This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2880981, IEEE Transactions on Dependable and Secure Computing

Enabling Privacy-Preserving Shortest Distance Queries on Encrypted Graph Data

Chang Liu *Student Member, IEEE*, Liehuang Zhu *Member, IEEE*, Xiangjian He *Senior Member, IEEE*, Jinjun Chen *Senior Member, IEEE*

Abstract—When coming to perform shortest distance queries on encrypted graph data outsourced in external storage infrastructure such as cloud, a significant challenge is how to compute the shortest distance in an accurate, efficient and secure way. This issue is addressed by a recent work, which makes use of somewhat homomorphic encryption (SWHE) to encrypt distance values output by a 2-hop cover labeling (2HCL) scheme. However, it may import large errors and even yield negative results. Besides, SWHE would be too inefficient for normal clients. In this paper, we propose GENOA, a novel Graph ENcryption scheme for shOrtest distAnce queries. GENOA employs only efficient symmetric-key primitives while significantly enhances the accuracy compared to the prior work. As a reasonable trade-off, it additionally reveals the order information among queried distance values in the 2HCL index. We theoretically prove the accuracy and security of GENOA under rigorous cryptographic model. Detailed experiments on eight real-world graphs demonstrate that GENOA is efficient and can produce almost exact results.

Index Terms—Graph encryption, shortest distance, secure data outsourcing, 2-hop cover labeling.

1 Introduction

Data outsourcing has become one of the most important applications in cloud computing, as it significantly reduces clients' costs on data storage and management [26], [28]. However, the privacy of outsourced data might be compromised as data owners have lost the physical control over data [15]. To prevent privacy leakage, data should be encrypted before outsourcing, especially when the storage provider is untrusted. Traditional cryptographic encryption tools directly destroy the data usability because encrypted data is difficult to be queried. To address this problem, a series of searchable encryption schemes (see a survey [5]) have been proposed, which enable the storage provider to perform keyword searches on encrypted textual data without leaking privacy information. Besides textual data, graph data is also an important data type in many applications, such as social networks, road networks and biological networks. However, few solutions have been proposed to enable privacy-preserving queries on encrypted graph data, which limits the applications of graph data outsourcing.

As one of the most fundamental graph queries, shortest distance query has wide applications such as closeness testing in social networks, route planning in road networks, causal reasoning in biological networks, etc. In this paper, we investigate how to perform shortest distance queries on encrypted graph data. That is, a data owner properly encrypts a graph and outsources it to a storage provider while the latter can still answer shortest distance queries without learning useful information about the graph and queries.

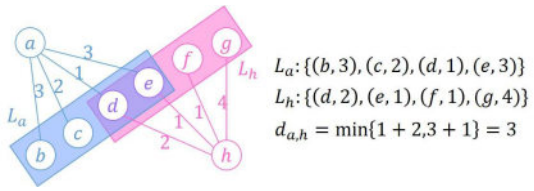


Fig. 1. Computing the shortest distance using 2-hop cover labeling scheme: an illustration.

A series of 2-hop cover labeling (2HCL) schemes [27], [2], [1], [14], [3], [24], [10] have been proposed to compute shortest distances in plaintext setting. Such schemes precompute an index so that subsequent shortest distance queries can be answered efficiently. Specifically, given a graph G, for each vertex $v \in G$ the 2HCL scheme properly collects a set $S(v)$ of candidate vertices such that for each vertex pair (s, t) there is at least one vertex u satisfying that: (1) $u \in S(s)$, (2) $u \in S(t)$, and (3) u is on a shortest (or an approximate shortest) path between s and t. Each vertex $v \in G$ is labeled with $L_v = \{(u, d_{u,v})\}_{u \in S(v)}$, in which u is a vertex identifier and $d_{u,v}$ is the shortest distance between u and v. The 2HCL index is the collection of all labels, namely $\{L_v\}_{v \in G}$. Given an index, answering the shortest distance between vertices s and t is simply computing $\min\{d_{u,s} + d_{u,t} | (u, d_{u,s}) \in L_s, (u, d_{u,t}) \in L_t\}$.

- *C. Liu is with the Faculty of Engineering and Information Technology, University of Technology Sydney, NSW 2007, Australia, and also with Beijing Engineering Research Center of Massive Language Information Processing and Cloud Computing Application, School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China.*
  E-mail: changliu.bit@gmail.com
- *L. Zhu is with Beijing Engineering Research Center of Massive Language Information Processing and Cloud Computing Application, School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China.*
  E-mail: liehuangz@bit.edu.cn
- *X. He is with Faculty of Engineering and Information Technology, University of Technology Sydney, NSW 2007, Australia.*
  E-mail: xiangjian.he@uts.edu.au
- *J. Chen is with Swinburne Data Science Research Institute, Swinburne University of Technology, Melbourne, VIC 3122, Australia.*
  E-mail: jinjun.chen@gmail.com

Fig. 1 shows an example that how we obtain the shortest distance between vertices a and h.

Without encryption, the 2HCL index will reveal much information to the storage provider. Firstly, as all the data is unencrypted, the index directly reveals vertex identifiers and distance values. Thus the storage provider is able to compute the shortest distance between any two vertices in the graph. Secondly, the number of labels in the index reveals the number of vertices in the graph. Thirdly, the index reveals the length of each label. And fourthly, during queries, the storage provider learns which vertices are being queried by the client. The above information may compromise the privacy of the client. Therefore, our essential goal in this paper is to appropriately encrypt the 2HCL index so that the storage provider can still use it to answer shortest distance queries.

The only existing approach fully addressing the above issue appears in a recent work by Meng et al. [20]. However, their proposed GRECS scheme[1] suffers accuracy and efficiency drawbacks: (1) It may import large errors even using an exact-2HCL scheme. As a result, their scheme even yields negative distances in some cases. (2) It makes use of somewhat homomorphic encryption (SWHE) scheme (e.g., the BGN construction in [4]) which simultaneously supports multiple homomorphic additions and at least one homomorphic multiplication. Compared with symmetric-key cryptographic primitives such as pseudo-random functions (PRFs), the SWHE scheme requires much more computation. Hence, normal clients with limited computation resources might be not able to use the scheme.

In this paper, we propose a new 2HCL-based graph encryption scheme. Both our scheme and the GRECS scheme in [20] are built on top of a 2HCL scheme, therefore the accuracy of the two graph encryption schemes are theorectically bounded by the accuracy of the 2HCL scheme. The GRECS scheme additionally imports remarkable errors onto the 2HCL scheme. Our scheme achieves higher accuracy at the cost of leaking the order information among part of the distance values in the 2HCL index. As a result, our scheme almost maintains the accuracy of the 2HCL scheme. Moreover, our scheme employs only symmetric-key cryptographic primitives so that it is more efficient than the GRECS scheme.

We summarize our contributions as follows:

- We address the issue of privacy-preserving shortest distance query on encrypted graph data by defining the framework of 2HCL-based graph encryption. In particular, we present an accuracy definition which is parametrized by the accuracy bounds of the underlying 2HCL scheme and the additional errors imported by the specific design.
- We propose an order-preserving encryption scheme OPE and show how to execute approximate shortest distances on OPE -encrypted distance values.
- Using OPE as a building block, we propose GENOA, a Graph ENcryption scheme for shOrtest distAnce queries. GENOA is (1) general - supporting any 2HCL schemes, (2) accurate - almost preserving the accuracy of the underlying 2HCL scheme, (3) secure - revealing limited information to the storage provider, and (4) efficient - requiring low computation and space overheads.
- We implement GENOA and conduct detailed performance evaluations on a number of real-world graphs.

The rest of this paper is organized as follows. We review related work in Section 2. In Section 3 we present technical preliminaries. In Section 4 we introduce how to perform distance computations on encrypted values. We provide detailed design of GENOA in Section 5 and its accuracy and security proofs in Section 6. We explain our performance evaluation results in Section 7. Finally, we conclude the paper and discuss future directions in Section 8.

2 Related Work

The study of querying encrypted data is first addressed to enable keyword search on textual data [16], [13], [8], [7], [25], [22]. The proposed schemes are commonly known as Searchable Symmetric Encryption (SSE). A milestone of SSE is the work by Curtmola et al [11], as they gave a sound security definition for SSE. Their security model was then generalized by Chase and Kamara in [9] such that it adapts to arbitrary structured data type, including graphs. In detail, the security model uses leakage functions to capture what is leaked during the execution of the system. A scheme is considered secure if there is a simulator, who is given the output of leakage functions, can simulate a runtime environment which is indistinguishable from the real runtime environment. As this security model has been widely accepted in the literature, we use it in this paper.

Querying outsourced and encrypted graphs has drawn more and more attention recently. The work by Chase and Kamara [9] studied neighbor query which answers all the neighbor vertices of the queried vertex, adjacency query which returns 1 (or 0) if two queried vertices are adjacent (or not), and a special subgraph query which returns a subgraph containing all vertices either linked to or linked by a set of queried vertices. In [29], Yin et al. studied privacy-preserving reachability query. In [6], Cao et al. studied subgraph query over an encrypted graph database, i.e., returning all graphs in the database that are supergraphs of a queried graph. Shortest distance query was recently studied by Meng et al. in [20]. Their scheme is also based on a 2HCL scheme. It uses somewhat homomorphic encryption to enable distance computations. However, SWHE does not support the "min" operations so that the scheme is not able to identify which is the shortest distance if there is more than one candidate path, as the example shown in Fig. 1. Therefore, the scheme loosens the accuracy to obtain approximate shortest distances as we have discussed in Section 4

There are a number of graph privacy works using different security models than ours. Some of them [30], [19], [12] protect neighborhood information using anonymization techniques, but do not consider other partial information of the graph as well as the query privacy. Some approaches [17], [21] hide the query information but completely leak the information of the graph itself. Consequently, these works are not fully addressing our data outsourcing scenario.

1. Meng et al. [20] propose three graph encryption schemes in a progressive way, which are respectively named $\mathsf{GrephEnc}_1$, $\mathsf{GrephEnc}_2$ and $\mathsf{GrephEnc}_3$ in their paper. In this paper, we discuss only the $\mathsf{GrephEnc}_3$ scheme, and we denote it as GRECS for brevity.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2880981, IEEE Transactions on Dependable and Secure Computing

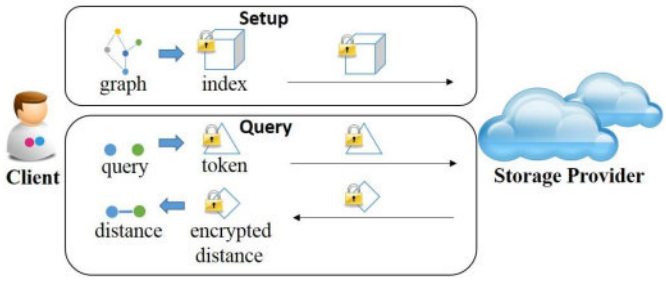


Fig. 2. The system model of graph encryption: an illustration

3 PRELIMINARIES

3.1 Graph Encryption Framework and System Model

We define the framework of 2HCL-based graph encryption as follows:

Definition 1 (2HCL-based Graph Encryption). A 2HCL-based graph encryption scheme Π is a collection of five algorithms $\Pi = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Dist}, \mathsf{Dec})$:

- $(K, \{L_v\}_{v\in G}) \leftarrow \mathsf{Init}(\lambda, G)$: is an algorithm that takes as input a security parameter λ and a graph G. It outputs a secret key K and a 2HCL index $\{L_v\}_{v\in G}$.
- $I \leftarrow \mathsf{Enc}(K, \{L_v\}_{v\in G})$: is an algorithm that takes as input a secret key K and a 2HCL index $\{L_v\}_{v\in G}$. It outputs an encrypted index I.
- $\tau_{s,t} \leftarrow \mathsf{Token}(K, s, t)$: is an algorithm that takes as input a secret key K and two vertices s, t. It outputs a query token $\tau_{s,t}$.
- $D_{s,t} \leftarrow \mathsf{Dist}(I, \tau_{s,t})$: is an algorithm that takes as input an index I and a token $\tau_{s,t}$. It outputs an encrypted distance $D_{s,t}$.
- $d_{s,t} \leftarrow \mathsf{Dec}(K, D_{s,t})$: is an algorithm that takes as input a secret key K and an encrypted distance $D_{s,t}$. It outputs a decrypted distance $d_{s,t}$.

We consider a graph encryption system involving a client $\mathcal{C}$ who owns a graph G and a storage provider $\mathcal{P}$ who stores encrypted data outsourced by $\mathcal{C}$. The system model includes two protocols Setup and Query, defined as follows:

- Setup: $\mathcal{C}$ firstly obtains $(K, \{L_v\}_{v\in G}) \leftarrow \mathsf{Init}(\lambda, G)$ and then generates an index $I \leftarrow \mathsf{Enc}(K, \{L_v\}_{v\in G})$. Finally $\mathcal{C}$ sends I to $\mathcal{P}$. $\mathcal{P}$ receives and stores the index.
- Query: To query the shortest distance between vertices $s, t \in G$, $\mathcal{C}$ obtains a token $\tau_{s,t} \leftarrow \mathsf{Token}(K, s, t)$ and sends it to $\mathcal{P}$. $\mathcal{P}$ then computes $D_{s,t} \leftarrow \mathsf{Dist}(I, \tau_{s,t})$ and returns it to $\mathcal{C}$. Finally, $\mathcal{C}$ decrypts the distance as $d_{s,t} \leftarrow \mathsf{Dec}(K, D_{s,t})$.

We show an illustration of the system model in Fig. 2.

3.2 Accuracy and Security

Different 2HCL schemes have different strategies for generating labels. Some [1], [14], [3] can answer exact shortest distances while some [27], [2], [24], [10] only answer approximate shortest distances. We call the former exact-2HCL schemes and the latter approximate-2HCL schemes. The accuracy of a 2HCL scheme is described by a lower bound $\mathcal{B}_{lower}$ and an upper bound $\mathcal{B}_{upper}$. For example, let $d_{s,t}$ be the exact distance between arbitrary vertices s, t in the graph, the approximate-2HCL scheme by Agarwal et al. [2] has $\mathcal{B}_{lower}(s,t) = d_{s,t}$ and $\mathcal{B}_{upper}(s,t) = 2 \cdot d_{s,t}$. Obviously, any exact-2HCL scheme has $\mathcal{B}_{lower}(s,t) = \mathcal{B}_{upper}(s,t) = d_{s,t}$.

In our framework a graph encryption scheme is built upon a 2HCL scheme, thus the accuracy of the graph encryption scheme depends on the accuracy bounds of the underlying 2HCL scheme. Besides, the graph encryption scheme may import additional errors due to specific design. Thus we use two parameters α, β to denote how much the graph encryption scheme breaks the lower bound and upper bound of the underlying 2HCL scheme, respectively. Our accuracy definition is as follows:

Definition 2 (Accuracy). Let $\Pi = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Dist}, \mathsf{Dec})$ be a 2HCL-based graph encryption scheme. Given a graph G and two veritces $s, t \in G$, let $\mathcal{B}_{lower}$ and $\mathcal{B}_{upper}$ be the accuracy lower and upper bound of the underlying 2HCL scheme. Assume λ is a security parameter. We say Π achieves $(\mathcal{B}_{lower}, \mathcal{B}_{upper}, \alpha, \beta)$-accuracy if for all graph G, $s, t \in G$,

$$\Pr[\mathcal{B}_{lower}(s,t) - \alpha \le d_{s,t} \le \mathcal{B}_{upper}(s,t) + \beta] \ge 1 - \mathrm{negl}(\lambda)$$

where negl is a negligible function and $d_{s,t}$ is computed using the following sequence: $(K, \{L_v\}_{v\in G}) \leftarrow \mathsf{Init}(\lambda, G)$, $I \leftarrow \mathsf{Enc}(K, \{L_v\}_{v\in G})$, $\tau_{s,t} \leftarrow \mathsf{Token}(K, s, t)$, $D_{s,t} \leftarrow \mathsf{Dist}(I, \tau_{s,t})$, $d_{s,t} \leftarrow \mathsf{Dec}(K, D_{s,t})$.

We define the security of a 2HCL-based graph encryption scheme using the standard simulation-based model which has been widely adopted by secure computation works [11], [9], [20], etc. We consider an adversary[2] who has access to all encrypted data and queries from $\mathcal{C}$. The adversary behaves honest-but-curious, namely, the adversary honestly performs all operations required by our system model but curiously infers private information from $\mathcal{C}$'s data and queries.

Definition 3 (Security). Let $\Pi = (\mathsf{Init}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Dist}, \mathsf{Dec})$ be a 2HCL-based graph encryption scheme. Let $\mathcal{L}_{setup}$ and $\mathcal{L}_{query}$ be leakage functions which capture the leaked information during Setup and Query, respectively. Let $\mathcal{A}$ be an adversary and $\mathcal{S}$ be a simulator. Assume λ is a security parameter. We define two games as follows:

- $\mathbf{Real}_{\mathcal{A}}(\lambda)$: $\mathcal{A}$ chooses a graph G. The game then generates $(K, \{L_v\}_{v\in G}) \leftarrow \mathsf{Init}(\lambda, G)$ and computes $I \leftarrow \mathsf{Enc}(K, \{L_v\}_{v\in G})$ and gives I to $\mathcal{A}$. Then $\mathcal{A}$ outputs a polynomial number of shortest distance queries: $\mathbf{q} = \{(s_1, t_1), ..., (s_q, t_q)\}$. For each query $(s,t) \in \mathbf{q}$, the game generates a token $\tau_{s,t} \leftarrow \mathsf{Token}(K, s, t)$ and gives it to $\mathcal{A}$. $\mathcal{A}$ obtains $D_{s,t} \leftarrow \mathsf{Dist}(I, \tau_{s,t})$. Finally $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$ which is also the output of the game.
- $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$: $\mathcal{A}$ chooses a graph G. Given $\mathcal{L}_{setup}(G)$, $\mathcal{S}$ simulates an index I^* and sends it to $\mathcal{A}$. Then $\mathcal{A}$ outputs a polynomial number of shortest distance queries: $\mathbf{q} = \{(s_1, t_1), ..., (s_q, t_q)\}$. Given $\mathcal{L}_{query}(G, \mathbf{q})$, $\mathcal{S}$ simulates and sends to $\mathcal{A}$ a token $\tau^*_{s,t}$ for each query $(s,t) \in \mathbf{q}$. $\mathcal{A}$ obtains $D^*_{s,t} \leftarrow \mathsf{Dist}(I^*, \tau^*_{s,t})$. Finally $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$ which is also the output of the game.

2. In practice, $\mathcal{P}$ is the potential adversary

TABLE 1
Symbols

Symbol	Description
λ	A security parameter
K	A secret key
G	A graph
u, v, s, t	Identifiers of vertices in the graph
U, V	The encoded u, v
$d_{u,v}$	The shortest distance from u to v
$D_{u,v}$	The encrypted shortest distance from u to v
L_u, L_v, L_s, L_t	The labels of vertex u, v, s, t in the 2HCL index
I	A graph encryption index
f, g	Pseudo-Random Functions

We say Π achieves $(\mathcal{L}_{setup}, \mathcal{L}_{query})$-security if for any probability polynomial time (PPT) adversary $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ such that

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

where negl is a negligible function. The security model guarantees that a secure graph encryption scheme does not leak more than the output of leakage functions. We present the specific leakage functions of our scheme in Section 6.

3.3 Tools and Notations

2-hop cover labeling. A 2HCL scheme LAB consists of two algorithms: the labeling algorithm Label and the shortest distance calculation algorithm Dist. In this paper we only use the labeling algorithm. Label takes as input a graph G, it outputs a 2HCL index $\{L_v\}_{v \in G}$. We call L_v the label of vertex v, and the length of L_v is the number of $(u, d_{u,v})$ pairs in L_v.

Cryptographic primitives. We use secure Symmetric-key Encryption (SKE) and Pseudo-Random Function (PRF). As SKE and PRF are common cryptographic primitives, we omit their formal security definition in this paper.

Dictionary. Our index is built as a dictionary data structure. A dictionary I stores $(key, value)$ pairs such that given a key, the corresponding $value$ can be returned in $O(1)$ time. We use $I[key]$ to denote the $value$ labeled by key in I. If key does not exist in I, then $I[key]$ returns $\perp$.

Notations. In the rest of the paper, we use the notation "$scheme.algorithm$" to denote calling the algorithm $algorithm$ of scheme $scheme$. We use "$y \leftarrow algorithm(x)$" for setting y as the output of algorithm $algorithm$ on input x. The notation "$a||b$" means a concatenation of string a and b. "$x \xleftarrow{\$} X$" means randomly selecting an element x from set X.

3.4 Symbols

In the rest of the paper, although we offer definitions before using symbols, we list general symbols and their descriptions in Table 1 for ease of reading. Those symbols which are associated with context are not listed here.

4 Shortest Distance Computation on Encrypted Values

In this section we introduce how to encrypt distance values in the 2HCL index so that shortest distance computations can be performed on encrypted values.

As shown by the example in Fig. 1, the essential operations of a shortest distance computation include add operations ("+") and a minimum search ("min"). Ideally, an encryption required here should be not only additively-homomorphic but also order-preserving. However, as far as we know, such an encryption is still missing from the current literature.

To address this problem, the work by Meng et al. [20] leverages a somewhat homomorphic encryption (SWHE=(Enc,Dec,Eval)) to support "+" operations over encrypted distance values. Since the encryption is not order-preserving, the subsequent "min" operation cannot be calculated if there are more than one candidate result. Their idea is to derive an approximate result from all candidate results as follows: $\mathcal{C}$ encrypts each distance value d in the 2HCL index as $D \leftarrow \mathsf{SWHE.Enc}(K, 2^{d_{max}-d})$, in which d_{max} is the maximum distance value. Given a query (s, t), $\mathcal{P}$ returns $c = \Sigma_{(u,D_{u,s}) \in L_s, (u,D_{u,t}) \in L_t} D_{u,s} \times D_{u,t}$, where "$\times$" and "$\Sigma$" are calculated by $\mathsf{SWHE.Eval}$. Finally, $\mathcal{C}$ computes $d_{s,t} = 2d_{max} - \log \mathsf{SWHE.Dec}(K, c)$. For the example in Fig. 1, $\mathcal{C}$ obtains $d_{a,h} = 2d_{max} - \log(2^{2d_{max}-(1+2)} + 2^{2d_{max}-(3+1)}) = -\log(2^{-3} + 2^{-4}) \approx 2.42$. In general, $\min\{r_1, ..., r_x\}$ is approximated as $r_{approx} = -\log(2^{-r_1} +, ..., +2^{-r_x})$.

Their solution has three drawbacks: (1) It yields errors whenever $x > 1$. (2) It may yield large relative errors, especially when distance values are small and x is large. For example, consider four candidate results 3,2,3,4, the approximate result is 0.83, with a relative error 0.585. (3) It is not friendly to large d_{max}, because large $2^{d_{max}-d}$ would increase the computation and space overhead.

Our idea comes from a very simple observation. When comparing the sum of two values a, b and the sum of two values c, d, if $a \leq c$ and $b \leq d$ then $a + b \leq c + d$, while if $a \geq c$ and $b \geq d$ then $a + b \geq c + d$. Therefore, in many cases we are able to compare these two sums using only the order of a, b, c, d. For the rest cases, i.e. $a > c$ and $b < d$, or $a < c$ and $b > d$, these two sums are not theorectically comparable using only the order information. Such an observation is really helpful to answer shortest distance queries without knowing exact distance values, because their order information can be used to filter out most candidate results (i.e. value pairs), which are comparable to and larger than any one of other candidate result. To formalize the idea, we construct an order-preserving encryption (OPE) scheme OPE, which involves two encoding methods defined as follows:

Definition 4 (Order Encoding (OE)). OE is a function that takes as input a set S of values and a value $d \in S$. It outputs the order (from small to large) of d in S.

For example, given a set $S = \{45, 23, 57, 7, 16\}$, then $\mathsf{OE}(S, 45) = 4, \mathsf{OE}(S, 23) = 3, \mathsf{OE}(S, 57) = 5, \mathsf{OE}(S, 7) = 1, \mathsf{OE}(S, 16) = 2$.

Definition 5 (Interval Encoding (IE)). IE is a function that takes as input a value d_{min}, a value d_{max} ($d_{max} >$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2880981, IEEE Transactions on Dependable and Secure Computing

Let S be the set of all distance values in the 2HCL index, and d_{min}, d_{max} be the minimum and maximum value respectively. Let SKE be an SKE.

- $\mathsf{Enc}(K, d)$: takes as input a secret key K and a distance value $d \in S$ to be encrypted. It outputs (enc, ord, itv), in which $enc \leftarrow \mathsf{SKE.Enc}(K, d)$, $ord \leftarrow \mathsf{OE}(S, d)$, $itv \leftarrow \mathsf{IE}(d_{min}, d_{max}, k, d)$.
- $\mathsf{Dec}(K, D)$: takes as input a secret key K and an encrypted value D. It parses D as (enc, ord, itv) and outputs $\mathsf{SKE.Dec}(K, enc)$.

Fig. 3. The OPE scheme

d_{min}), an division parameter k (integer) and a value $d_{min} \leq d \leq d_{max}$. It equally divides the interval $[d_{min}, d_{max}]$ into k sub-intervals labeled with order numbers $0, ..., k-1$, and outputs the order number of the sub-interval which contains d.

For example, given $d_{min} = 0$, $d_{max} = 60$ and $k = 3$, then $\mathsf{IE}(d_{min}, d_{max}, k, 7) = 0$, $\mathsf{IE}(d_{min}, d_{max}, k, 21) = 1$, $\mathsf{IE}(d_{min}, d_{max}, k, 35) = 1$, $\mathsf{IE}(d_{min}, d_{max}, k, 52) = 2$.

Fig. 3 presents our OPE scheme. As SKE is secure, the leakage of OPE is from OE and IE, informally described as follows:

- *Order information.* Order information indicates the order-relations (i.e., "<", ">" or "=") between the underlying values of any two encrypted values. Actually, order information is leaked by any OPE [23].
- *Interval information.* Interval information is parametrized by the division parameter k. It indicates the order number of the sub-interval which contains the underlying value of an encrypted value, but not reveals any information about d_{min}, d_{max}.

We formally define order information and interval information in Section 6. Actually, in OPE we can use a Deterministic Encryption (DET) instead of SKE, because the leakage of DET is less than order information. Note that the leakage of OPE is parametrized by k. When $k = 1$, OPE achieves the optimal security for any order-preserving encryption scheme, i.e., only leaking order information.

Let D_1, D_2 be two OPE-encrypted values, we say $D_1 < D_2$, $D_1 = D_2$, $D_1 > D_2$ if $D_1.ord < D_2.ord$, $D_1.ord = D_2.ord$, $D_1.ord > D_2.ord$, respectively. A candidate result is a pair of two encrypted values (D_1, D_2) and we always suppose $D_1 \leq D_2$. If there is more than one candidate results, we use two steps to filter out larger candidate results. In the first step, we make use of order information. Consider two candidate results (D_1, D_2) and (D_3, D_4), we have:

- $(D_1, D_2) = (D_3, D_4)$, if $D_1 = D_3$ and $D_2 = D_4$.
- $(D_1, D_2) < (D_3, D_4)$, if $D_1 < D_3$ and $D_2 <= D_4$ or $D_1 <= D_3$ and $D_2 < D_4$.
- (D_1, D_2) and (D_3, D_4) are not comparable, if $D_1 < D_3$ and $D_2 > D_4$, or $D_1 > D_3$ and $D_2 < D_4$.

Therefore, without performing "+" operations we can still filter out most candidate results, each of which is at least equal or larger than one of the other candidate results. Given a set of candidate results, we define its Filtered Set as follows:

Algorithm 1: Filter

Input: C: A set of candidate results
Output: F: A set of filtered candidate results

```
Initialize an empty set F
for c ∈ C do
    flag = 0
    for f ∈ F do
        if c < f then
            Remove f from F
        else if c ≥ f then
            flag = 1
            break
    if flag = 0 then
        Add c into F
    if F is empty then
        Add c into F
Return F
```

Algorithm 2: Select

Input: F: A set of filtered candidate results
Output: R: A final result

$R = \underset{(D_1, D_2) \in F}{\arg\min} \; D_1.itv + D_2.itv$
Return R

Definition 6 (Filtered Set). Given a set C of candidate results, its Filtered Set F satisfies: (1) $F \subseteq C$. (2) For all $f \in F$ and $c \in C \setminus F$, $f \leq c$. (3) For any two candidate results $f_1, f_2 \in F$, f_1 and f_2 are not comparable.

In Algorithm 1 we formulate an algorithm Filter for generating Filtered Set.

According to the definition, the minimum candidate result must be in the Filtered Set. Since the candidate results in Filtered Set are non-comparable to each other, we randomly select one of them as the final result if there are still more than one candidate results in the Filtered Set. In section 7 we will show that only a few (normally two or three) candidate results are left in Filtered Set. Though we have remarkable chance to successfully select the minimum result, it is still theoretically possible to select a candidate result which is much larger than the minimum one. Therefore, in the second step, we make use of the interval information to bound errors. Given a Filtered Set, we select the final result with the minimum sum of sub-interval order numbers, as shown in Algorithm 2.

We theoretically and experimentally analyze the accuracy of our solution in Section 6 and Section 7, respectively.

5 Our Graph Encryption Scheme

Using OPE as a building block, we present our graph encryption scheme for shortest distance queries. In our design, we use two specific PRFs f, g with the following parameters:

$$f : \{0,1\}^\lambda \times \{0,1\}^* \rightarrow \{0,1\}^\lambda$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2880981, IEEE Transactions on Dependable and Secure Computing

- Setup: $\mathcal{C}$ selects a random key K and inputs a graph G. Then he/she generates the 2HCL index $\{L_v\}_{v\in G} \leftarrow \mathsf{LAB.Label}(G)$. For each $(u, d_{u,v}) \in L_v$, $\mathcal{C}$ uses PRF f to encode vertex identifier u as $U \leftarrow f(K, u||0)$, and uses OPE to encrypt distance value $d_{u,v}$ as $D_{u,v} \leftarrow \mathsf{OPE.Enc}(K, d_{u,v})$. For each $v \in G$, $\mathcal{C}$ computes a token $T_v \leftarrow f(K, v||1)$, and inserts all (T_v, L_v) pairs into a dictionary I which we refer to as an encrypted index. At last, $\mathcal{C}$ sends I to $\mathcal{P}$
- Query: $\mathcal{C}$ inputs two vertices s, t, computes their tokens $T_s \leftarrow f(K, s||1), T_t \leftarrow f(K, t||1)$, and sends T_s, T_t to $\mathcal{P}$. $\mathcal{P}$ searches in the index I and obtains L_s, L_t. For each encoded vertex identifier U that both appears in L_s and L_t, $\mathcal{P}$ obtains a candidate result $(D_{u,s}, D_{u,t})$. After performing Filter and Select $\mathcal{P}$ returns the selected result r to $\mathcal{C}$. Finally, $\mathcal{C}$ decrypts the two distance values in r and calculates the sum as the shortest distance between s, t.

Fig. 4. The straightforward graph encryption scheme

$$g : \{0,1\}^{\lambda} \times \{0,1\}^{*} \rightarrow \{0,1\}^{\lambda+2\cdot len(int)}$$

where λ is the security parameter and $len(int)$ is the number of bits used for storing an integer value.

5.1 Overview of a straightforward approach

We start with a straightforward graph encryption scheme described in Fig. 4. At a high-level, The straightforward scheme works as follows: During Setup phrase, $\mathcal{C}$ computes a 2HCL index and encodes each vertex identifier using f while encrypts each distance value using OPE. $\mathcal{C}$ then outsources the encrypted 2HCL index to $\mathcal{P}$. During Query phase, $\mathcal{C}$ computes the encoded form of the two vertices to be queried and sends them to $\mathcal{P}$. $\mathcal{P}$ then locates the corresponding two entries in the index, computes an encrypted result using the method we discussed in Section 4, and returns the result to $\mathcal{C}$. Finally, $\mathcal{C}$ decrypts the result.

The straightforward scheme successfully answers shortest distance queries on encrypted graph. However, though it encodes all vertex identifiers and distance values, it still leaks much information. Firstly, it leaks the length of each L_v. Secondly, note that f is a deterministic function, $\mathcal{P}$ can judge the equality of any two encoded vertices in the index. In this sense, the straightforward scheme leaks the number of common vertices between any two labels. Once the number is zero, $\mathcal{P}$ learns that the underlying two vertices are unreachable. Thirdly, it leaks the order information and interval information of all distance values in the 2HCL index.

5.2 Our main graph encryption scheme *GENOA*

We now present our main graph encryption scheme GENOA, which prevents the information leakage of the straightforward scheme. We describe the five algorithms (Definition 1), respectively.

Init. In the initialization algorithm, we pick a random key K and generate 2HCL labels using LAB.Label as the same in the straightforward scheme. The algorithm is formulated in Algorithm 3.

Algorithm 3: Init

Input: (λ, G): A security parameter and a graph
Output: $(K, \{L_v\}_{v\in G})$: A secret key and a set of labels

$K \xleftarrow{\$} \{0,1\}^{\lambda}$
$\{L_v\}_{v\in G} \leftarrow \mathsf{LAB.Label}(G)$
Return $(K, \{L_v\}_{v\in G})$

Algorithm 4: Enc

Input: $(K, \{L_v\}_{v\in G})$: A secret key and a set of labels
Output: I: An index

Initialize a dictionary I
for $v \in G$ **do**
  $T_v \leftarrow f(K, v||1)$
  $K_v \leftarrow f(K, v||2)$
  Initialize a counter $c = 0$
  for $(u, d_{u,v}) \in L_v$ **do**
    $U \leftarrow f(K, u||0)$
    $D_{u,v} \leftarrow \mathsf{OPE.Enc}(K, d_{u,v})$
    $T_{u,v} \leftarrow f(T_v, c)$
    $K_{u,v} \leftarrow g(K_v, c)$
    $C_{u,v} = K_{u,v} \oplus (U||D_{u,v}.ord||D_{u,v}.itv)$
    $I[T_{u,v}] = (C_{u,v}, D_{u,v}.enc)$
    $c = c + 1$
return I

Algorithm 5: Token

Input: (K, s, t): A secret key and two vertices
Output: $\tau_{s,t}$: A token

$T_s \leftarrow f(K, s||1)$, $K_s \leftarrow f(K, s||2)$
$T_t \leftarrow f(K, t||1)$, $K_t \leftarrow f(K, t||2)$
return $\tau_{s,t} \leftarrow (T_s, T_t, K_s, K_t)$

Enc. We make several changes for label encryptions. In detail, we use two leakage prevention methods, as follows:

Index Obfuscation. To hide the length of each L_v, we unpack L_v and store each $(U, D_{u,v}) \in L_v$ into the index respectively. The idea is from a searchable encryption work by Cash et al. [7], which hides the number of data files containing a specific keyword. After assigning a token $T_v \leftarrow f(K, v||1)$ for L_v, we further use T_v to compute a sub-token $T_{u,v}$ for each $(U, D_{u,v}) \in L_v$ as follows: Suppose L_v contains l pairs, we generate l sub-tokens as the output of $f(T_v, c)$ in which c is an integer counter increasing from 1 to l. Finally, we store each $(U, D_{u,v}) \in L_v$ into the index I such that it is indexed by its sub-token $T_{u,v}$.

2-Round Encryption. During a query, we have to reveal the number of common vertices (i.e., the number of candidate results) of the involved labels, the order information and the interval information of the distance values stored in the involved labels. However, for those labels having not been queried, we should avoid above leakage. Therefore, we use another round of encryption, which ran-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2880981, IEEE Transactions on Dependable and Secure Computing

Algorithm 6: Dist

Input: $(I, \tau_{s,t})$: An index and a token
Output: $D_{s,t}$: An encrypted result

Parse $\tau_{s,t}$ as (T_s, T_t, K_s, K_t)
Initialize two set L_s, L_t
for $c = 0$ *until* $I[T_{u,s}] = \perp$ **do**
  $T_{u,s} \leftarrow f(T_s, c)$
  $K_{u,s} \leftarrow g(K_s, c)$
  $(C_{u,s}, D_{u,s}.enc) \leftarrow I[T_{u,s}]$
  $U||D_{u,s}.ord||D_{u,s}.itv = K_{u,s} \oplus C_{u,s}$
  $D_{u,s} \leftarrow (D_{u,s}.enc, D_{u,s}.ord, D_{u,s}.itv)$
  Insert $(U, D_{u,s})$ into L_s
for $c = 0$ *until* $I[T_{u,t}] = \perp$ **do**
  $T_{u,t} \leftarrow f(T_t, c)$
  $K_{u,t} \leftarrow g(K_t, c)$
  $(C_{u,t}, D_{u,t}.enc) \leftarrow I[T_{u,t}]$
  $U||D_{u,t}.ord||D_{u,t}.itv = K_{u,t} \oplus C_{u,t}$
  $D_{u,t} \leftarrow (D_{u,t}.enc, D_{u,t}.ord, D_{u,t}.itv)$
  Insert $(U, D_{u,t})$ into L_t
Initialize a set C
Insert all (D_1, D_2) into C where $(U_1, D_1) \in L_s$, $(U_2, D_2) \in L_t$, and $U_1 = U_2$
$F \leftarrow \mathsf{Filter}(C)$
$D_{s,t} \leftarrow \mathsf{Select}(F)$
Return $D_{s,t}$

Algorithm 7: Dec

Input: $K, D_{s,t}$: A secret key and an encrypted result
Output: $d_{s,t}$: A distance value

Parse $D_{s,t}$ as (D_1, D_2)
$d_1 \leftarrow \mathsf{OPE.Dec}(K, D_1)$
$d_2 \leftarrow \mathsf{OPE.Dec}(K, D_2)$
Return $d_{s,t} = d_1 + d_2$

domizes the output of f, OE and IE. Specifically, for each $(U, D_{u,v}) \in L_v, v \in G$, we encrypt U, $D_{u,v}.ord$, $D_{u,v}.itv$ as $C_{u,v} = K_{u,v} \oplus (U||D_{u,v}.ord||D_{u,v}.itv)$, where $K_{u,v}$ is generated in the similar way we compute sub-tokens, namely, assigning a per-vertex key $K_v \leftarrow f(K, v||2)$ for v and computing sub-keys $K_{u,v} \leftarrow f(K_v, c)$ with an increasing counter c.

For the convenience of readers, we summarize the usage of f as follows:

- $V \leftarrow f(K, v||0)$: is to encode the vertex identifier v.
- $T_v \leftarrow f(K, v||1)$: is to generate the token of v.
- $K_v \leftarrow f(K, v||2)$: is to generate the per-vertex key of v.
- $T_{u,v} \leftarrow f(T_v, c)$: is to generate the sub-tokens for the pairs in L_v.
- $K_{u,v} \leftarrow f(K_v, c)$: is to generate the sub-keys for the pairs in L_v.

The formal description of Enc is presented in Algorithm 4.

Token. A query token for vertex pair (s, t) includes (1) label tokens T_s, T_t, which are used to compute the sub-tokens, and (2) per-vertex keys K_s, K_t, which are used to compute the sub-keys. We formulate Token in Algorithm 5.

Dist. Upon receiving the token $T_{s,t}$, $\mathcal{P}$ first computes sub-tokens $T_{u,s} \leftarrow f(T_s, c)$, $T_{u,t} \leftarrow f(T_t, c)$ with an increasing counter c, and obtains all two-round-encrypted pairs. Meanwhile, $\mathcal{P}$ computes the sub-keys $K_{u,s} \leftarrow f(K_s, c)$, $K_{u,t} \leftarrow f(K_t, c)$ with an increasing counter c, decrypts the second round encryptions and repacks L_s and L_t respectively. Then, $\mathcal{P}$ searches common vertices between L_s and L_t and forms a set of candidate results. Finally, $\mathcal{P}$ executes Filter and Select to obtain a final result. We formulate Dist in Algorithm 6.

Dec. Upon receiving the encrypted result $D_{s,t}$ from $\mathcal{P}$. $\mathcal{C}$ parses it as (D_1, D_2) and decrypts them as $d_1 \leftarrow \mathsf{OPE.Dec}(K, D_1)$ and $d_2 \leftarrow \mathsf{OPE.Dec}(K, D_2)$. Finally $\mathcal{C}$ gets the (approximate) shortest distance $d_{s,t} = d_1 + d_2$. We formulate Dec in Algorithm 7.

Using Algorithm 3-7, we formulate protocol Setup and Query in Protocol 1,2.

6 Accuracy and Security Proofs

Theorem 1. If SKE is a secure SKE, f, g are secure PRFs, and LAB has accuracy lower bound $\mathcal{B}_{lower}$ and upper bound $\mathcal{B}_{upper}$, then for a given graph G, GENOA achieves $(\mathcal{B}_{lower}, \mathcal{B}_{upper}, 0, \frac{d_{max} - d_{min}}{k}) - accuracy$ for $k = 1, 2$ and $(\mathcal{B}_{lower}, \mathcal{B}_{upper}, 0, 2 \cdot \frac{d_{max} - d_{min}}{k}) - accuracy$ for $k > 2$, in which d_{max}, d_{min} are the maximum and minimum distance values in the 2HCL index output by $\mathsf{LAB.Label}(G)$ and k is the division parameter.

Proof: For any graph G and vertices $s, t \in G$, let $d_{s,t}^{\mathsf{LAB}}$ be the resultant shortest distance output by LAB, i.e.,

$$d_{s,t}^{\mathsf{LAB}} = \min\{d_{u,s}^{\mathsf{LAB}} + d_{u,t}^{\mathsf{LAB}} | (u, d_{u,s}^{\mathsf{LAB}}) \in L_s^{\mathsf{LAB}}, (u, d_{u,t}^{\mathsf{LAB}}) \in L_t^{\mathsf{LAB}}\}$$

where $L_s^{\mathsf{LAB}}, L_t^{\mathsf{LAB}} \in \{L_v^{\mathsf{LAB}}\}_{v \in G} \leftarrow \mathsf{LAB.Label}(G)$, and let $d_{s,t}$ be the resultant shortest distance output by GENOA, i.e., $(K, \{L_v^{\mathsf{LAB}}\}_{v \in G}) \leftarrow \mathsf{Init}(\lambda, G)$, $I \leftarrow \mathsf{Enc}(K, \{L_v^{\mathsf{LAB}}\}_{v \in G})$, $\tau_{s,t} \leftarrow \mathsf{Token}(K, s, t)$, $D_{s,t} \leftarrow \mathsf{Dist}(I, \tau_{s,t})$, $d_{s,t} \leftarrow \mathsf{Dec}(K, D_{s,t})$.

According to Algorithm 4, 5, for any $v \in G$, we can retrieve the encrypted version of L_v^{LAB} except that with negligible probability collisions happen amongst the outputs of f. As g is also a secure PRF, we can correctly decrypt the second round encryption and obtain all (U, D) pairs, where $U \leftarrow f(K, u||0)$ and $D \leftarrow \mathsf{OPE.Enc}(K, d)$ for all $(u, d) \in L_v^{\mathsf{LAB}}$. Consider a pair $(u_1, d_1) \in L_s^{\mathsf{LAB}}$ and a pair $(u_2, d_2) \in L_t^{\mathsf{LAB}}$, we have $f(K, u_1||0) = f(K, u_2||0)$ if $u_1 = u_2$, and $f(K, u_1||0) \neq f(K, u_2||0)$ if $u_1 \neq u_2$ except that with negligible probability collisions happen amongst the outputs f. Therefore, with high probability the candidate results extracted from encrypted labels are exactly the OPE ciphertexts of that extracted from unencrypted labels. As SKE used in OPE is secure, all encrypted candidate results can be correctly decrypted. Therefore, we have:

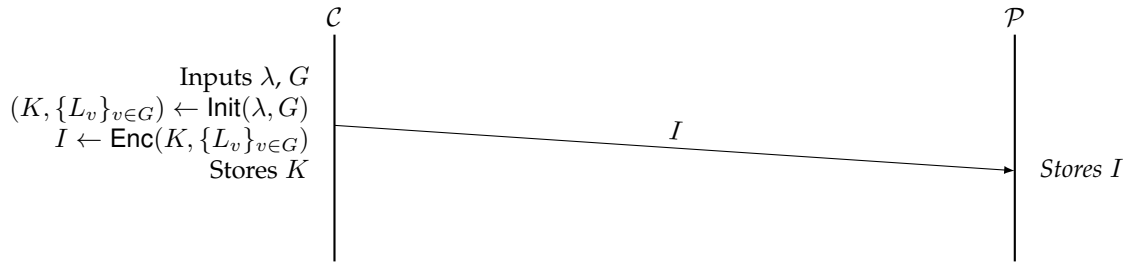
$$\Pr[d_{s,t} = \mathsf{Filter}(\mathsf{Select}(C))] \geq 1 - \mathrm{negl}(\lambda)$$

where C is the candidate result set extracted from L_s^{LAB} and L_t^{LAB}.

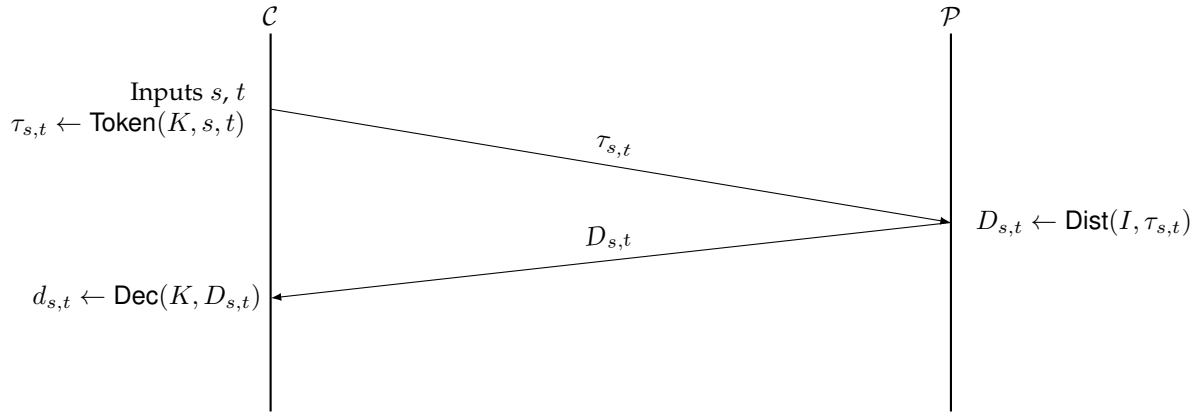
We suppose $d_{s,t}^{\mathsf{LAB}}$ results from $d_s^{\mathsf{LAB}} + d_t^{\mathsf{LAB}}$ and $d_{s,t}$ results from $d_s + d_t$. Note that $d_{s,t}^{\mathsf{LAB}} = \min\{d_1 + d_2 | (d_1, d_2) \in C\}$. Thus:

$$(d_s + d_t) - (d_s^{\mathsf{LAB}} + d_t^{\mathsf{LAB}}) \geq 0$$

Protocol 1: Setup



Protocol 2: Query



According to the definition of Filter, we have:

$$d_s^{\mathsf{LAB}} < d_s \text{ and } d_t^{\mathsf{LAB}} > d_t$$
or
$$d_s^{\mathsf{LAB}} > d_s \text{ and } d_t^{\mathsf{LAB}} < d_t$$
or
$$d_s^{\mathsf{LAB}} = d_s \text{ and } d_t^{\mathsf{LAB}} = d_t$$

Suppose $itv(d_s^{\mathsf{LAB}}) \leftarrow \mathsf{IE}(d_{min}, d_{max}, k, d_s^{\mathsf{LAB}})$, $itv(d_t^{\mathsf{LAB}}) \leftarrow \mathsf{IE}(d_{min}, d_{max}, k, d_t^{\mathsf{LAB}})$, $itv(d_s) \leftarrow \mathsf{IE}(d_{min}, d_{max}, k, d_s)$, $itv(d_t) \leftarrow \mathsf{IE}(d_{min}, d_{max}, k, d_t)$ and $l = \frac{d_{max} - d_{min}}{k}$, we have:

$$itv(d_s^{\mathsf{LAB}}) \cdot l \leq d_s^{\mathsf{LAB}} < itv(d_s^{\mathsf{LAB}}) \cdot l + l$$

$$itv(d_t^{\mathsf{LAB}}) \cdot l \leq d_t^{\mathsf{LAB}} < itv(d_t^{\mathsf{LAB}}) \cdot l + l$$

$$itv(d_s) \cdot l \leq d_s < itv(d_s) \cdot l + l$$

$$itv(d_t) \cdot l \leq d_t < itv(d_t) \cdot l + l$$

According to the definition of Select, we have:

$$itv(d_s^{\mathsf{LAB}}) + itv(d_t^{\mathsf{LAB}}) = itv(d_s) + itv(d_t)$$

Therefore:

$$\begin{aligned}&(d_s + d_t) - (d_s^{\mathsf{LAB}} + d_t^{\mathsf{LAB}})\\ <&itv(d_s) \cdot l + l + itv(d_t) \cdot l + l - itv(d_s^{\mathsf{LAB}}) \cdot l - itv(d_t^{\mathsf{LAB}}) \cdot l\\ =&2 \cdot l\end{aligned}$$

However, when $k = 1, 2$, it holds that $itv(d_s^{\mathsf{LAB}}) = itv(d_s)$ and $itv(d_t^{\mathsf{LAB}}) = itv(d_t)$. We have:

$$\begin{aligned}&(d_s + d_t) - (d_s^{\mathsf{LAB}} + d_t^{\mathsf{LAB}})\\ =&(d_s - d_s^{\mathsf{LAB}}) + (d_t - d_t^{\mathsf{LAB}})\\ <&l\end{aligned}$$

Since $\mathcal{B}_{lower}(s,t) \leq d_{s,t}^{\mathsf{LAB}} \leq \mathcal{B}_{upper}(s,t)$, we have:

$$\Pr[\mathcal{B}_{lower}(s,t) \leq d_{s,t} \leq \mathcal{B}_{upper}(s,t) + 2 \cdot l] \geq 1 - \text{negl}(\lambda)$$

for $k > 2$, and:

$$\Pr[\mathcal{B}_{lower}(s,t) \leq d_{s,t} \leq \mathcal{B}_{upper}(s,t) + l] \geq 1 - \text{negl}(\lambda)$$

for $k = 1, 2$. □

Before the security proof of GENOA, we first describe the leakage functions of the scheme. In the Setup protocol, by observing the length of the encrypted index the adversary only learns the total number of (u, d) pairs in the 2HCL index. Therefore, given a graph G

$$\mathcal{L}_{setup}(G) = \Sigma_{v\in G}|L_v|$$

where $L_v \in \mathsf{LAB.Label}(G)$.

During a single query (s, t) in Query protocol, the adversary learns how many (u, d) pairs are in L_s and L_t. After decrypting the second round encryptions the adversary learns the number of common vertices between L_s and L_t, and the order information and interval information for all underlying distance values stored in L_s and L_t.

To understand what is additionally leaked during multiple queries, we first consider what the adversary additionally learns from two queries (s_1, t_1) and (s_2, t_2). Since f

is deterministic, by observing the tokens the adversary is able to identify the equivalence among s_1, t_1, s_2, t_2 (e.g., $s_1 \neq t_1, s_1 = s_2, t_1 = t_2$). We call such information the repetition information as the adversary learns whether a vertex is repeatedly queried. For multiple queries, the adversary additionally learns the repetition information among all queries. In summary, given a graph G and a sequence of q queries $\mathbf{q} = (s_1, t_1), ..., (s_q, t_q)$,

$$\mathcal{L}_{query}(G, \mathbf{q}) = (\mathsf{len}, \mathsf{num}, \mathsf{ord}, \mathsf{itv}, \mathsf{rep})$$

where len, num, ord, itv, rep are formally defined as follows:

- *Length information (*len). len is an array of size $2q$ such that $\mathsf{len}[i] = |L_{s_i}|$ for $i = 1, 3, ..., 2q-1$ and $\mathsf{len}[i] = |L_{t_i}|$ for $i = 2, 4, ..., 2q$, where $L_v \in \mathsf{LAB.Label}(G)$.
- *Number of common vertices information (*num). num is an array of size q such that for $1 \leq i \leq q$ $\mathsf{num}[i] = |\{u | (u, *) \in L_{s_i}, (u, *) \in L_{s_i}\}|$, where $L_v \in \mathsf{LAB.Label}(G)$ and $*$ denotes arbitrary distance value).
- *Order information (*ord). Let S be the set of all distance values in $\mathsf{LAB.Label}(G)$. ord is an array of size $\Sigma_{1 \leq i \leq 2q}\mathsf{len}[i]$ such that for $1 \leq i \leq \Sigma_{1 \leq i \leq 2q}\mathsf{len}[i]$ $\mathsf{ord}[i] = \{\mathsf{OE}(S, d) | (u, d) \in L_{s_1} || L_{t_1} || ... || L_{s_q} || L_{t_q}\}$.
- *Interval information (*itv). Let S be the set of all distance values in $\mathsf{LAB.Label}(G)$. itv is an array of size $\Sigma_{1 \leq i \leq 2q}\mathsf{len}[i]$ such that for $1 \leq i \leq \Sigma_{1 \leq i \leq 2q}\mathsf{len}[i]$ $\mathsf{itv}[i] = \{\mathsf{IE}(S, d) | (u, d) \in L_{s_1} || L_{t_1} || ... || L_{s_q} || L_{t_q}\}$.
- *Repetition information (*rep). rep is an array of size q such that for $1 \leq i \leq q$ $\mathsf{rep}[i] = (x, y)$ if s_i, t_i is the xth,yth different vertex in the sequence $s_1, t_1, ..., s_i, t_i$, respectively.

We now prove the security of GENOA.

Theorem 2. If SKE is a secure SKE, f, g are secure PRFs, then GENOA achieves $(\mathcal{L}_{setup}, \mathcal{L}_{query})$-security.

Proof: At a high level, the proof is to construct a simulator $\mathcal{S}$ in the game $\mathbf{Ideal}$, who uses $\mathcal{L}_{setup}$ and $\mathcal{L}_{query}$ to simulate a proper index I^*, and tokens $\tau^*_{s,t}$ for all $(s, t) \in \mathbf{q}$. In the following proofs, we say X and X^* are indistinguishable to $\mathcal{A}$ to mean that the probability of distinguishing X and X^* is bounded by $\mathrm{negl}(\lambda)$.

Let $m = \Sigma_{1 \leq i \leq 2q}\mathsf{len}[i]$. $\mathcal{S}$ simulates a set of $\Sigma_{v \in G}|L_v|$ distance values $S^* = \{d^*_1, ..., d^*_m\}$ such that for $1 \leq i \leq m$, $\mathsf{OE}(S^*, d^*_i) = \mathsf{ord}[i]$ and $\mathsf{IE}(\min\{S^*\}, \max\{S^*\}, \max\{itv\} + 1, d^*_i) = \mathsf{itv}[i]$. Then, $\mathcal{S}$ simulates m vertex identifiers $u^*_1, ..., u^*_m$ and insert (d^*_i, u^*_i) into $2q$ labels $L^*_1, ..., L^*_{2q}$ such that (1) for $1 \leq i \leq 2q$, $|L_i| = \mathsf{len}[i]$ and (2) for $1 \leq i \leq q$, the number of common vertices in L^*_{2i-1} and L^*_{2i} is equal to $\mathsf{num}[i]$. $\mathcal{S}$ then deletes repetitive labels and finally obtains, say p different labels. Let's re-denote them as $L^*_1, ..., L^*_p$. $\mathcal{S}$ simulates L^*_{p+1} such that all vertex identifiers and distance values are randomly chosen and $|L^*_{p+1}| = \Sigma_{v \in G}|L_v| - \Sigma_{1 \leq i \leq p}|L^*_i|$. Then $\mathcal{S}$ is ready to simulate index and tokens.

Simulating the index I^*. $\mathcal{S}$ sets $K^* \xleftarrow{\$} \{0,1\}^\lambda$ and obtains $I^* \leftarrow \mathsf{GENOA.Enc}(K, \{L_i\}_{1 \leq i \leq p+1})$. Since $\Sigma_{v \in G}|L_v| = \Sigma_{1 \leq i \leq p+1}|L^*_i|$, the length of I and I^* are equal. Since SKE, f and g are secure, the content of I and I^* are indistinguishable to $\mathcal{A}$ before queries.

Simulating the query $\tau^*_{s_i,t_i}$ ($1 \leq i \leq q$). $\mathcal{S}$ parses $\mathsf{rep}[i]$ as (x, y) and obtains $\tau^*_{x,y} \leftarrow \mathsf{Token}(K^*, s_i, t_i)$. Since f is secure, (s_i, t_i) and (s^*_i, t^*_i) are indistinguishable to $\mathcal{A}$. Since SKE, f and g are secure, the distance computation process and outcome of I and I^* are indistinguishable to $\mathcal{A}$ during queries.

Therefore, the view of $\mathcal{A}$ in game $\mathbf{Real}$ and $\mathbf{Ideal}$ are indistinguishable to $\mathcal{A}$, i.e.,

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

□

7 Performance Evaluation

In this section, we evaluate the performance of GENOA. All test programs were written in Python and run on a 64-bit Windows machine with Intel Core i5-4570 (3.20GHz) CPU and 8GB RAM. We implemented the 2-hop cover labeling scheme proposed by Akiba, Lwata and Yoshida [3], which is an exact-2HCL. SKE and PRFs were implemented using AES and HMAC from the Python Cryptography Toolkit[3] (PyCrypto). The security parameter was set as 128-bit. The default value of k is set as 16, however, the setting of k does not affects the results on efficiency tests (Section 7.3 and 7.4).

7.1 Datasets

TABLE 2
Dataset Statistics

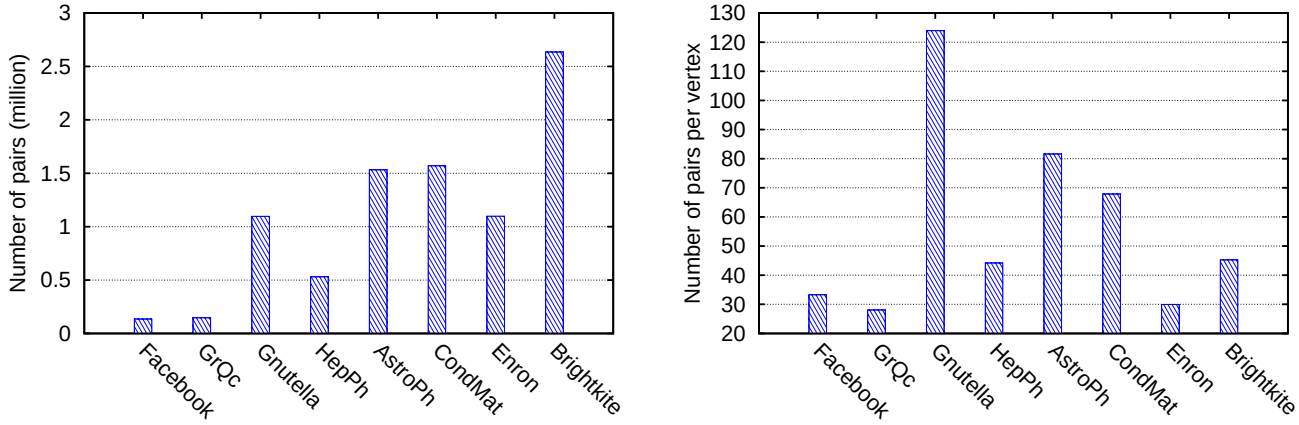
Dataset	Type	Vertices	Edges
Facebook	social network	4,039	88,234
GrQc	collaboration network	5,242	14,496
Gnutella	file sharing network	8,846	31,839
HepPh	collaboration network	12,008	118,521
AstroPh	collaboration network	18,772	198,110
CondMat	collaboration network	23,133	93,497
Enron	email communication network	36,692	183,831
Brightkite	social network	58,228	214,078

We selected various scales of real-world graphs from SNAP [18], as shown in Table 2. For Facebook and Brightkite, each vertex stands for a user and each edge stands for a friend relationship. GrQc, HepPh, AstroPh and CmonMat are collaboration networks of Arxiv on different research areas, in which each vertex represents an author and each edge represents a co-authorship. Gnutella is a peer-to-peer file sharing network, where each vertex represents a host in the Gnutella network topology and each edge represents a connection between two hosts. Enron is an email communication network, where each vertex represents an email address and each edge represents that there is at least one email exchange between the two addresses. Each edge in all networks were assigned a random decimal distance value in the range [0,10] with the precision 0.01.

Since the experiment results in the following sections are highly related to the number of (u, d) pairs in the 2HCL index, here we report such numbers on all 8 datasets in Fig. 5. We can see that the number of (u, d) pairs does not strictly scale with the number of vertices or the number of edges. This is because the number of pairs is also affected by the structural characteristics of the network.

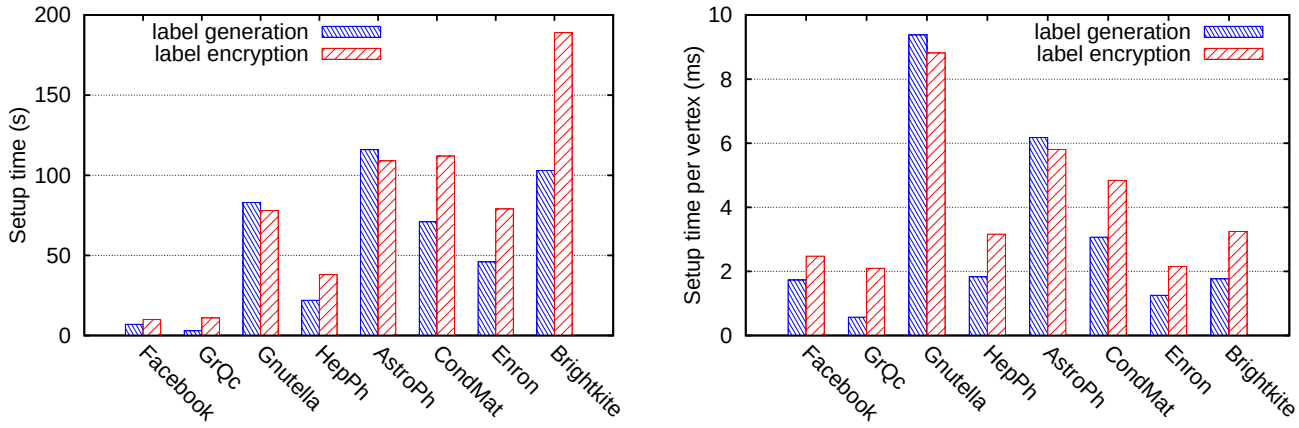
3. https://www.dlitz.net/software/pycrypto/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2880981, IEEE Transactions on Dependable and Secure Computing



(a) Total (b) Per-vertex

Fig. 5. Number of (u, d) pairs in the 2HCL index.



(a) Total (b) Per-vertex

Fig. 6. Execution time for Setup.

7.2 Overview of Complexities

Before experimental study, we compare theorectical complexities of GRECS and GENOA in Table 3. Since both sinces are built on top of a 2HCL scheme, their overheads are highly depends on the size of the 2HCL index generated from the 2HCL scheme. In Table 3, we use n to denote the number of vertices in the graph and m to denote the length of the label in 2HCL index. As a result, both schemes achieve same computation and commication complexities. However, GENOA uses the light-weight OPE instead of SWHE, therefore GENOA is more computationally efficient than GRECS in reality.

TABLE 3
Complexity

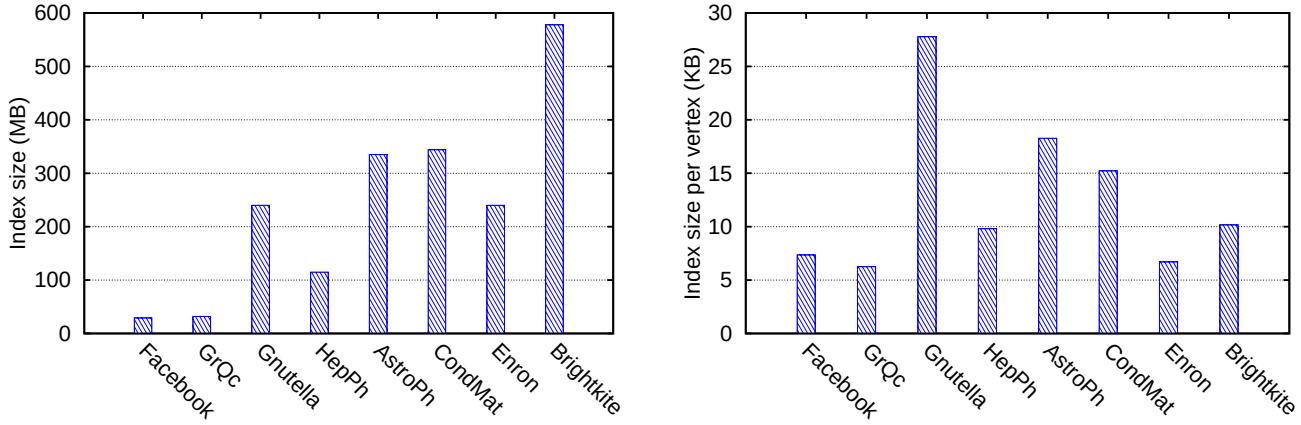
Scheme	Space	Setup Time	Query Time	Communication
GRECS	$O(mn)$	$O(mn)$	$O(m)$	$O(1)$
GENOA	$O(mn)$	$O(mn)$	$O(m)$	$O(1)$

7.3 Setup Performance

We report the execution time and the resultant index size of Setup. We split the entire Setup time into label generation time and label encryption time, as shown in Fig. 6. As the label generation time is decided by the specific 2HCL scheme, we refer the readers to [3] for detailed analyses. We can see from Fig. 5(a),6(a) that the total encryption time is linear in the number of (u, d) pairs, which roughly grows with the size of the network. Even for dataset Brightkite with more than 2.5 million pairs, the encryption finished in about 3 minutes. In Fig. 6(b) we divide the results by the number of vertices. The per-vertex encryption times are less than 10 millisecond for all datasets.

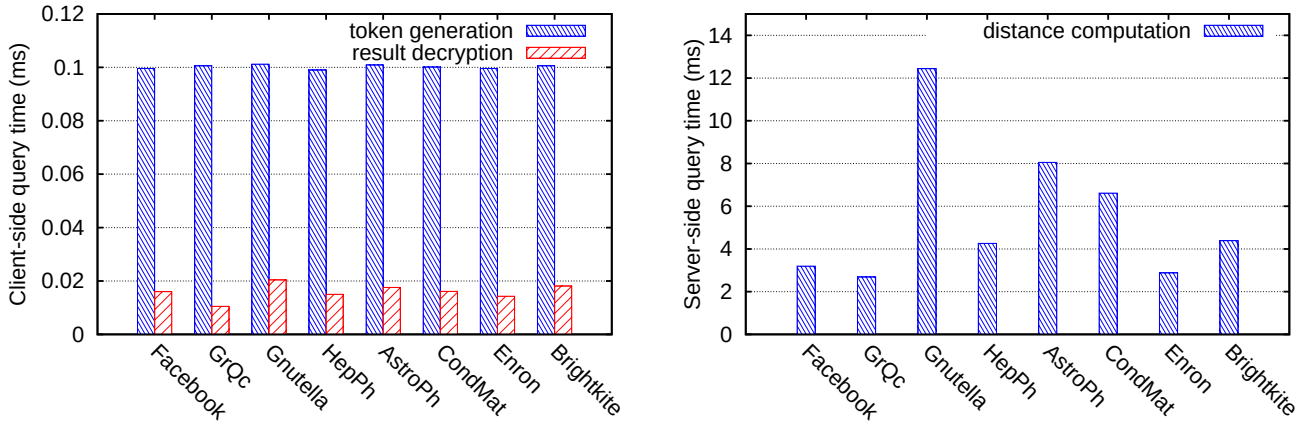
Fig. 7(a) reports the index sizes. According to GENOA.Enc (Algorithm 4), the index size should be also linear in the number of (u, d) pairs. In the experiment results, the smallest index is about 29 MB (on Facebook) while the largest index is about 578 MB (on Brightkite). Normally, exact-2HCL schemes yield more (u, d) pairs than approximate-2HCL schemes. In this sense, we can get even smaller index if we use an approximate-2HCL scheme. In Fig. 7(b), we present the per-vertex sizes. The results range from 6 KB/vertex (on GrQc) to 28 KB/vertex (on Gnutella).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2880981, IEEE Transactions on Dependable and Secure Computing



(a) Total (b) Per-vertex

Fig. 7. Index size.



(a) Client side (b) Server side

Fig. 8. Execution time for Query

7.4 Query Performance

Protocol Query involves client-side (i.e., $\mathcal{C}$-side) token generation (Algorithm 5), result decryption (Algorithm 7) and server-side (i.e., $\mathcal{P}$-side) distance computation (Algorithm 6). If $\mathcal{P}$ finds no candidate result for a query, $\mathcal{C}$ will set the result as zero without decryption. In such a case, we set the decryption time as zero. In the tests we chose 10,000 random queries and obtained the average execution time, as shown in Fig. 8. From Algorithm 5 we can see that the token generation time is independent of the dataset. Thus the token generation times stabilize at about 0.1 milliseconds. The result decryption times are quite small because Algorithm 7 requires only two SKE decryptions. According to Algorithm 6, the distance computation time is decided by the length of L_s, L_t and the number of candidate results. Specifically, an execution requires $|L_s| + |L_t|$ PRF computations, dictionary searches and XORs, $|L_s| \times |L_t|$ condition judgments, a Filter execution and a Select execution. The results show that the distance computation is fairly time saving. Even for Gnutella, whose per-vertex pair number is the largest, the average execution time is just above 12 milliseconds.

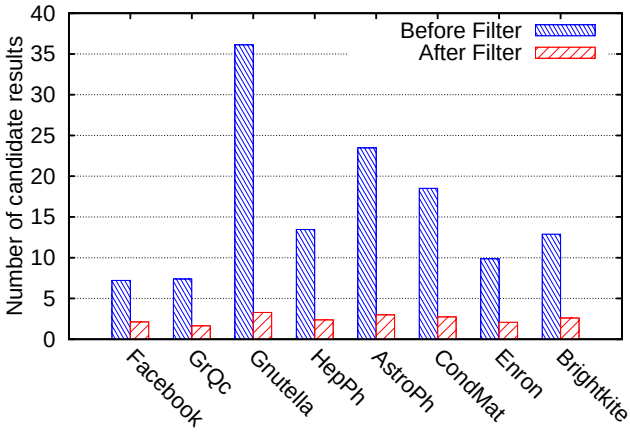


Fig. 9. Performance of algorithm Filter

7.5 Accuracy

We first show the results on the performance of Filter (Algorithm 1) in Fig. 9. The average number of candidate results in a query ranges from 7.2 (on Facebook) to 36.1 (on Gnutella). After performing Filter, the average numbers

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2880981, IEEE Transactions on Dependable and Secure Computing

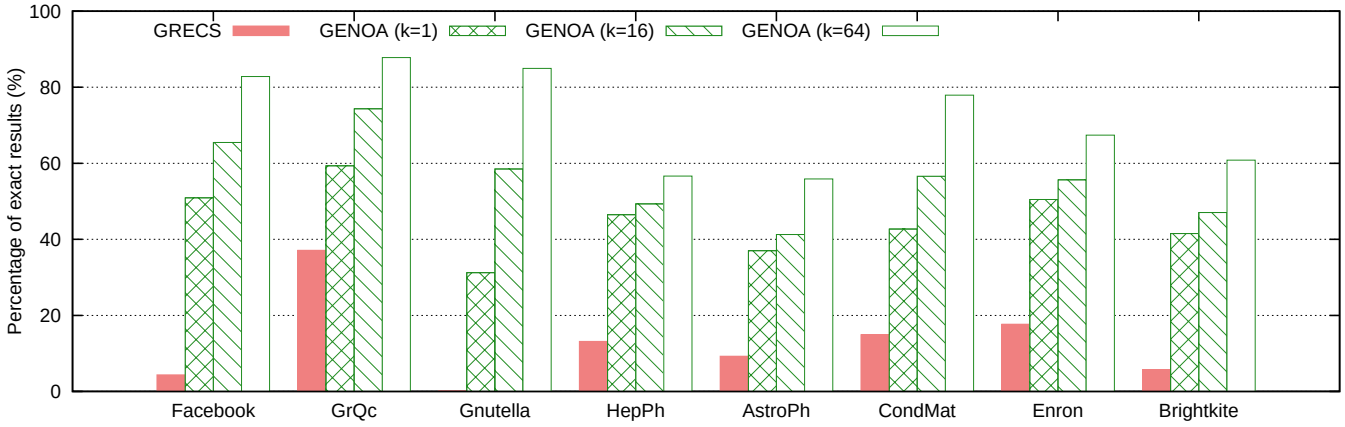


Fig. 10. Percentage of exact results.

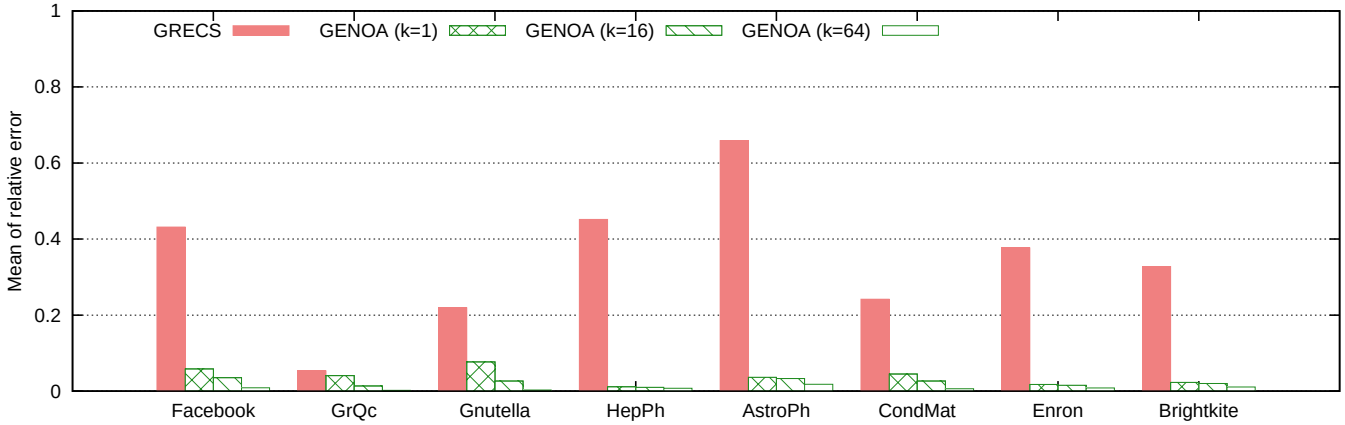


Fig. 11. Mean of related error.

significantly reduce, ranging from 1.6 (on GrQc) to 3.2 (on Gnutella). The results indicate that $\mathcal{P}$ have remarkable chance to obtain the exact distance even without using the interval information.

We compared the accuracy of the proposed GENOA scheme and the scheme in [20] (called GRECS). In the tests we chose 10,000 random queries and obtained the average percentage of exact results and the average related error for both schemes. For GENOA, we separately set the division parameter k used in Select (Algorithm 2) as 1, 16, 64. From Fig. 10 we can see that GENOA yields much more exact results than GRECS even when $k = 1$. Larger k helps to increase the accuracy. For all datasets, more than 50% results of GENOA are exact when $k = 64$. Fig. 11 reports the means of related error. For all datasets, the related error of GENOA is smaller than 0.1 when $k = 1$ and 0.02 when $k = 64$. However, the related error of GRECS is higher than 0.2 for 7 out of 8 datasets, and even exceeds 0.6 on AstroPh.

8 Conclusion and Future Work

In this paper, we proposed a 2-hop cover labeling based graph encryption scheme to answer shortest distance queries on an encrypted and remotely stored graph. Compared to prior work, our scheme can be seen as an alternative for users who desire high efficiency and accuracy, as our experimental results show that the scheme is quite time-saving, space-saving, and can answer almost exact shortest distances. We believe that the leakage of our scheme is acceptable in many applications.

Potential future research directions are summarized as follows. (1) Although leakage of a graph encryption scheme can be precisely define, it is not clear that how much risk users would take in reality. We need deeper understanding of the leakage such as query patterns. One of the possible solutions would be to explore attacks which exploit the leakage to obtain private information. (2) Another research direction is to design graph encryption schemes which support higher-lvel query types. For exampple, using the GENOA scheme as a building block, one can construct provably secure graph encryption schemes supporting k-nearest neighbor (kNN). (3) It is also interesting to explore stronger adversary models, such as malicious adversary who can dishonestly answer queries. In such a case, we need correctness-verifiable schemes.

REFERENCES

[1] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. Hierarchical hub labelings for shortest paths. In *Algorithms–ESA*, pages 24–35. Springer, 2012.

[2] R. Agarwal, P. Godfrey, and S. Har-Peled. Approximate distance queries and compact routing in sparse graphs. In *IEEE INFOCOM*, pages 1754–1762, 2011.

[3] T. Akiba, Y. Iwata, and Y. Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *ACM SIGMOD*, pages 349–360, 2013.

[4] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of Cryptography Conference*, pages 325–341, 2005.

[5] C. Bösch, P. Hartel, W. Jonker, and A. Peter. A survey of provably secure searchable encryption. *ACM Computing Surveys (CSUR)*, 47(2):18, 2014.

[6] N. Cao, Z. Yang, C. Wang, K. Ren, and W. Lou. Privacy-preserving query over encrypted graph-structured data in cloud computing. In *IEEE ICDCS*, pages 393–402, 2011.

[7] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Dynamic searchable encryption in very large databases: Data structures and implementation. In *NDSS*, 2014.

[8] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *CRYPTO*, pages 353–373. Springer, 2013.

[9] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *ASIACRYPT*, pages 577–594. Springer, 2010.

[10] S. Chechik. Approximate distance oracles with constant query time. In *ACM STOC*, pages 654–663, 2014.

[11] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *ACM CCS*, pages 79–88, 2006.

[12] J. Gao, J. X. Yu, R. Jin, J. Zhou, T. Wang, and D. Yang. Neighborhood-privacy protected shortest distance computing in cloud. In *ACM SIGMOD*, pages 409–420, 2011.

[13] S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Outsourced symmetric private information retrieval. In *ACM CCS*, pages 875–888, 2013.

[29] S. Yin, Z. Fan, P. Yi, B. Choi, J. Xu, and S. Zhou. Privacy-preserving reachability query services. In *DASFAA*, pages 203–219. 2014.

[14] R. Jin, N. Ruan, Y. Xiang, and V. Lee. A highway-centric labeling approach for answering distance queries on large sparse graphs. In *ACM SIGMOD*, pages 445–456, 2012.

[15] S. Kamara and K. Lauter. Cryptographic cloud storage. In *Financial Cryptography and Data Security (FC)*, pages 136–149. Springer, 2010.

[16] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *ACM CCS*, pages 965–976, 2012.

[17] K. C. Lee, W.-C. Lee, H. V. Leong, and B. Zheng. Navigational path privacy protection: navigational path privacy protection. In *ACM CIKM*, pages 691–700, 2009.

[18] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[19] K. Liu and E. Terzi. Towards identity anonymization on graphs. In *ACM SIGMOD*, pages 93–106, 2008.

[20] X. Meng, S. Kamara, K. Nissim, and G. Kollios. Grecs: Graph encryption for approximate shortest distance queries. In *ACM CCS*, pages 505–517, 2015.

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

[22] M. Naveed, M. Prabhakaran, and C. A. Gunter. Dynamic searchable encryption via blind storage. In *IEEE S&P*, 2014.

[23] R. A. Popa, F. H. Li, and N. Zeldovich. An ideal-security protocol for order-preserving encoding. In *IEEE S&P*, pages 463–477, 2013.

[24] Z. Qi, Y. Xiao, B. Shao, and H. Wang. Toward a distance oracle for billion-node graphs. *PVLDB*, 7(1):61–72, 2013.

[25] E. Stefanov, C. Papamanthou, and E. Shi. Practical dynamic searchable encryption with small leakage. In *NDSS*, 2014.

[26] T. Velte, A. Velte, and R. Elsenpeter. *Cloud computing, a practical approach*. McGraw-Hill, Inc., 2009.

[27] M. V. Vieira, B. M. Fonseca, R. Damazio, P. B. Golgher, D. d. C. Reis, and B. Ribeiro-Neto. Efficient search ranking in social networks. In *ACM CIKM*, pages 563–572, 2007.

[28] J. Wu, L. Ping, X. Ge, Y. Wang, and J. Fu. Cloud storage as the infrastructure of cloud computing. In *IEEE ICICCI*, pages 380–383, 2010.

[30] B. Zhou and J. Pei. Preserving privacy in social networks against neighborhood attacks. In *IEEE ICDE*, pages 506–515, 2008.