Privacy-Preserving Reachability Query Services for Massive Networks

Jiaxin Jiang, Peipei Yi, Byron Choi, Zhiwei Zhang
Department of Computer Science
Hong Kong Baptist University
{jxjian,csppyi,bchoi,cszwzhang}@comp.hkbu.edu.hk

Xiaohui Yu
School of Computer Science and Technology
Shandong University
xyu@sdu.edu.cn

ABSTRACT

This paper studies privacy-preserving reachability query services under the paradigm of data outsourcing. Specifically, graph data have been outsourced to a third-party service provider ($\mathcal{SP}$), query clients submit their queries to the $\mathcal{SP}$, and the $\mathcal{SP}$ returns the query answers to the clients. However, the $\mathcal{SP}$ may not always be trustworthy. Hence, this paper investigates protecting the structural information of the graph data and the query answers from the $\mathcal{SP}$. Existing techniques are either insecure or not scalable. This paper proposes a privacy-preserving labeling, called `ppTopo`. To our knowledge, `ppTopo` is the first work that can produce reachability index on massive networks *and* is secure against known plaintext attacks (KPA). Specifically, we propose a *scalable index construction algorithm* by employing the idea of topological folding, recently proposed by Cheng et al. We propose a novel *asymmetric scalar product encryption in modulo 3* (ASPE3). It allows us to encrypt the index labels and transforms the queries into scalar products of encrypted labels. We perform an experimental study of the proposed technique on the SNAP networks. Compared with the existing methods, our results show that our technique is capable of producing the encrypted indexes at least 5 times faster for massive networks and the client's decryption time is 2-3 times smaller for most graphs.

Keywords

Graph Databases; Data and Query Privacies; Reachability Queries

1. INTRODUCTION

Graphs have been found in many emerging applications including bioinformatics analysis, communication networks, social networks, knowledge networks and semi-structured databases. Due to the massive volume of graph data from such a wide range of applications and the IT resources required to evaluate numerous queries at large scale, it is economically appealing to outsource the data to a third-party service provider ($\mathcal{SP}$) who provides query services.

An application scenario. Suppose the police are at an early stage of identifying a terror suspect who is studying in a university. The police may issue numerous queries – "*who are reachable by the suspect?*" – on the communication network (*e.g.*, email or phone communications) owned by the university. While the university is required by law to cooperate with the police, it does not prefer its daily operations affected by the queries. Hence, when needed, it may outsource its network to an $\mathcal{SP}$ for processing such queries. At the meantime, the network should be protected from the $\mathcal{SP}$ as it contains sensitive information of its staff and students. Importantly, the police do not prefer to expose their query answers to the $\mathcal{SP}$, which in turn expose their investigations. Therefore, it is imperative to propose a scheme for privacy-preserving query services.

In this paper, we consider the service of privacy-preserving *reachability* query, which is one of the most popular and fundamental queries on graphs [27, 9, 23, 14, 31, 25, 16, 2, 22, 7, 1, 29, 26, 15, 27] and their applications include pattern matching in graphs, XML and Semantic Web, and analyzing gene ontology and call graphs of computer programs. This paper considers the well-received system model from data outsourcing literature: In particular, the data owner encrypts their data graphs and outsources to an $\mathcal{SP}$. The data owner sends the secret key to the query client to encrypt the queries. The client sends the encrypted queries to the $\mathcal{SP}$. The $\mathcal{SP}$ then processes the queries in the encrypted domain and returns the encrypted query results for the client to decrypt. We assume that the $\mathcal{SP}$ may be an attacker, and he/she is honest but curious (a.k.a semi-honest).

Our private querying technique is proposed on top of the `2-hop` labeling scheme. The main benefits of the `2-hop` approach are threefold. First, the `2-hop` labels (introduced later) are simple: each vertex is only associated with two sets of vertices. Second, the query evaluation of `2-hop` labeling is simply an *intersection* between two sets. Such simple structure and algorithm make the analysis of privacy simple as well. Third, the large body of works for `2-hop` labeling (*e.g.*, [7, 1]) can be adopted.

The technical challenges of such a privacy-preserving query service are mainly twofold. Firstly, data graphs can be large nowadays. It has been known that the more secure the private query processing is, the less time-efficient it is. In particular, encrypting the data graph itself can be computationally costly. Thus, it has always been a design concern to strike a balance between the security and efficiency of a pri-

CIKM'16 , October 24-28, 2016, Indianapolis, IN, USA

DOI: http://dx.doi.org/10.1145/2983323.2983799

vate query method. The second challenge is the index construction (in this case, 2-hop construction) of large graphs is time-consuming. While there have been scalable solutions for 2-hop construction, it remains open how one may apply (if ever possible) a scalable solution to privacy-preserving query processing.

A recent attempt is a privacy-preserving 2-hop, called (pp-2-hop) [32]. The encryption and querying algorithms are derived from a partially homomorphic encryption, namely the ElGamal encryption. Despite its privacy against ciphertext-only attacks (COA), pp-2-hop suffers from its potentially large size (due to privacy requirement), and the known problems of high 2-hop construction cost. A follow-up work [30] enhances both time- and space-efficiencies of [32]. The security of this work is also COA, despite its claim on chosen-plaintext attacks (CPA).

In this paper, we make several new contributions. We propose *privacy-preserving topological folding* 2-hop *labeling*, called ppTopo. It includes an encoding and encryption for private intersections, called asymmetric scalar-product preserving encryption in modulo 3 (ASPE3). We propose a 2-hop construction inspired by topological folding and it optimizes private intersections, derived from private scalar product [28]. As a result, we yield a 2-hop labeling that can be run on massive networks and is private against *known plaintext attack* (KPA). To our knowledge, this is the first work that private reachability queries can be evaluated on networks with 69 million edges.

Contributions. The contributions of this paper can be summarized as follows:

- We propose a novel encryption scheme, adopted from asymmetric scalar-product preserving encryption [28], for private query processing. We propose a new heuristic construction by employing the topological folding idea that optimizes for the query performance;
- We analyze the privacies of our approach. Our finding is that our approach is secure against KPA; and
- We present an experimental evaluation on the real networks. Compared with the existing methods, our technique is capable of producing the encrypted indexes at least 5 times faster for massive networks and the client's decryption time is 2-3 times smaller for most graphs.

2. BACKGROUND AND PROBLEM FORMULATION

In this section, we provide the background of the paper and formulate the problem being studied. The background includes the data model, the system model, the attack models and the privacy targets.

Data model. In this paper, we consider a *directed node-labeled graph* $G = (V, E)$ where V and E are the vertex and edge sets of G, respectively. We may use the terms *vertex* and *nodes* interchangeably. For $u, v \in V$, u can *reach* v if there exists a directed path from u to v. We use $u \rightsquigarrow v$ to denote that. A strongly connected component *(SCC)* of a graph G is a maximal set of vertices $C \subseteq V$ such that $\forall u, v \in C$, $v \rightsquigarrow u$ and $u \rightsquigarrow v$. Since the reachability information of vertices of an *SCC* is trivial, we assume an arbitrary graph has been *reduced* to a directed acyclic graph (DAG) G^{SCC}, where each *SCC* is represented by a supernode. G^{SCC} can be constructed by classical algorithms such as Tarjan's algorithm [24], that runs in $O(|V| + |E|)$. Since the technical discussions always assume the reduced graph, for notation simplicity, the subsequent discussion uses G instead of G^{SCC}, unless otherwise specified. A *reachability query* on G takes two vertices u and v as input, denoted as $\mathsf{Reach}(u, v)$, and returns true *iff* v is reachable from u.

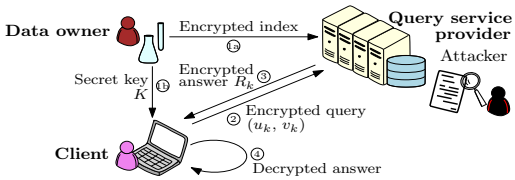


Figure 1: Overview of the system model

System model. In the literature of database outsourcing, the system model typically contains the following three parties (Figure 1). We follow this model in the paper.

1. *Data owner* ($\mathcal{DO}$): A data owner owns the graph data and precomputes indexes *offline once*. It outsources the encrypted data and indexes to a *Service Provider* ($\mathcal{SP}$), and sends query clients a secret key K, to encrypt queries and decrypt query results;

2. *Service provider* ($\mathcal{SP}$): An $\mathcal{SP}$ is often equipped with highly available, powerful computing utilities such as a cloud. The $\mathcal{SP}$ processes voluminous encrypted query requests like (u_k, v_k) on encrypted data and returns the encrypted answers R_k to clients; and

3. *Client*: A client encrypts his/her query using the secret key from the $\mathcal{DO}$, sends it to the $\mathcal{SP}$ and decrypts the answer R_k from the $\mathcal{SP}$. We assume that the clients and $\mathcal{SP}$ do not collude.

Attack model and privacy target. The attackers may be the $\mathcal{SP}$ or another adversary hacking the $\mathcal{SP}$. For simplicity, we *term the attackers as the* $\mathcal{SP}$. We assume the $\mathcal{SP}$s are *honest-but-curious*. That is, the $\mathcal{SP}$ performs computations according to the system model, but the $\mathcal{SP}$ may be interested in inferring private information.

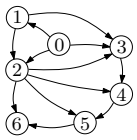
There has been a well-known tradeoff between privacy and utility of database outsourcing. The techniques that are more secure can often be inefficient and hence have less utility. This paper studies *cryptographic attack models* including ciphertext only attack (COA) and known plaintext attack (KPA), where KPA is known to be harder to execute than COA. Under a cryptographic attack, our *privacy target* is to keep the following three pieces of information private.

- *Reachability of the query vertices* a.k.a the query result. In particular, given a reachability query $\mathsf{Reach}(u, v)$, the $\mathcal{SP}$ cannot infer whether $u \rightsquigarrow v$;

- *Graph structure* a.k.a the topology of the data graph. The $\mathcal{SP}$ cannot infer the existence of an edge; and

- No structural information in addition to $\mathsf{Reach}(u, v)$ is made known to the client.

Problem statement. With the background presented above, we are ready to present the problem statement of this paper: The client submits the reachability query to the $\mathcal{SP}$ side and obtains the query answers from $\mathcal{SP}$ and meanwhile, the privacy target is met against the $\mathcal{SP}$'s attack model.



v	$\texttt{Lin}(v)$	$\texttt{Lout}(v)$
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

Figure 2: A small network (LHS) and its 2-hop labels (RHS)

3. BACKGROUND OF 2-HOP LABELING

This section provides the background for 2-hop labeling, which forms the basis of our discussion of privacy-preserving techniques for efficient reachability queries. For the details of 2-hop labeling, please refer to the seminal paper by Cohen et al. [9] and its latest developments [6].

The 2-hop labels. Each vertex $u \in V(G)$ is associated with two sets of vertices, denoted as $\texttt{Lout}(u)$ and $\texttt{Lin}(u)$, called 2-hop *labels*. Vertices in $\texttt{Lout}(u)$ (resp., $\texttt{Lin}(u)$) can be reachable from u (resp., can reach u), which are also called *center nodes*. A reachability query can be evaluated by an intersection of 2-hop labels as follows: Given two vertices u and v, $u \rightsquigarrow v$ *if and only if* $\texttt{Lout}(u) \cap \texttt{Lin}(v) \neq \emptyset$.

EXAMPLE 3.1. Consider a small communication network shown in the LHS of Figure 2. The vertex ID represents the user ID and the edge denotes a message from one user to another. A possible 2-hop labeling of the graph is shown in the RHS of Figure 2. Vertex 1 can reach Vertex 5, *i.e.*, $1 \rightsquigarrow 5$. $\texttt{Lout}(1) \cap \texttt{Lin}(5) = \{5\}$. Vertex 0 is not reachable from Vertex 6 as $\texttt{Lout}(6) \cap \texttt{Lin}(0) = \emptyset$.

It is apparent that when $\texttt{Lin}(v)$ and $\texttt{Lout}(v)$ contains all nodes that can reach v and those v can reach, for all $v \in V(G)$, such 2-hop labels are the transitive closure $T(G)$ of G. Since $T(G)$ can be large, there has been a stream of work to ensure that the 2-hop labels are small and meanwhile, cover all the reachability (a.k.a connectivity) information of the network, *i.e.*, all elements in $T(G)$. However, constructing the 2-hop labels of the minimum size has known to be NP-hard. There are various works that significantly optimize the time and scalability of 2-hop construction (*e.g.*, [7, 6]).

Outline of the 2-hop construction (offline processing). As we shall present our techniques with reference to 2-hop construction, we briefly outline the heuristic construction approach [9]. Existing 2-hop constructions mainly focus on minimizing the size of 2-hop labeling, defined as $\sum_{u \in V(G)}(|\texttt{Lout}(u)| + |\texttt{Lin}(u)|)$. 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 $\texttt{maxDensCover}$ defined below.

$$\texttt{maxDensCover}(w) = \frac{|E_i \cap T'|}{|L_i \cup R_i|} \quad (1)$$

In each iteration, the node w with the largest $\texttt{maxDensCover}$ 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.

Existing privacy preserving 2-hop labeling. Since the heuristic $\texttt{maxDensCover}$ was not designed for privacy preservation, the 2-hop labels may leak non-trivial graph structures. For example, if the size of $\texttt{Lin}(v) \cap \texttt{Lout}(u)$ is large, it implies that there are a few distinct paths between u and v. In addition, since Formula 1 iteratively determines the center that covers the densest subgraph G_i, the center nodes are likely to be highly connected nodes.

Two recent works ([32] and [30]) propose techniques to protect the sensitive information. A summary of the approaches and this paper is presented in Table 1. Yin et al. [32] propose pp-2-hop that optimizes the labels such that *all* Lin labels (respectively, Lout labels), and meanwhile, *all* intersection results of the same size. The node IDs of pp-2-hop are hashed. The contents of the labels are encrypted by the ElGamal encryption scheme. *This is the main reason why the $\mathcal{SP}$ does not learn any structural information from the encrypted labels (input) and the query results.* One drawback of this technique is that the unification of label (and respectively, the result) sizes may lead to a significant increase in sizes. The other drawback is that the encrypted node IDs are available to the $\mathcal{SP}$. The $\mathcal{SP}$ analyzes the distributions of the encrypted ID. Yi et al. [30] observed that real graphs are often sparse, most reachable queries return negative answer, and the intersections of the 2-hop labels of these queries are the empty set. To ensure the encrypted outputs are of the same size, Yi et al. [30] propose mui-2-hop that pre-processes the 2-hop labels such that the encrypted result is always a singleton set. To enhance performance, it does not unify the respective sizes of Lin and Lout labels.

mui-2-hop can only be secure against COA but not KPA. The reason is that the $\mathcal{SP}$ has the encrypted 2-hop labels and can enumerate the encrypted intersection results offline. The intersection results of mui-2-hop are always a singleton set containing either a true or false flag. If the $\mathcal{SP}$ can simply check a pair of the plaintext and ciphertext of true or false, the $\mathcal{SP}$ can determine the transitive closure of the graph, and the $\mathcal{SP}$ obtains the answers of all queries. This violates the privacy target of this paper.

4. ASYMMETRIC SCALAR-PRODUCT PRESERVING ENCRYPTION IN MODULO 3 FOR PRIVATE INTERSECTION

In Section 4, we propose asymmetric scalar product preserving encryption in modulo 3 (ASPE3), to efficiently support private intersection: "$\texttt{Lin}(v) \cap \texttt{Lout}(u) = \emptyset$?". ASPE3 is based on the asymmetric scalar-product preserving encryption (ASPE) proposed for private kNN queries [28], that facilitates private-preserving distance computation.

The overview of the application of ASPE3 to our problem is presented in Figure 3. The figure shows that it is assumed the $\mathcal{DO}$ preprocesses the graphs into 2-hop labels by topological folding. The $\mathcal{DO}$ then uses ASPE3 to encode 2-hop label sets as binary vectors. The vectors are split and encrypted. The $\mathcal{DO}$ passes the secret keys (an invertible-matrix pair and the splitting information) to the client. The client encrypts a matrix for querying with the secret keys. The $\mathcal{SP}$ performs the private intersections and returns the encrypted answers to the client to decrypt. We then elaborate the main procedures of ASPE3: (i) encoding of labels and (ii) encrypting the encoded labels.

Table 1: A summary of approaches

Method	Encryption	Approach	Strengths	Weaknesses
pp-2-hop[32]	ElGamal	unify and encrypt respectively the sizes of encrypted Lin labels, those of Lout labels, and the encrypted result sizes	secure against COA	slow and not scalable
mui-2-hop[30]	AES	determine the encrypted Lin and Lout s.t. their results are always an encrypted singleton set	secure against COA	not scalable
pp-tflabel	ASPE3	process queries via private intersections	secure against KPA & scalable	none

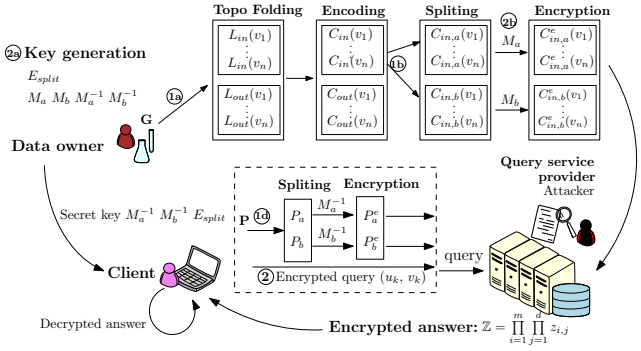


Figure 3: Overview of ppTopo

4.1 Encoding scheme

1a. Encoding 2-hop *labels as vectors:* Consider a query $\mathsf{Reach}(u, v)$. For simplification, we remove v and u from the notations of $\mathtt{Lin}(v)$ and $\mathtt{Lout}(u)$, when they are clear from the context. We encode Lin into a vector $C_{in} = (c_1, c_2, \ldots, c_{|W|})$, where W is the set of centers.

The value of c_i is defined as:

$$c_i = \begin{cases} 1, & \text{if } w_i \in \mathtt{Lin} \\ 0, & \text{otherwise} \end{cases}$$

Next, we encode Lout into a vector $C_{out} = (c_1, c_2, \ldots, c_{|W|})$. The value of c_i is:

$$c_i = \begin{cases} 2, & \text{if } w_i \in \mathtt{Lout} \\ 1, & \text{otherwise} \end{cases}$$

In this encoding, the sum of c_i in C_{in} and c_i in C_{out} is 3 if and only if w_i exists in both Lin *and* Lout*. C_{in}* may be a long vector because the set of centers W may be large. We decompose C_{in} into $m = \lceil \frac{|V|}{d} \rceil$ d-dimensional fragment vectors for efficient manipulation of vectors. The i-th fragment vector is $C_{in}^i = (c_1, c_2, \ldots, c_d)$ where c_j of C_{in}^i is equal $c_{d\times(i-1)+j}$ of C_{in}.

Example 4.1. *Assume a* 2-hop *labeling where $|W| = 8$, $\mathtt{Lin}(v) = \{1, 2, 3, 6, 7\}$ and $\mathtt{Lout}(u) = \{3, 4\}$. $\mathtt{Lin}(v)$ is encoded into $C_{in}(v) = (1, 1, 1, 0, 0, 1, 1, 0)$. Suppose d is 4. Then $C_{in}(v)$ is decomposed into fragment vectors $C_{in}^1 = (1, 1, 1, 0)$ and $C_{in}^2 = (0, 1, 1, 0)$. Similarly, $\mathtt{Lout}(u)$ is encoded into $(1, 1, 2, 2, 1, 1, 1, 1)$. Then, it is decomposed into fragment vectors $C_{out}^1 = (1, 1, 2, 2)$ and $C_{out}^2 = (1, 1, 1, 1)$. When we add C_{in} and C_{out}, only the entries at the 3rd position are added up to 3, which will be used to signify a common center of C_{in} and C_{out}.*

1b. Splitting the vectors to provide security. After encoding labels as vectors, we split them based on a secret *configuration bit-vector* E_{split}. Since we apply the same splitting to the vectors of Lin and Lout labels and their fragments, we omit the superscript for indicating fragment ID and subscript for indicating Lin and Lout of C for a concise presentation. Each fragment vector $C = (c_1, c_2, \ldots, c_d)$ split into two vectors $C_a = (a_1, a_2, \ldots, a_d)$ and $C_b = (b_1, b_2, \ldots, b_d)$, such that

- $a_j = b_j = c_j$, if $E_{split}[j] = 1$; and
- $a_j + b_j = c_j$, if $E_{split}[j] = 0$,

1c. Encoding and splitting of a query permutation matrix. The client generates a *random $d \times d$ query permutation matrix* (denoted as P) to hide the secret M^{-1} which is needed for query processing. P is a binary matrix that has exactly one entry of 1 in each row and each column and 0s elsewhere. To resist KPA, the client multiples P by a random factor r. The splitting of the permutation matrix is slightly different from that of the labels' vectors. Specifically, given E_{split}, P is split into two matrices $P_a = (x_{i,j})_{d\times d}$ and $P_b = (y_{i,j})_{d\times d}$ as follows:

- $x_{i,j} = y_{i,j} = P_{i,j}$, if $E_{split}[i] = 0$; and
- $x_{i,j} + y_{i,j} = P_{i,j}$, if $E_{split}[i] = 1$.

Example 4.2. *Consider the encoded and decomposed labels $C_{in}^1(v)$ and $C_{out}^1(u)$ in Example 4.1. The $\mathcal{DO}$ generates the keys (M_a, M_b, M_a^{-1}, M_b^{-1} and E_{split}) as showed in the leftmost part of Figure 4. Then, $C_{in}^1(v)$ is split into $C_{in,a}(v) = (1, 3, 2, 0)$ and $C_{in,b}(v) = (1, -2, -1, 0)$ according to E_{split}. Similarly, $C_{out}^1(u)$ is split into $C_{out,a}(u) = (1, 5, 4, 2)$ and $C_{out,b}(u) = (1, -4, -2, 2)$, as shown in the middle part of Figure 4. The splitting of the query permutation matrix is shown in the middle lower part of Figure 4. For presentation brevity, we omit the tedious operations that yield P_a and P_b.*

4.2 Encryption scheme

2a. Key generation: The $\mathcal{DO}$ generates two $d \times d$ invertible matrices, M_a and M_b, and their inverse matrices, M_a^{-1} and M_b^{-1}, and a secret configuration bit vector $E_{split} = (e_1, e_2, \ldots, e_d)$ offline. M_a, M_b and E_{split} are kept by the $\mathcal{DO}$, whereas M_a^{-1}, M_b^{-1} and E_{split} are assigned to the client.

2b. Encryption of labels' vectors: To encrypt 2-hop labels, the $\mathcal{DO}$ simply multiplies M_a by C_a and M_b by C_b, respectively. The corresponding ciphertexts can be denoted as $C_a^e = M_a^T {C_a}^T$ and $C_b^e = M_b^T {C_b}^T$. The $\mathcal{DO}$ then outsources the ciphertexts to the $\mathcal{SP}$.

2c. Encryption of query requests: The client uses the query permutation matrix (P_a and P_b) to encrypt the secrets M_a^{-1} and M_b^{-1}. The corresponding ciphertexts are $P_a^e = M_a^{-1} P_a$ and $P_b^e = M_b^{-1} P_b$. The client then sends the query request $\mathsf{Reach}(u, v)$ and the corresponding ciphertext to the $\mathcal{SP}$.

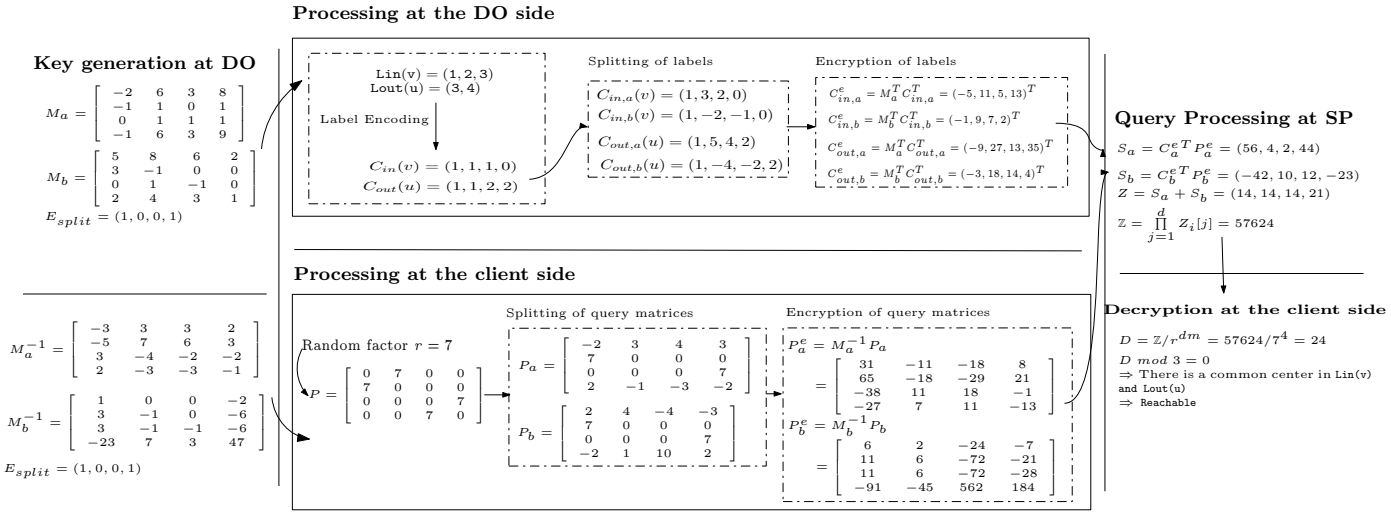


Figure 4: An example of ppTopo for a reachability query: $u \rightsquigarrow v = \texttt{true}$, where $\texttt{Lin}(v) \cap \texttt{Lout}(u) = \{3\}$

4.3 Private query processing

We first present the private query computation at the $\mathcal{SP}$ and then the decryption at the client side. Their correctness is presented in Theorem 4.1.

3a. Private intersection as vector computations at the $\mathcal{SP}$: The $\mathcal{SP}$ retrieves the corresponding encrypted labels ($C^e_{in,a}$, $C^e_{in,b}$) for $\texttt{Lin}(v)$ and ($C^e_{out,a}(u)$, $C^e_{out,b}$) for $\texttt{Lout}(u)$. The $\mathcal{SP}$ does the addition operation (in the encrypted domain) as the intersection operation of the plaintexts, $\texttt{Lin}(v)$ and $\texttt{Lout}(u)$:

- $\mathbb{C}^e_a = C^e_{in,a} + C^e_{out,a}$
- $\mathbb{C}^e_b = C^e_{in,b} + C^e_{out,b}$

The $\mathcal{SP}$ then computes the scalar products of $\mathbb{C}^e_a$ and P^e_a, denoted as $S_a = \mathbb{C}^{eT}_a P^e_a$, and similarly, computes S_b as $\mathbb{C}^{eT}_b P^e_b$. Recall that P^e_a is a product of M_a^{-1} and P_a. Intuitively, while the secret M_a^{-1} is used to compute S_a, the matrix has been permuted by P_a, which is not accessible by the $\mathcal{SP}$.

To reduce the communication costs, the $\mathcal{SP}$ *aggregates* the results so far into small messages. First, the $\mathcal{SP}$ computes $Z = S_a + S_b$. Second, it computes the product of the values of Z_i, where Z_i is the i-th fragment of the query result. The product can be denoted as $\mathbb{Z}_i = \prod_{j=1}^{d} Z_i[j]$. Then, the $\mathcal{SP}$ multiplies the products of all fragments as the *encrypted query answer*: $\mathbb{Z} = \prod_{i=1}^{m} \mathbb{Z}_i$.

3b. Decryption at the client side: The client decrypts the encrypted query answer $\mathbb{Z}$ as follows. The client eliminates the random factor of P by a division $D = \mathbb{Z}/r^{dm}$ (recall that there are m d-dimensional vectors). The value of r^{dm} can be computed in $O(\texttt{log}(dm))$ and only one division of the large numbers. Then, $u \rightsquigarrow v$ is true if and only if D is divisible by 3. Theorem 4.1 states the correctness of the private query processing.

Example 4.3. *Following up Example 4.2, the middle part of Figure 4 show the encryptions of the labels and the query matrices. The $\mathcal{DO}$ outsourced the encrypted labels to the $\mathcal{SP}$. The client sends the encrypted matrices to the $\mathcal{SP}$ for query processing. Note that the secrets M_a^{-1} and M_b^{-1} are not known to the $\mathcal{SP}$ as it has been encrypted with the query matrices (P_a and P_b). At query processing time, the $\mathcal{SP}$ retrieves the encrypted* 2-hop *labels of the query nodes C^e_a and C^e_b. Following the processing discussed in this subsection, the RHS of Figure 4 shows the $\mathcal{SP}$ computes S_a, S_b, Z and finally $\mathbb{Z}$, which is 57624. $\mathbb{Z}$ is returned to the client as an encrypted answer. The client simply eliminates the random number r from $\mathbb{Z}$ and performs one modulo 3 operation. Since the value is 0, there must be a common center in C_{in} and C_{out}, whose encoded values add up to 3. Therefore, the client obtains* true *as the answer.*

Theorem 4.1. *The value of D returned by private query processing implies the following: D mod 3 = 0 if and only if $u \rightsquigarrow v$.*

Proof. Recall from the definitions of the variables the following:

$$D = \mathbb{Z}/r^{dm} = (\prod_{i=1}^{m} \mathbb{Z}_i)/r^{dm} = (\prod_{i=1}^{m}\prod_{j=1}^{d} z_{i,j})/r^{dm} = \prod_{i=1}^{m}\prod_{j=1}^{d} z_{i,j}/r$$

Hence, the following statement holds.

$$D \bmod 3 = 0 \Leftrightarrow \exists i, j \; z_{i,j}/r \bmod 3 = 0 \tag{2}$$

Furthermore, we note that each fragment query result

$$\begin{aligned}
Z = S_a + S_b &= (\mathbb{C}^{eT}_a P^e_a + \mathbb{C}^{eT}_b P^e_b) \\
&= (C^e_{in,a} + C^e_{out,a})^T P^e_a + (C^e_{in,b} + C^e_{out,b})^T P^e_b \\
&= ((M_a^T (C_{in,a} + C_{out,a})^T)^T M_a^{-1} P_a + \\
&\quad (M_b^T (C_{in,b} + C_{out,b})^T)^T M_b^{-1} P_b) \\
&= (C_{in,a} + C_{out,a}) P_a + (C_{in,b} + C_{out,b}) P_b \\
&= (C_{in,a} P_a + C_{in,b} P_b) + (C_{out,a} P_a + C_{out,b} P_b) \\
&= (C_{in} P + C_{out} P) \\
&= (C_{in} + C_{out}) P
\end{aligned}$$

Recall that P is a permutation matrix. Suppose $P_{k,j}$ is nonzero. $z_{i,j}/r$ of Formula 2 is $(C_{in} + C_{out})^T P_{:,j}/r = (C_{in}(k) + C_{out}(k)) * r/r = (C_{in}(k) + C_{out}(k))$.

According to the encoding scheme, $C_{in}(k) \in \{0,1\}, C_{out}(k) \in \{1,2\}$ and $C_{in}(k) + C_{out}(k) = 3$ iff $C_{in}(k) = 1$ and $C_{out}(k) = 2$, which indicates that `Lin` and `Lout` have a common center (at the position k). □

PROPOSITION 4.1. *The attacker breaks ASPE3 if and only if he/she breaks ASPE.*[1]

PROOF. *(Sketch)* ***Case 1:*** *Suppose the attacker can break ASPE.*

Since both ASPE and ASPE3 split the data by E_{split} and then encrypt the data by multiplying the plaintext by M^T, in this case, the attacker is capable of recovering C_a and C_b of the graph. Then, the attacker can determine the value of each dimension of the labels, which indicates the existence of centers in the labels. Consequently, the attacker can simply intersect the recovered C_{in} and C_{out} labels to yield the reachability result.

Case 2: *Suppose the attacker can break ASPE3.* It implies the following facts:

1. Given any corresponding ciphertext C_a^e and C_b^e of the encoded label C, the attacker can infer C; and
2. Given any corresponding ciphertext P_a^e and P_b^e of query matrix P, the attacker can recover P.

Next, we show how the attacker recovers each data point p and a query point q and hence the answer of q under ASPE.

Consider an encrypted data point denoted as p_a^e and p_b^e under ASPE. Both p of ASPE and C of ASPE3 are multi-dimensional vectors. By the first fact, the attacker can recover the corresponding plaintext p (data point).

Denote the ciphertext and plaintext of a query point of ASPE to be (q_a^e, q_b^e) and q, respectively. The attacker can extend the ciphertext q_a^e and q_b^e (by repeating q_a^e and q_b^e) to two square matrices, denoted as Q_a^e and Q_b^e. That is, the resulting matrices are simply $Q_a^e = (q_a^e, q_a^e, \ldots, q_a^e)$ and $Q_b^e = (q_b^e, q_b^e, \ldots, q_b^e)$. Then, the dimensions of the encrypted query of ASPE are identical to those of ASPE3. By the second fact, the attacker can determine the plaintext: $Q = (q_1, q_2, \ldots, q_{|W|})$. In addition, $q_1 = q_2 = \ldots = q_{|W|} = q$: If $\exists i, j, q_i \neq q_j$, then $q_{i,a} \neq q_{j,a}$ or $q_{i,b} \neq q_{j,b}$, after splitting using E_{split}. W.l.o.g, we assume $q_{i,a} \neq q_{j,a}$. According to ASPE3, $M_a^{-1} q_{i,a} = q_a^e$. Since M_a^{-1} is an invertible matrix, there is exactly one $q_{i,a}$. According to ASPE3, $M_a^{-1} q_{j,a} = q_a^e$. Hence, $q_{i,a} = q_{j,a}$. It is a contradiction. Therefore, the attacker can recover the corresponding plaintext q (query point). □

PROPOSITION 4.2. *The reachability of the query is protected by APSE3 under KPA.*

PROOF. The proposition is a consequence of Proposition 4.1. By Proposition 4.1, ASPE3 and ASPE offer the same level of security. ASPE is secure against KPA [28]. So is APSE3. Therefore, Proposition 4.2 holds. □

5. SCALABLE APSE3-AWARE 2HOP LABELING – PPTFLABELING

Based on Section 4, it is easy to derive that the time complexities of the ASPE3 encryption (Section 4.2) and private query processing (Section 4.3) are dependent to the dimensions of the vectors and matrices, specifically, $O(|W|^2)$, where $|W|$ is the number of distinct centers. When existing work on `2-hop` construction was directly adopted, our preliminary experiments showed that W can be large for some graphs. In this section, we present an algorithm which can reduce the number of distinct centers up to 58% (from 7% to 58%) for benchmarked real graphs. We first provide the background of topological folding, which is the current state-of-the art of scalable `2-hop` construction (in Section 5.1), and then exploit it to obtain a scalable `2-hop` construction that minimizes $|W|$ for efficient ASPE3 (in Section 5.2).

5.1 Topological folding

In this subsection, we review topological folding [6], which is employed for our algorithm. In a nutshell, the vertices of a DAG G are assigned with their topological level numbers (or simply levels). The main idea is to compute the *topological folding* of G, which is *a series of DAGs $\mathcal{G}$ obtained from a series of foldings*. In each folding step, the vertices of odd (or even) levels are folded. The folding terminates when there is only one level of vertices. Then, the labels are built on the DAGs. We elaborate some crucial details below.

Step 1. Assigning the topological level number. Each vertex v of a *DAG* $G(V, E)$ has one and only one *topological level number* (denoted as $l(v, G)$) defined as follows:

$$l(v,G) = \begin{cases} \max\{(l(u,G)+1) : u \in nb_{in}(v,G)\}, & \text{if } nb_{in}(v,G) \neq \emptyset; \\ 1, & \text{otherwise,} \end{cases} \quad (3)$$

where $nb_{in}(v, G)$ is the in-neighbors of v, defined as $\{u : (u, v) \in E\}$.

The topological level numbers of vertices can be evidently derived from topological ordering of the vertices. As in Formula (3), the topological level number of v is determined by the maximum topological number of its in-neighbors plus one (*i.e.*, $\max\{l(u, G) + 1\}$, where $u \in nb_{in}(v, G)$). The *topological number* of G is defined as $l(G) = \max\{l(v, G) : v \in V\}$.

Step 2. Topological folding. After assigning each vertex a topological level number, the DAG G can be separated into a k-partite DAG where k is $l(G)$, the topological number of G. The ith partite node set is defined as $L_i(G) = \{v : v \in V, l(v, G) = i\}$. Recall that each node set $L_i(G)$ is an independent set [6], *i.e.*, the nodes in $L_i(G)$ are not reachable between each other. *In a folding step, G is folded up by taking away half of the levels.* More specifically, the folding procedure produces a set of DAGs $\mathbb{G} = \{G_1, G_2, \ldots, G_f\}$, where $f = \lfloor log_2 k \rfloor + 1$ and

1. $G_1 = G$; and
2. $G_i = (V_i, E_i)$ is defined with the DAG G_{i-1} obtained after $(i-1)$-th folding, where $1 < i \leq f$ and
   (a) $V_i = \cup_{1 \leq j \leq \lfloor l(G_{i-1})/2 \rfloor} L_{2j}(G_{i-1})$ and
   (b) $E_i = \cup_{1 \leq j \leq \lfloor l(G_{i-1})/2 \rfloor} \{(u, v) : u \in nb_{in}(v_{odd}, G_{i-1}), v \in nb_{out}(v_{odd}, G_{i-1})\}$, where $v_{odd} \in L_{2j-1}(G_{i-1})$, and $nb_{in}(v, G)$ and $nb_{out}(v, G)$ are the in-neighbors and out-neighbors of v in G.

A vertex u to-be-folded at level l may involve in *cross edges*, *i.e.*, the edges of u connecting to another vertex whose level is larger than l+1; or connecting from another vertex whose level is smaller than l-1. When u is folded, the reachability information of its cross-level edges must be preserved.

[1] The term "break" means that the attacker can obtain the plaintext from the ciphertext of an encryption scheme.

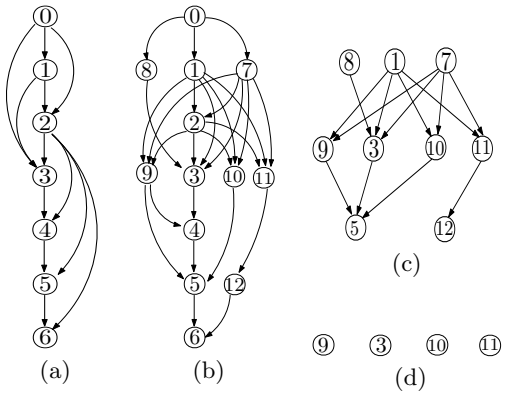


Figure 5: Example of topological folding: (a) A small DAG; (b) Adding dummy nodes to the cross-level edges of the nodes at the odd levels; (c) The DAG after folding the odd level nodes; and (d) The DAG when the folding terminates

Dummy nodes are introduced to encode such information before folding, as follows: Consider a cross edge $(u, v) \in E_{i-1}$ where u is at level j and it is being folded at the i-th folding. A dummy node w is added to $L_{j+1}(G_{i-1})$, and (u, w) and (w, v) are added to E_{i-1}. As a result, in the i-th folding, the reachability information of (u, w) is folded and that of (w, v) is preserved in G_i.

Example 5.1. *Figures. 5a to 5d show some major structures during topological folding. Figure 5a is simply the DAG on the LHS of Figure 2, by rearranging the nodes according to their topological level numbers. Suppose the nodes at the odd levels are being folded. Edges* $(0, 2)$*,* $(0, 3)$*,* $(1, 3)$*,* $(2, 4)$*,* $(2, 5)$*,* $(2, 6)$ *are cross-level edges of the nodes at the odd levels. For vertex 0, dummy vertices 7 and 8 are introduced. Edges* $(0, 7)$*,* $(7, 2)$*,* $(0, 8)$ *and* $(8, 3)$ *are introduced. Then, we remove the edges* $(0, 2)$ *and* $(0, 3)$*. Figure 5b shows the DAG after the cross edges are processed. After one folding step, the DAG is reduced to Figure 5c. The topological folding is recursively applied until the DAG contains only one level, as shown in Figure 5d.*

Step 3. Label generation. Topological folding (Step 2) produces a set of DAGs, $\mathbb{G} = \{G_1, G_2, \ldots, G_f\}$. The topological folding numbers of vertices, defined as $l(v) = \max\{i : v \in V_i\}$, are associated with the vertices. The `2-hop` labels of the DAGs are generated as follows. Initially, v is added to $\texttt{Lin}(v)$. Then, for any $u \in \texttt{Lin}(v)$, labels are added such that $nb_{in}(u, G_{l(u)}) \subset \texttt{Lin}(v)$. This step repeats if there still exists some nodes that can be added to some `Lin`s. The labels `Lout` are generated similarly.

We remark that since the query $\mathsf{Reach}(v, v)$ is trivially true, we assume that the trivial labels are not generated. That is, if v *only* appears in $\texttt{Lin}(v)$ or $\texttt{Lout}(v)$, we can simply remove it. Another remark is that Step 2 introduces dummy nodes due to cross edges. A postprocessing step is introduced to remove them from the `Lin` and `Lout` labels [6].

We present the generated labels of Figure 5a in Table 2.

5.2 Heuristic topological folding

It is clear from Section 5.1 that topological folding was not designed for optimizing $|W|$. In particular, each folding step folds up half of the levels (*e.g.*, the odd levels). While it leads to unprecedented high efficiency and scalability, the levels to be folded do not relate to $|W|$. In this subsection, we present the heuristic algorithm, namely `ppTopoConstruction`, in Algorithm 1 that iteratively greedily selects the level to be folded that leads to a small $|W|$.

Table 2: `2-hop` labeling for the example in Figure 5a

Vertex	Lout	Lin
0	{0,1,2,3}	{0}
1	{1,2,3}	{1}
2	{2,3}	{0,1,2}
3	{3}	{3}
4	{5}	{2,3}
5	{5}	{2,3,5}
6	{}	{2,3,5}

Table 3: `2-hop` labels produced by `ppTopo`, where Vertex 1 is not used in the labels

Vertex	Lout	Lin
0	{0,2,3}	{0}
1	{2,3}	{0}
2	{2}	{2}
3	{3}	{2,3}
4	{5}	{2,3}
5	{5}	{2,3,5}
6	{}	{2,3,5}

Algorithm 1 first initializes `Lin` and `Lout` of each vertex to empty sets (Line 1). Next, it iteratively generates the labels of the nodes level by level via a heuristic function (Lines 3-7). In each iteration (Line 3), Algorithm 1 picks the topological level L_i of the DAG with the maximum weight (Lines 8-11 and 12) defined as follows:

$$weight(L_i) = \frac{\sum_{v \in L_i(G)}(|nb_{in}(v)| + |nb_{out}(v)|)}{|L_i(G)|} \quad (4)$$

The numerator estimates the `Lin` and `Lout` labels that the nodes in $L_i(G)$ to be added as new centers. ($|nb_{in}|$ and $|nb_{out}|$, as opposed to ancestors (`anc`) and descendants (`desc`), are used because of efficiency.) The denominator is favorable to the levels containing a small number of nodes (*i.e.*, new distinct centers). Hence, Formula 4 minimizes $|W|$.

Then, in Lines 5-6, each v at the level L is added into the $\texttt{Lin}(u)$ such that $u \in \mathsf{desc}(v)$, and $\texttt{Lout}(u)$ such that $u \in \mathsf{anc}(v)$, respectively. After adding v to the related `Lin` and `Lout` labels, the vertex v and the incident and outgoing edges of v are removed from the G (Line 7) because their reachability information have been recorded. The iteration terminates when there is no edge left in G.

Example 5.2. *Consider Fig. 5a, since* $weight(L_3) = 6$*,* $weight(L_1) = weight(L_2) = weight(L_5) = weight(L_6) = 3$*,* $weight(L_3) = 4$ *and* $weight(L_7) = 2$*, the vertex* 2 *is picked in the first iteration.* L_3 *is selected. And Vertex* 2 *will be added into* `Lout` *of vertices* $0, 1, 2$ *and* `Lin` *of vertices* $2, 3, 4, 5, 6$*. And the incident and outgoing edges of Vertex* 2 *are removed from* G*. The iteration terminates when there is no edge left. The labels are showed in Table 3. In this example, when compared to Table 2, the* `2-hop` *labels produced by Algorithm 1 require one few center node (i.e., Vertex 1 is not needed in the labels). The value of* $|W|$ *is 4.*

The labels generated will be encoded and encrypted by APSE3 as presented in Section 4.

Algorithm 1 Heuristics Topological Folding for Privacy-Preserving 2-hop (ppTopoConstruction)

Input: DAG G
Output: 2-hop labeling

```
1: initialize Lout and Lin of each vertex to ∅
2: topologically sort G into t topological levels of vertices L(G) = {L_i(G)}
3: while L ← PickLevel(G) do
4:     for c ∈ L do
5:         Lout(u) ← Lout(u) ∪ {c}, for u ∈ anc(c)
6:         Lin(v) ← Lin(v) ∪ {c}, for v ∈ desc(c)
7:         G ← G\c                    ▷ Remove c and incident edges of c
8: function PickLevel(G)
9:     if E(G) == ∅ then return ∅      ▷ Terminating cond.
10:    maintain |nb_in(v)| and |nb_out(v)| for each v
11:    return arg max_{L∈L(G)}(Util(L))
12: function Util(L)
13:    return (Σ_{v∈L}(|nb_in(v)|+|nb_out(v)|)) / |L|      ▷ Formula 4
```

Table 4: Statistics of real-world and synthetic datasets

Graph G	$\|V(G)\|$	$\|E(G)\|$	$\|V(G^{SCC})\|$	$\|E(G^{SCC})\|$
p2p-30	36,682	88,328	28,193	56,622
p2p-31	62,586	147,892	48,438	96,976
Cit-HepPh	34,546	421,578	21,608	281,030
Amazon0302	262,111	1,234,877	6,594	30,785
Wiki-Vote	7,115	103,689	5,816	64,233
WikiTalk	2,394,385	5,021,410	2,281,879	2,311,570
LiveJournal	4,847,571	68,993,773	971,232	1,079,293
web-BerkStan	685,230	7,600,595	109,406	583,771

6. EXPERIMENTAL EVALUATION

In this section, we present a detailed experimental evaluation to investigate the performance of our techniques, mui-2-hop and ppTopo, on real world datasets.

Experimental setup. We obtained the implementations of existing works from their authors [32] and [30] for experimental comparisons. We implemented our method in JAVA. The timings of our experiments were measured by running our implementations on a machine with a 2.93GHz CPU and 8GB memory. We remark that the $\mathcal{SP}$ is often equipped with powerful machines. For simplicity, we use the same machine configuration for the $\mathcal{SP}$ and the client. We benchmarked our implementations with 8 real-world datasets[2] including both small and massive graphs. Some important statistics of the datasets are reported in Table 4.

6.1 Performance of existing work

In this subsection, we first report the performance of existing work. We show the statistics and performance of the state-of-art approach of mui-2-hop [30].

Table 5 reports the characteristics of mui-2-hop. It shows that the number of trues in $TC(G^{SCC})$ ($|TC(G^{SCC})|$) are around 7% of $|V(G^{SCC})|^2$. Thus, this is favorable to mui-2-hop as it is designed for sparse graphs. We also note that the sizes of mui-2-hop are significantly smaller than $|V(G^{SCC})|^2$ (at most 3%). However, since mui-2-hop builds the transitive closure of the graph in its construction procedure, the implementation of mui-2-hop cannot run on networks larger than those listed in Table 5 because the corresponding size of the transitive closure exceeds the size of memory.

[2]The datasets are available at http://snap.stanford.edu/data/.

Table 5: Statistics of mui-2-hop

Graph G	|mui-2-hop|	$\|TC(G^{SCC})\|$	$\|V(G^{SCC})\|^2$	Cons. time
p2p-30	2.10M	7.02M	794.85M	6min12s
p2p-31	5.69M	18.17M	2.35G	28min12s
Cit-HepPh	17.73M	84.25M	466.91M	19min20s
Amazon0302	367.10K	28.075K	43.48M	15.21s
Wiki-Vote	927.44K	3.93M	33.83M	20.503s

Query performance of mui-2-hop. For the reachability queries, we randomly selected 1,000 pairs of vertices as queries. The time cost at the $\mathcal{SP}$ side for running all the queries are small, in particular, at most 0.18s for real data graphs.

We remark that while the query time is short, as discussed in Section 3, it is less secure than our proposed method.

6.2 Performance of ppTopo

In this part, we show the statistics and performance for ppTopo.

Distinct centers. Table 6 lists the number of distinct centers $|W|$ of the proposed algorithm, ppTopoConstruction. Table 6 shows that the distinct centers of TF-labeling are almost one fifth of that of $|V(G^{SCC})|$ for graphs with small degree, *e.g.*, p2p-30. For massive graphs, such as LiverJournal, the size of the distinct centers is around 30% which is much less than that of the nodes.

Encryption performance of ppTopo. Table 7 reports the encryption performance with the two addition encryption algorithms. The first one is a partially homomorphic encryption called Paillier [21] and the second one is an efficient encryption that supports partial matrix additions and multiplications called CGBE [10]. These encryption approaches can work on graphs. We set the time limit of this experiment as 48 hours. We cut the corresponding experiment if it cannot finish in time and we use *DNF* in the results to denote that. As the results shown in Table 7, Paillier is the most time-consuming approach because additive homomorphic encryption is known to be inefficient. More specifically, Paillier cannot finish encrypting on some massive graphs, such as WikiTalk and LiveJournal, in limited time. When compared to Paillier, the encryption time cost of CGBE is almost 25-30 times faster than that of Paillier. Our results show that the encryption time of our approach of ASPE3 is even about 5 times faster than that of CGBE. The reason is that, both Paillier and CGBE consider each individual node separately and apply multiple operations on each node to achieve the security requirement. Our approach outperforms them as we consider a d-dimensional vector at a time with only several matrix operations on them.

Query performance of ppTopo. Regarding the query performance, we randomly selected 1,000 pairs of vertices as our queries. We measured the time cost of query processing at the $\mathcal{SP}$ side and the decryption cost at the client side to study the query performance. The results are shown in Table 8 and Table 9, respectively. Table 8 shows that ASPE3 is about one order of magnitude slower than Paillier at the $\mathcal{SP}$. The reason is that Paillier requires multiplications of $|W|$ encrypted centers, which could be faster than the matrix operations of ASPE3 *when the graphs are small.* Table 9 shows that the decryption performance of ASPE3 is 3 orders of magnitude faster than Paillier, because ASPE3 re-

Table 6: Distinct centers of ppTopo

Graph G	$\|V(G^{SCC})\|$	ppTopo	Cons. time
p2p-30	28,193	3,683	23.127s
p2p-31	48,438	7,112	22.988s
Cit-HepPh	21,608	13,039	22.346s
Amazon0302	6,594	1,273	834ms
Wiki-Vote	5,816	1,017	601ms
WikiTalk	2,281,879	31,161	7hours57min
LiveJournal	971,232	38,483	1hour23min
web-BerkStan	109,406	30,253	5min51s

Table 7: Encryption performance of TF-labeling

Graph G	ASPE3	Paillier	CGBE
p2p-30	33.17s	1hour37min	3min28s
p2p-31	1min51s	4hours52min	11min55s
Cit-HepPh	1min30s	4hours2min	9min43s
Amazon0302	2.83s	8min32s	16.74s
Wiki-Vote	1.95s	5min0s	11.78s
WikiTalk	6hours21min	*DNF*	41hours29min
LiveJournal	3hours13min	*DNF*	21hours21min
web-BerkStan	18min37s	47hours2min	1hour52min

quires essentially 1 division of two large numbers, while Paillier requires essentially 1 exponentiation modulo for each of $O(|W|)$ encrypted messages. Importantly, ASPE3 finishes on all graphs while the Paillier encryption did not finish when graphs are massive (*e.g.*, WikiTalk and LiverJournal). Also, the performance bottleneck is the decryption at the client side as the clients may not be always equipped with powerful machines. Therefore, ASPE3 is more competitive than Paillier for private reachability queries as it costs much less at the client side. Similar arguments can be applied to CGBE, as its decryption times are almost always significantly slower than ASPE3's.

Effectiveness of Algorithm 1. Table 10 and Table 11 report the encryption and decryption performance with and without the heuristic algorithm optimizations in Algorithm 1. The algorithm without the heuristic algorithm is the original TF-labeling algorithm. Table 10 shows that Algorithm 1 can reduce 60% distinct centers for some graphs and 10% $\sim$ 40% of encryption times, *e.g.*, p2p-30$\sim$p2p-31. Such optimizations perform well especially on massive graphs because it can reduce the size of distinct centers which will reduce the size of the vectors need to be considered. For the decryption performance at the client side, Table 11 shows that the Algorithm 1 leads to similar improvement on query performance. It is obvious that the smaller the number of distinct centers, the better the decryption performance. Moreover, the heuristic algorithm can reduce 60% query time at most, *e.g.*, Wiki-Vote.

Table 8: Total times of 1,000 privacy intersections at the $\mathcal{SP}$ side

Graph G	ASPE3	Paillier	CGBE
p2p-30	1.18s	152.2ms	11.97s
p2p-31	2.25s	266.6ms	23.16s
Cit-HepPh	4.26s	497.1ms	42.72s
Amazon0302	406ms	57.8ms	4.16s
Wiki-Vote	334ms	38ms	3.23s
WikiTalk	11.007s	*DNF*	2min2s
LiveJournal	14.180s	*DNF*	2min29s
web-BerkStan	11.072s	1.37s	1min55s

Table 9: Time cost of 1,000 decryptions at the client side

Graph G	ASPE3	Paillier	CGBE
p2p-30	18.9ms	3min27s	55.5ms
p2p-31	34.6ms	6min2ss	56.4ms
Cit-HepPh	61.1ms	11min16s	56.7ms
Amazon0302	2.7ms	1min18s	59.3ms
Wiki-Vote	2.6ms	51.68s	55.8ms
WikiTalk	129.5ms	*DNF*	133ms
LiveJournal	174.4ms	*DNF*	153ms
web-BerkStan	128.7ms	30min19s	116ms

Table 10: Encryption performance of opt. in ASPE3

Graph	none-optimized		optimized	
	$\|W\|$	Encrypt. time	$\|W\|$	Encrypt. time
p2p-30	4,721	42.84s	3,683	33.17s
p2p-31	8,309	2min10s	7,112	1min51s
Cit-HepPh	18,158	2min14s	13,039	1min30s
Amazon0302	844	1.84s	1,273	2.83s
Wiki-Vote	2,467	4.98s	1,017	1.95s
WikiTalk	56,934	14hours2min	31,161	6hours21min
LiveJournal	52,649	5hours30min	38,483	3hours13min
web-BerkStan	37,155	26min56s	30,253	18min37s

7. RELATED WORK

In addition to the privacy-preserving 2-hop labelings ([32] and [30]) discussed in Section 3, there have been a few related research. (Due to space restrictions, we omit the discussions on other privacy-preserving works that are not directly relevant to graph data.)

There have been related works on security of graph queries. One stream of works addresses the authenticity of subgraph data or query answers [19, 18, 17, 11], where the clients verify the data/answers returned by an $\mathcal{SP}$ are not tampered with.

Regarding the confidentiality of graph queries or data, He et al. [13] analyze the reachability of vertices and Gao et al. [12] propose neighborhood-privacy protected shortest distance. However, their privacy targets are different from our work. Mouratidis et al. [20] determine the shortest path of the query nodes with no information leakage by using PIR [8]. However, the high computational cost of PIR is still a concern. Chang et al. [5] consider privacy model based on k-automorphism for subgraph matching. Cao et al. [4] and Fan et al [10] consider subgraph queries but not reachability queries. Cao te al. [3] investigated patterns queries on encrypted trees.

8. CONCLUSIONS

This paper studies privacy-preserving query services for reachability queries under the paradigm of data outsourcing. The paper is the first work that can construct privacy-preserving index for large networks. Our main technique relies on adopting the concept of topological folding for scalable index construction. Private reachability query processing is done by private intersections. We propose a novel asymmetric scalar product encryption modulo 3 (ASPE3) for efficient private intersection. We have experimentally compared with existing work and showed that our proposed techniques are efficient and effective on large networks, and APSE3 is efficient when compared related encryptions including Paillier and CGBE. As for future work, we are investigating the extension on label-constraint reachability queries.

Table 11: Decryption performance of opt. at client side

Graph	none-optimized		optimized	
	$\|W\|$	Query time	$\|W\|$	Query time
p2p-30	4,721	20.3ms	3,683	18.9ms
p2p-31	8,309	33.3ms	7,112	34.6ms
Cit-HepPh	18,158	112.6ms	13,039	61.1ms
Amazon0302	844	2.2ms	1,273	2.7ms
Wiki-Vote	2,467	5.8ms	1,017	2.6ms
WikiTalk	56,934	350.0ms	31,161	129.5ms
LiveJournal	52,649	298.1ms	38,483	174.4ms
web-BerkStan	37,155	182.3ms	30,253	128.7ms

Acknowledgements. We thank Zhe Fan for his insightful comments on the earlier versions of this work. J. Jiang, P. Yi, B. Choi and Z. Zhang are partially supported by HKRGC GRFs 12201315 and 12232716, and HKBU FRG2/13-14/073.

9. REFERENCES

[1] R. Bramandia, B. Choi, and W. K. Ng. Incremental maintenance of 2-hop labeling of large graphs. *TKDE*, 22(5):682–698, 2010.

[2] J. Cai and C. K. Poon. Path-hop: efficiently indexing large graphs for reachability queries. In *CIKM*, pages 119–128, 2010.

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

[4] 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.

[5] Z. Chang, L. Zou, and F. Li. Privacy preserving subgraph matching on large graphs in cloud. In *SIGMOD*, pages 199–213, 2016.

[6] J. Cheng, S. Huang, H. Wu, and A. Fu. Tf-label: a topological-folding labeling scheme for reachability querying in a large graph. In *SIGMOD*, pages 193–204, 2013.

[7] J. Cheng, J. X. Yu, X. Lin, H. Wang, and P. S. Yu. Fast computing reachability labelings for large graphs with high compression rate. In *EDBT*, pages 193–204, 2008.

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

[9] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. In *SODA*, 2002.

[10] Z. Fan, B. Choi, Q. Chen, J. Xu, H. Hu, and S. Bhowmick. Structure-preserving subgraph query services. *TKDE*, 27(8):2275–2290, 2015.

[11] Z. Fan, Y. Peng, B. Choi, J. Xu, and S. Bhowmick. Towards efficient authenticated subgraph query service in outsourced graph databases. *TSC*, 7(4):696–713, 2013.

[12] 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.

[13] X. He, J. Vaidya, B. Shafiq, N. Adam, and X. Lin. Reachability analysis in privacy-preserving perturbed graphs. In *WI-IAT*, pages 691–694, 2010.

[14] R. Jin, N. Ruan, S. Dey, and J. Y. Xu. Scarab: scaling reachability computation on large graphs. In *SIGMOD*, pages 169–180, 2012.

[15] 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.

[16] R. Jin, Y. Xiang, N. Ruan, and D. Fuhry. 3-hop: a high-compression indexing scheme for reachability query. In *SIGMOD*, pages 813–826, 2009.

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

[18] A. Kundu, M. J. Atallah, and E. Bertino. Leakage-free redactable signatures. In *CODASPY*, pages 307–316, 2012.

[19] A. Kundu and E. Bertino. How to authenticate graphs without leaking. In *EDBT*, pages 609–620, 2010.

[20] K. Mouratidis and M. L. Yiu. Shortest path computation with no information leakage. *PVLDB*, 5(8):692–703, 2012.

[21] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[22] R. Schenkel, A. Theobald, and G. Weikum. HOPI: An efficient connection index for complex XML document collections. In *EDBT*, pages 237–255, 2004.

[23] S. Seufert, A. Anand, S. Bedathur, and G. Weikum. Ferrari: Flexible and efficient reachability range assignment for graph indexing. In *ICDE*, pages 1009–1020, 2013.

[24] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput*, 1(2):146–160, 1972.

[25] S. Trissl and U. Leser. Fast and practical indexing and querying of very large graphs. In *SIGMOD*, pages 845–856, 2007.

[26] S. J. van Schaik and O. de Moor. A memory efficient reachability data structure through bit vector compression. In *SIGMOD*, pages 913–924, 2011.

[27] H. Wei, J. X. Yu, C. Lu, and R. Jin. Reachability querying: An independent permutation labeling approach. *PVLDB*, 7(12):1191–1202, 2014.

[28] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis. Secure knn computation on encrypted databases. In *SIGMOD*, pages 139–152, 2009.

[29] K. Xu, L. Zou, J. X. Yu, L. Chen, Y. Xiao, and D. Zhao. Answering label-constraint reachability in large graphs. In *CIKM*, pages 1595–1600, 2011.

[30] P. Yi, Z. Fan, and S. Yin. Privacy-preserving reachability query services for sparse graphs. In *ICDE Workshops (GDM)*, pages 32–35, 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] S. Yin, Z. Fan, P. Yi, B. Choi, J. Xu, and S. Zhou. Privacy-preserving reachability query services. In *DASFAA*, pages 203–219, 2014.