 6. For example, the first center of $\texttt{Lin}^{\texttt{s}}(4)$, (4, 0), is encrypted as $(h_{s_1}(4), \texttt{E}(0))$.

4.4 Private Query Processing

Based on the encryption of the pp-2-hop labeling in Def. 4, we present its query processing without decryption. There are three main steps: (1) The client encrypts the query — the query $u \rightsquigarrow v$ is hashed to $u_e \rightsquigarrow v_e$; (2) The $\mathcal{SP}$ intersects $\texttt{Lout}^{\texttt{e}}(u_e)$ and $\texttt{Lin}^{\texttt{e}}(v_e)$ and returns the encrypted result R_e to the client; and (3) The client uses the secret key K and an Elgamal decryption to decrypt the result decryption.

Naïve solution. The naïve solution for processing a query $u \rightsquigarrow v$ is to perform an intersection on the centers in $\texttt{Lout}^{\texttt{e}}(u_e)$ and $\texttt{Lin}^{\texttt{e}}(v_e)$ and transmit the encrypted flag of the centers in the intersection results to clients. The client decrypts each of the encrypted flag and checks if there is at least one flag that signifies a real center. However, this solution requires $I_{\texttt{max}}$ decryptions.

Multiplicative homomorphic query processing. It is known that decryption is costly, especially when the client is not equipped with powerful hardware. Therefore, we propose a query processing that requires *one* decryption at the client side. We define the intersection result of u_e and v_e as $R(u_e,v_e)$, or simply R, where

$$R = \{(w_e, f_e) \mid (w_e, f_e) \in \texttt{Lout}^{\texttt{e}}(u_e) \text{ and } (w_e, f'_e) \in \texttt{Lin}^{\texttt{e}}(v_e)\}.$$

The encrypted result R_e, defined as $\prod_{(w_e,f_e)\in R} f_e$,[5] is transmitted to the client. At the client side, the client decrypts R_e by using the secret key K. If the decrypted message is 0, then u can reach v. Otherwise, u cannot reach v. Note that R_e is a product of flag values. The product is 0 *iff* there is a real node (whose flag is 0) in the intersection result. That is, if all centers in the results are surrogates, the product R_e is 1.

[5] We use $\prod$ and $\times$ to denote the modular multiplications in the Elgamal encryption scheme.

Example 6. Consider the private query processing of the query $1 \rightsquigarrow 5$, following Example 1 for clarity. The query processing on pp-2-hop in Example 3 is similar: (1) the client hashes the query nodes as $h_{s_2}(1)$ and $h_{s_2}(5)$ by using the salt s_2 from the data owner, and issues to the $\mathcal{SP}$; (2) the $\mathcal{SP}$ performs $\texttt{Lout}^{\texttt{e}}(h_{s_2}(1)) \cap \texttt{Lin}^{\texttt{e}}(h_{s_2}(5))$ and obtains $\{(h_{s_1}(5), \texttt{E}(0)), (h_{s_1}(7), \texttt{E}(1)), (h_{s_1}(8), \texttt{E}(1))\}$. Based on the result, the $\mathcal{SP}$ computes the result $R_e = \texttt{E}(0) \times \texttt{E}(1) \times \texttt{E}(1) = \texttt{E}(0)$ and returns it to the client; and (3) the client decrypts the R_e, which is 0, and obtains that Node 1 can reach Node 5.

5 Analysis of Privacy

In this section, we provide an analysis of the privacy under the assumptions of our attack model, *i.e.*, the size based attack and ciphertext only attack (stated in Sec. 3).

Privacy against Ciphertext Only Attack. We prove that the reachability of the query nodes and the topology of the graph have been protected from the $\mathcal{SP}$ under the *ciphertext only attack.*

Proposition 2. *The $\mathcal{SP}$ breaks the reachability of query nodes only if the $\mathcal{SP}$ breaks either the one-way collision-resistant hash function or the Elgamal encryption.*

Proof. (Sketch) **Case 1**: (i) Suppose the $\mathcal{SP}$ can break the Elgamal encryption. The $\mathcal{SP}$ can determine whether the flag of a center signifies a real center or not. During query processing, the $\mathcal{SP}$ can analyze the intersection result R. The $\mathcal{SP}$ identifies the reachability of a pair of query nodes by checking if there is a real center R.

(ii) If the $\mathcal{SP}$ can break the hash function (*e.g.*, SHA-1), it can determine the center identities, *i.e.*, the center IDs in $\texttt{Lin}^{\texttt{s}}$ or $\texttt{Lout}^{\texttt{s}}$. Then, the $\mathcal{SP}$ can check if a center is real by checking if it has corresponding $\texttt{Lin}^{\texttt{s}}$ and $\texttt{Lout}^{\texttt{s}}$ in pp-2-hop.

Case 2: Suppose the $\mathcal{SP}$ cannot break the one-way collision-resistant hash function (*e.g.*, SHA-1) and the Elgamal encryption. We analyze step by step the information the $\mathcal{SP}$ obtains during query processing. Given a query u_e and v_e, the $\mathcal{SP}$ retrieves $\texttt{Lout}^{\texttt{s}}(u_e)$ and $\texttt{Lin}^{\texttt{s}}(v)$. The $\mathcal{SP}$ computes R_e under the Elgamal encryption.

Since the $\mathcal{SP}$ cannot break the one-way collision-resistant hash function, it cannot determine either the nodes of the query (u_e and v_e) or the centers in $\texttt{Lout}^{\texttt{s}}(u_e)$ and $\texttt{Lin}^{\texttt{s}}(v_e)$. Moreover, since we assume that the $\mathcal{SP}$ cannot break the Elgamal encryption, it cannot determine the flags of the centers in $\texttt{Lout}^{\texttt{s}}(u_e)$ and $\texttt{Lin}^{\texttt{s}}(v_e)$. Due the homomorphic multiplication supported by the Elgamal encryption, the $\mathcal{SP}$ cannot determine the plaintext of R_e. Thus, the $\mathcal{SP}$ does not know the reachability of query nodes.

By exploiting the preservation of the reachability of any two nodes, we prove that pp-2-hop protects the graph structure from the $\mathcal{SP}$. It is straightforward to argue that it is not possible to determine the *existence of an edge* in a graph under pp-2-hop. Hence, it is not possible to infer the topology of the graph structure.

Proposition 3. *The $\mathcal{SP}$ can determine the existence of an edge only if it breaks either the one-way collision-resistant hash function or the Elgamal encryption.*

Proof. We establish the proposition via proof by contradiction. Suppose the $\mathcal{SP}$ can determine the existence of one edge (u,v). The $\mathcal{SP}$ has broken the reachability of at least one query $u \rightsquigarrow v$. By Prop. 2, this is possible only if the $\mathcal{SP}$ breaks either the one-way collision-resistant hash function or the Elgamal encryption.

Privacy against Size-Based Attack. In addition to the analysis of privacy against ciphertext only attack, we prove privacy under *size-based attack*.

Proposition 4. *When δ is set to 0, the reachability of the query nodes is perfectly protected against size-based attack.*

Proof. We prove the proposition via proof of contradiction. Suppose the $\mathcal{SP}$ can determine the reachability of the query nodes, $u_e \rightsquigarrow v_e$, under size-based attack. The $\mathcal{SP}$ can thus infer the reachability from (1) the size of $\texttt{Lout}^{\texttt{e}}(u_e) \cap \texttt{Lin}^{\texttt{e}}(v_e)$; and (2) the size of both $\texttt{Lout}^{\texttt{e}}(u)$ and $\texttt{Lin}^{\texttt{e}}(v_e)$. However, the size of $\texttt{Lout}^{\texttt{e}}(u_e) \cap \texttt{Lin}^{\texttt{e}}(v_e)$ always exactly equals $I_{\texttt{max}}$, and $|\texttt{Lout}^{\texttt{e}}(u_e)| = \texttt{Lout}_{\texttt{max}}$, $|\texttt{Lin}^{\texttt{e}}(v_e)| = \texttt{Lin}_{\texttt{max}}$. Therefore, the $\mathcal{SP}$ gains zero information content from the sizes.

Proposition 5. *When δ is set to 0, the graph structure is perfectly protected against size based attack.*

Proof. The proof is similar to that of Prop. 4.

In practice, δ may not necessarily be set to 0 as the sizes of $\texttt{Lin}^{\texttt{s}}$ and $\texttt{Lout}^{\texttt{s}}$ do not directly represent the connectivity of a node after surrogates are added to $\texttt{Lin}^{\texttt{s}}$ and Lout. However, a non-zero value of δ requires non-trivial privacy analysis to quantify the information leakage. Hence, we omit its analysis.

6 Experimental Evaluation

In this section, we present the experimental evaluation that verifies the performance of our proposed techniques and the effectiveness of our optimization.

6.1 Experimental Setup

Running platform. We conducted all experiments using a machine with Intel Core i3-2310 2.10GHz CPU and 4G RAM running Windows 7 OS. All algorithms were implemented using C++ based on the implementation of 2-hop labelings provided by R.Bramandia et al. [3]. The hash function (h_{s_1} and h_{s_2}) was 160-bit SHA-1 using two different salts. The encryption E was 1024-bit Elgamal [11].

Datasets. We used three synthetic datasets (denoted as SYN) and four real-world datasets. Some of their characteristics are shown in Tables 1 and 2. The synthetic datasets were all scale-free graphs, which are popular in experimentation. The generator used was provided by Choi et al. [32]. We controlled the sizes and densities of the graphs by setting $\alpha = 0.27$ and $\beta = 10$. The real-world datasets are all publicly available.[6]

Table 1. Synthetic datasets

Synthetic graph G	$\|V(G)\|$	$\|E(G)\|$	$\|E(G)\|/\|V(G)\|$
SYN-1	3073	37615	12.24
SYN-2	5651	15968	2.83
SYN-3	4880	27946	5.73

Table 2. Real-world datasets

Real graph G	$\|V(G)\|$	$\|E(G)\|$	$\|E(G)\|/\|V(G)\|$
YEAST	2361	7182	3.04
ODLIS	2909	18419	6.33
ERDOS	6927	11850	1.71
ROGET	1022	5075	4.97

[6] YEAST: http://vlado.fmf.uni-lj.si/pub/networks/data/bio/Yeast/Yeast.htm
ODLTS: http://vlado.fmf.uni-lj.si/pub/networks/data/dic/odlis/Odlis.htm
ERDOS: http://vlado.fmf.uni-lj.si/pub/networks/data/Erdos/Erdos02.net
ROGET: http://vlado.fmf.uni-lj.si/pub/networks/data/dic/roget/Roget.htm

Query sets. For each of the synthetic and real-world datasets, we generated 1000 random queries, 50% of which were positive queries, generated from the transitive closure, and 50% were negative queries.

Heuristics. We have implemented the classical 2-hop heuristic `maxDensCover` [10], the heuristic `maxSetCover` $= |E_w \cap T'|$, also proposed by Cheng et al. [8], and our heuristic `maxISCover`. These heuristics are plugged into 2-hop construction (Sec. 3). δ is set to 0 by default.

6.2 Experiments on Synthetic Datasets

Effectiveness of `maxISCover`. Table 3 reports the comparison on $I_{\texttt{max}}$ of the above three heuristics. We can see that `maxISCover` *always* produced the smallest $I_{\texttt{max}}$ when compared to `maxDensCover` and `maxSetCover`, as `maxISCover` considers the intersection size for each iteration of the 2-hop construction. For instance, $I_{\texttt{max}}$ with `maxISCover` heuristic was 88 and 2.5 times smaller than that with `maxDensCover` and `maxSetCover` heuristics on average, respectively. We note that while `maxSetCover` was not proposed to minimize $I_{\texttt{max}}$, `maxSetCover` greedily determines centers that most cover the uncovered $T(G)$. Our experiment showed that this sometimes led to small $I_{\texttt{max}}$s.

Effectiveness of `unifyIS`. Next, we tested the algorithms for unification of the intersection results. In particular, Table 4 reports the comparison of the number of distinct surrogate nodes introduced by `unifyIS` (Algo. 1) and a baseline algorithm `Naive`. `Naive` chooses to add unused surrogate nodes into the index rather than checking if the surrogate nodes from previous iterations can be reused. The number of added surrogate nodes in `MASN` was almost always at least three times fewer than that of `Naive` under all other heuristics. Moreover, as the $I_{\texttt{max}}$ was the smallest under `maxISCover` (Table 3), such an $I_{\texttt{max}}$ leads to the smallest distinct number of surrogate nodes, except in SYN-1.

Table 3. The maximum intersection size $I_{\texttt{max}}$

Graph	$I_{\texttt{max}}$		
	`maxDensCover`	`maxSetCover`	`maxISCover`
SYN-1	2558	22	15
SYN-2	17	7	3
SYN-3	1169	48	13

Table 4. # of distinct added surrogate nodes

Graph	`Naive` vs. `MASN`		
	`maxDensCover`	`maxSetCover`	`maxISCover`
SYN-1	7.86M vs. 12.11k	67.61k vs. 17.30k	46.10k vs. 13.73k
SYN-2	96.07k vs. 8.75k	39.56k vs. 6.24k	16.95k vs. 6.08k
SYN-3	5.70M vs. 23.03k	0.23M vs. 18.92k	63.44k vs. 11.11k

Table 5. Query time at $\mathcal{SP}$ and client

Graph	$\mathcal{SP}$ (ms) vs. Client (ms)		
	`maxDensCover`	`maxSetCover`	`maxISCover`
SYN-1	106.54 vs. 0.52	2.55 vs. 0.43	2.02 vs. 0.46
SYN-2	2.01 vs. 0.56	1.37 vs. 0.67	1.79 vs. 0.47
SYN-3	52.35 vs. 0.54	4.44 vs. 0.52	2.15 vs. 0.52

Table 6. Throughput at $\mathcal{SP}$

Graph	$\mathcal{SP}$ (query per second)		
	`maxDensCover`	`maxSetCover`	`maxISCover`
SYN-1	9	392	495
SYN-2	495	730	559
SYN-3	19	225	465

Query performance and throughput of `pp-2-hop`. Table 5 presents the query time at both the $\mathcal{SP}$ and the client side. Each of the reported times is the average of 1000 queries. For the query time at the $\mathcal{SP}$ side, as the $I_{\texttt{max}}$s due to `maxISCover` were small, the times of multiplications on the flags were small. Therefore, the query times of `maxISCover` were the best in all cases. For SYN-1 and SYN-3, `maxISCover` is more than an order of magnitude faster than `maxDensCover`; for SYN-3, `maxISCover` is more than twice as fast as `maxDensCover`. At the client side, the client only needs to

perform *one* decryption of R_e for every query and the decryption algorithm essentially did the same amount of computation. Thus, the times at the client side were roughly the same and very small.

Based on the query performances, we calculate the corresponding throughput of the $\mathcal{SP}$ in Table 6. The results showed that with a commodity machine, the $\mathcal{SP}$ using `maxISCover` consistently offers a throughput around 500 queries per second. In comparison, `maxDensCover` is the least efficient. While `maxSetCover` sometimes has comparable throughputs, it is more sensitive to the datasets used.

Table 7. The maximum intersection size $I_{\max}$

Graph	$I_{\max}$		
	`maxDensCover`	`maxSetCover`	`maxISCover`
YEAST	237	6	4
ODLIS	274	3	3
ERDOS	250	5	3
ROGET	752	6	4

Table 8. # of distinct added surrogate nodes

Graph	`Naive` vs. `MASN`		
	`maxDensCover`	`maxSetCover`	`maxISCover`
YEAST	0.56M vs. 7.72K	14.17K vs. 2.81K	9.44K vs. 2.72K
ODLIS	0.80M vs. 8.39K	8.73K vs. 2.98K	8.73K vs. 2.98K
ERDOS	1.73M vs. 8.86K	34.64K vs. 7.03K	20.78K vs. 7.01K
ROGET	0.77M vs. 3.75K	6.13K vs. 1.40	4.09K vs. 1.28K

Table 9. Query time at $\mathcal{SP}$ and client

Graph	$\mathcal{SP}$ (ms) vs. Client (ms)		
	`maxDensCover`	`maxSetCover`	`maxISCover`
YEAST	12.38 vs. 0.58	0.89 vs. 0.50	0.73 vs. 0.47
ODLIS	13.33 vs. 0.68	0.59 vs. 0.50	0.65 vs. 0.53
ERDOS	11.98 vs. 0.55	1.34 vs. 0.59	0.97 vs. 0.52
ROGET	31.91 vs. 0.64	0.57 vs. 0.59	0.29 vs. 0.64

6.3 Experiments on Real-World Datasets

Finally, we conducted a similar evaluation on four publicly available real-world datasets. Since the results were similar to those obtained from synthetic datasets, we only highlight some major results here.

Table 7 shows the performances of `maxISCover`. Our proposed `maxISCover` heuristic consistently produced the smallest $I_{\max}$ when compared to the other two heuristics. Since the $I_{\max}$s due to `maxISCover` were the smallest, the number of distinct added surrogate nodes by `unifyIS` were also the smallest as shown in Table 8. The query time at both the $\mathcal{SP}$ side and the client side are shown in Table 9. The query time of `maxISCover` at the $\mathcal{SP}$ side was almost always at least an order of magnitude faster than that of `maxDensCover`.

7 Conclusion

In this paper, we investigated privacy-preserving reachability query services. We proposed heuristic algorithms to determine a `2-hop` labeling called `pp-2-hop`. We proposed and analyzed its private query processing over `pp-2-hop`. We conducted experiments to show the performance of our techniques. In the future, we plan to (i) integrate the large body of optimizations for `2-hop` labeling into `pp-2-hop` (*e.g.*, [30]) and (ii) implement the shortest distance queries which are supported by the original `2-hop` labeling [10].

Acknowledgement. Zhe Fan, Peipei Yi and Byron Choi were partially supported by GRF 210510. Shuxiang Yin and Shuigeng Zhou were supported by the Research Innovation Program of Shanghai Municipal Education Commission under grant No.13ZZ003.

References

1. R. Agrawal, A. Borgida, and H. V. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. SIGMOD, 1989.
2. G. D. Bader, M. P. Cary, and C. Sander. Pathguide: a pathway resource list. *Nucleic Acids Research*, 34(suppl 1):D504–D506, 2006.
3. R. Bramandia, B. Choi, and W. K. Ng. Incremental maintenance of 2-hop labeling of large graphs. *TKDE*, 22(5):682–698, 2010.
4. J. Cai and C. K. Poon. Path-hop: efficiently indexing large graphs for reachability queries. CIKM, pages 119–128, 2010.
5. N. Cao, Z. Yang, C. Wang, K. Ren, and W. Lou. Privacy-preserving query over encrypted graph-structured data in cloud computing. In *ICDCS*, pages 393–402, 2011.
6. J. Cheng, S. Huang, H. Wu, and A. Fu. Tf-label: a topological-folding labeling scheme for reachability querying in a large graph. SIGMOD, pages 193–204, 2013.
7. J. Cheng, J. X. Yu, X. Lin, H. Wang, and P. S. Yu. Fast computation of reachability labeling for large graphs. EDBT, pages 961–979, 2006.
8. J. Cheng, J. X. Yu, X. Lin, H. Wang, and P. S. Yu. Fast computing reachability labelings for large graphs with high compression rate. EDBT, pages 193–204, 2008.
9. B. Chor et al. Private information retrieval. *J. ACM*, 45:965–981, 1998.
10. E. Cohen et al. Reachability and distance queries via 2-hop labels. SODA, 2002.
11. T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, pages 10–18, 1985.
12. Z. Fan, Y. Peng, B. Choi, J. Xu, and S. S. Bhowmick. Towards efficient authenticated subgraph query service in outsourced graph databases. *IEEE Transactions on Services Computing*, 99, 2013.
13. J. Gao, J. X. Yu, R. Jin, J. Zhou, T. Wang, and D. Yang. Neighborhood-privacy protected shortest distance computing in cloud. SIGMOD, pages 409–420, 2011.
14. X. He, J. Vaidya, B. Shafiq, N. Adam, and X. Lin. Reachability analysis in privacy-preserving perturbed graphs. WI-IAT, pages 691–694, 2010.
15. Informatics Outsourcing. Outsourcing Solution Service. *http://www.informaticsoutsourcing.com/*.
16. R. Jin, N. Ruan, S. Dey, and J. Y. Xu. Scarab: scaling reachability computation on large graphs. SIGMOD, pages 169–180, 2012.
17. R. Jin, N. Ruan, Y. Xiang, and H. Wang. Path-tree: An efficient reachability indexing scheme for large directed graphs. *TODS*, pages 7:1–7:44, 2011.
18. R. Jin, Y. Xiang, N. Ruan, and D. Fuhry. 3-hop: a high-compression indexing scheme for reachability query. SIGMOD, pages 813–826, 2009.
19. V. Karwa, S. Raskhodnikova, A. Smith, and G. Yaroslavtsev. Private analysis of graph structure. In *VLDB*, pages 1146–1157, 2011.
20. J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2007.
21. A. Kundu et al. How to authenticate graphs without leaking. In *EDBT*, pages 609–620, 2010.
22. A. Kundu et al. Efficient leakage-free authentication of trees, graphs and forests. *IACR Cryptology ePrint Archive*, page 36, 2012.
23. D. A. Menascé. Qos issues in web services. *Internet Computing*, 6(6):72–75, Nov. 2002.
24. K. Mouratidis et al. Shortest path computation with no information leakage. *PVLDB*, 2012.
25. R. Schenkel, A. Theobald, and G. Weikum. HOPI: An efficient connection index for complex XML document collections. In *EDBT*, pages 237–255, 2004.
26. S. Seufert, A. Anand, S. Bedathur, and G. Weikum. Ferrari: Flexible and efficient reachability range assignment for graph indexing. ICDE, pages 1009–1020, 2013.
27. S. Trissl and U. Leser. Fast and practical indexing and querying of very large graphs. SIGMOD, pages 845–856, 2007.
28. S. J. van Schaik and O. de Moor. A memory efficient reachability data structure through bit vector compression. SIGMOD, pages 913–924, 2011.
29. K. Xu, L. Zou, J. X. Yu, L. Chen, Y. Xiao, and D. Zhao. Answering label-constraint reachability in large graphs. CIKM, pages 1595–1600, 2011.
30. P. Yi, Z. Fan, and S. Yin. Privacy-preserving reachability query services for sparse graphs. GDM, 2014.
31. H. Yildirim, V. Chaoji, and M. J. Zaki. Grail: scalable reachability index for large graphs. *PVLDB*, 3(1-2):276–284, 2010.
32. L. Zhu, B. Choi, B. He, J. X. Yu, and W. K. Ng. A uniform framework for ad-hoc indexes to answer reachability queries on large graphs. DASFAA, pages 138–152, 2009.

A Appendix: Proof of Proposition 1

PROPOSITION 1. The problem of MASN is NP-hard.

Proof. (Sketch) The hardness is established by a simple reduction from the classical MINIMUM VERTEX COVER problem (MVC): "*Given a graph $G = (V, E)$, determine the smallest subset V' of V such that for each edge (u, v) in E, either u or v is a member of V'.*"

Reduction. Consider an instance of the MVC problem is $G = (V, E)$. We construct an instance of the MASN problem (*i.e.*,2-hop labels) described as follows: Each v_i in V corresponds to a surrogate node d_i, which can be added to Lins or Louts. Each edge (v_i, v_j) in E denotes an intersection $\texttt{Lin}(a_i) \cap \texttt{Lout}(b_j)$. We construct Lins and Louts in a special way such that the following rules are true:

1. Adding either d_i or d_j to both $\texttt{Lin}(a_i)$ and $\texttt{Lout}(b_j)$ makes $|\texttt{Lin}(a_i) \cap \texttt{Lout}(b_j)|$ exactly $I_{\texttt{max}}$.
2. Adding d_i or d_j to either $\texttt{Lin}(a_l)$ or $\texttt{Lout}(b_m)$ denoted by an edge (v_l, v_m) makes $|\texttt{Lin}(a_l) \cap \texttt{Lout}(b_j)| > I_{\texttt{max}}$ or $|\texttt{Lin}(a_i) \cap \texttt{Lout}(b_m)| > I_{\texttt{max}}$, where $v_i \neq v_l$ and $v_j \neq v_m$.

The above two rules state that d_i or d_j can only be added $\texttt{Lin}(a_i)$ and $\texttt{Lout}(b_j)$ and nothing else. Finally, we add real center nodes to Lin and Lout such that the sizes of Lins and Louts are respectively identical. Therefore, the constraint of MASNthat minimizes $\texttt{Lin}_{\texttt{max}}$ and $\texttt{Lout}_{\texttt{max}}$ has no effect when determining a solution of this MASNinstance. The size of the 2-hop labels constructed in this way is at most $|V|I_{\texttt{max}} \times 2|V|$, where $I_{\texttt{max}}$ are needed to encode the second condition for each v_i in V and $2|V|$ are the total number of Lins and Lout.

Analysis of the solution of MASN. Suppose that we have determined the solution of MASN D_w where all intersection results are $I_{\texttt{max}}$ D_w, which is a set of surrogate nodes used. First, either d_i or d_j has been added to $\texttt{Lin}(a_i) \cap \texttt{Lout}(b_j)$ for all intersections. Each intersection is an edge. Denote $C = \{v_i \text{ — } d_i \in D_w\}$. C is a vertex cover. We can have "D_w is minimized *if and only if* C is minimized" by a simple proof by contradiction. Thus, we obtain a solution for MVC by solving MASN. Since MVC is NP-hard, MASN is NP-hard.