Privacy-Preserving Approximate Top-k Nearest Keyword Queries over Encrypted Graphs

Meng Shen*, Minghui Wang†, Ke Xu‡ and Liehuang Zhu*
*School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing, P. R. China
†School of Computer Science, Beijing Institute of Technology, Beijing, P. R. China
‡Tsinghua University & BNRist, Beijing, P. R. China
{shenmeng, wangmh, liehuangz}@bit.edu.cn; xuke@tsinghua.edu.cn

***Abstract*—With the prosperity of graph-based applications, it is increasingly popular for graph nodes to have labels in terms of a set of keywords. The top-k nearest keyword (k-NK) query can find a set of k nearest nodes containing a designated keyword to a given source node. In cloud computing era, graph owners prefer to outsource their graphs to cloud servers, leading to severe privacy risk for conducting k-NK queries. The current studies fail to support efficient and accurate k-NK query under the premise of privacy protection.**

In this paper, we propose a new graph encryption scheme Aton, which enables efficient and privacy-preserving k-NK querying. Based on the symmetric-key encryption and particular pseudo-random functions, we construct a secure k-NK query index. Aton is built on a ciphertext sum comparison scheme which can achieve approximate distance comparison with high accuracy. Rigorous security analysis proves that it is CQA-2 secure. Experiments with real-world datasets demonstrate that it can efficiently answer k-NK queries with more accurate results compared with the state-of-the-art.

***Index Terms*—Cloud computing, privacy, graph encryption, top-k nearest keyword query**

I. Introduction

Recent years have witnessed a growing application of graph in a variety of fields, such as social networks, road networks and web networks [1]. There are many systems (e.g., Hdrf [2]) and engines (e.g., PowerGraph [3] and Snap [4]) to analyze and query massive graphs. To make graph data more expressive, many graphs in real-world applications have their nodes (a.k.a vertices) associated with *labels*, where a label is usually represented by one or multiple *keywords* [5]. For instance, in social networks like Facebook, each node representing a user may contain labels such as one's *work* or *hobbies*. In this paper, we focus on the top-k nearest keyword (k-NK) query on graphs. Given a graph G with edge weights, a k-NK query, consisting of a source node v, a keyword w, and an integer k, returns the k nodes closest to the source node with the desired keyword. For example, in social networks, a k-NK query can find for a given user, who likes *hiking*, the most closely related friends with the same hobby.

As the scale of graph data grows, graph owners are accustomed to outsource their data to cloud servers to provide high-quality services for geographically distributed users. However, plain graphs containing sensitive information in the cloud are facing severe risk of privacy leakage, due to the existence of malicious attackers and semi-honest servers [6–8]. Therefore, it is meaningful to study the privacy-preserving query over encrypted graphs. There are many algorithms that support k-NK queries on plain graphs [9–11]. However, these algorithms will lose the ability to query in the encrypted graph environment without a special design.

978-0-7381-3207-5/21/$31.00 © 2021 IEEE

TABLE I: Comparison of Existing Solutions

	[9, 10]	[12]	[14]	[16]	[18]	Aton
Privacy protection		✓	✓		✓	✓
General graphs	✓		✓	✓	✓	✓
Weighted graphs	✓			✓	✓	✓

To protect graph privacy on cloud, recent studies focus on k-NK queries over encrypted graphs, which can be roughly classified into two categories. The *first* category is concerned with queries on spatial data, where nodes in graphs are represented in the form of coordinates [12]. However, these schemes work only on graphs with coordinates, such as road networks, and are not applicable to general graphs without coordinates. The *second* category aims at enabling k-NK query over general graphs [13–16]. In these schemes, keyword search and adjacent node search are usually carried out one after the other, resulting in low inefficiency. Another limitation is that many schemes apply only to unweighted graphs because they cannot compare distance values over encrypted graphs. Several solutions [16–19] enable summation and comparison over ciphertexts, but they still have apparent limitations, in terms of insecurity [16], low efficiency [17, 18], and low accuracy [19]. Therefore, *it is still a challenging task to conduct efficient and accurate k-NK query while ensuring data security*.

In this paper, we propose a novel graph encryption scheme Aton, which enables efficient and privacy-preserving k-NK query over encrypted graphs. Due to the barrier imposed by data encryption, we resort to approximate solutions to answer k-NK query. Meanwhile, Aton can still achieve high accuracy compared with the exact answers. We calculate the shortest distance through the 2-hop labeling ($2HL$) index [20], which can answer shortest distance queries efficiently, and achieve keyword matching through keyword-lookup tree (KT) index [10]. Based on these two indexes, we build a secure k-NK query index, where nodes and keywords are encrypted by the symmetric-key encryption (SKE) and particular pseudo-

random functions (PRFs), respectively. To achieve secure distance comparison of ciphertexts, we propose a simple and efficient ciphertext sum comparison scheme by using order-revealing encryption (ORE) [21].

The overall differences of Aton from previous solutions are summarized in Table I, where Privacy refers to the distance between nodes. The main contribution of this paper are as follows:

1) We propose a novel graph encryption scheme Aton, which can realize the approximate top-k nearest keyword (k-NK) query over encrypted graphs. It can answer a k-NK query in a privacy-preserving and time-efficient way.
2) We design a linear efficient ciphertext sum comparison scheme based on the order-revealing encryption, which can compare the sum of two different integers to a great extent at the ciphertext level only through one interaction with no need of additional server.
3) We conduct rigorous security analysis of Aton and prove that it achieves CQA-2 security. Detailed experiments with real-world datasets demonstrate that Aton is linearly efficient and more accurate than existing schemes with the same security guarantees.

The rest of this paper is organized as follows. We describe the background of k-NK query in Section II and define the privacy-preserving k-NK query problem in Section III. We present design details of Aton in Section IV and describe the novel ciphertext sum comparison scheme in Section V. After that, we exhibit the security analyses in Section VI and evaluate the proposed scheme through experiments in Section VII. Finally, we review the related work in Section VIII and conclude this paper in Section IX.

II. Backgroud

This section presents the formal definition of the k-NK query and introduces the index structure used in the query.

A. k-NK Query

Given an undirected graph[1] $G = (V, E)$, where V is the set of nodes and E contains all the edges in G. The number of nodes and edges in G are denoted by $n = |V|$ and $m = |E|$. Each edge $e(s, t) \in E$ is associated with a positive length d_{st}. We use $d_G(v, u)$ to denote the shortest distance between node v and u. In the undirected graph we have d_{st}=d_{ts} and $d_G(v, u)$=$d_G(u, v)$. Let the set of keywords on the graph G be $W = \{w_1, w_2, \ldots, w_{|W|}\}$, where $|W|$ represents the number of different keywords. Each node $v \in V$ contains a set of one or more keywords which is denoted by W_v. Let V_w be the set of nodes containing the keyword w. The set of all (v, w) pairs in G is denoted by $doc(V)$. Some of notations used in this paper are listed in Table II.

Definition 1. *(k-NK QUERY).* Given a graph $G = (V, E)$, a k-NK query (v, w, k) contains three elements where $v \in V$ is the query node, w is a keyword, and k is a positive integer and returns a set R of k nodes, all of which belong to V_w. There does not exist a node $u \in V_w$, $u \notin R$ such that $d_G(v, u) < max_{y \in R} d_G(v, y)$.

[1]We refer to G as an undirected graph in this paper, unless otherwise specified. The extension to directed graphs will be described later.

TABLE II: List of Notations

Notation	Meaning
$G = (V, E)$	Input graph
n, m	Number of nodes and edges in G
d_{st}	Distance of the edge $e(s, t)$
λ	Security parameter
$d_G(v, u)$	Shortest distance from v to u
I	Secret k-NK index structure
$\Delta, \widetilde{\Delta}$	Plain and secret 2-hop label index
$\Psi, \widetilde{\Psi}$	Plain and secret 2-hop label backward index
$\Theta, \widetilde{\Theta}$	Plain and secret low frequency keyword index
$\Upsilon, \widetilde{\Upsilon}$	Plain and secret keyword-lookup tree index
$doc(V)$	Keywordlist of G

Example 1. *Fig. 1 shows a simple graph G, where the distance of each edge and the keyword of each node are marked alongside it. For example, given a k-NK query $(v_1, w_2, 2)$ then the answer will be $\{v_6, v_5\}$. Because $V_{w_2} = \{v_3, v_4, v_5, v_6\}$ and $d_G(v_1, v_6) < d_G(v_1, v_5) < d_G(v_1, v_3) = d_G(v_1, v_4)$.*

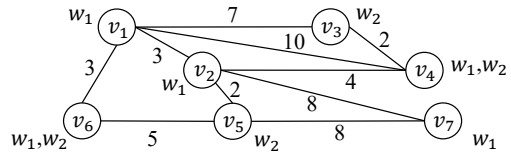


Fig. 1: A graph G with keywords.

B. Constructing Index

The encrypted index designed in this paper is mainly constructed based on the k-NK query index, which contains three parts: 2-hop labeling ($2HL$) index, 2-hop labeling backward (BL) index and keyword-lookup tree (KT) index.

2-hop labeling index. In order to speed up distance query, many $2HL$ index construction schemes are proposed [20].

Given a graph $G = (V, E)$, we denote its $2HL$ index as Δ. Each node $v \in V$ has a $2HL$ index $\Delta(v)$. For any entity $(u, d_{vu}) \in \Delta(v)$, d_{vu} is the shortest distance between node u and v. When we want to query the shortest distance from node s to t, we need to find the common node in $\Delta(s)$ and $\Delta(t)$ first and then select the shortest distance from s to t.

Example 2. *Take the graph G in Fig. 1 for example, the $2HL$ index Δ of G is shown in Fig. 2. If we want to get the shortest distance from node v_4 to v_5, we fisrt find the common nodes $\{v_1, v_2\}$ in $\Delta(v_4)$ and $\Delta(v_5)$, which is shown on the left in Fig. 3. Then calculate $d_{v_4 v_1} + d_{v_1 v_5} = 12$ and $d_{v_4 v_2} + d_{v_2 v_5} = 6$ separately. So $d_G(v_4, v_5) = 6$.*

2-hop labeling backward index. The BL index is designed to achieve efficient k-NK query under high frequency keywords. We denote the BL index as Ψ which is built based on $2HL$ index Δ. For each $v \in V$, if and only if $(v, d_{uv}) \in \Delta(u)$, there have $(u, d_{vu}) \in \Psi(v)$. Each entity in $\Psi(v)$ is sorted in non-ascending order of distance.

V	$2HL$ index
v_1	$\{(v_1,0)\}$
v_2	$\{(v_2,0),(v_1,3)\}$
v_3	$\{(v_3,0),(v_1,7),(v_2,6)\}$
v_4	$\{(v_4,0),(v_1,7),(v_2,4),(v_3,2)\}$
v_5	$\{(v_5,0),(v_1,5),(v_2,2)\}$
v_6	$\{(v_6,0),(v_1,3),(v_5,5)\}$
v_7	$\{(v_7,0),(v_1,11),(v_2,8),(v_5,8)\}$

V	BL index
v_1	$\{(v_7,11),(v_4,7),(v_3,7),(v_5,5),(v_6,3),(v_2,3),(v_1,0)\}$
v_2	$\{(v_7,8),(v_3,6),(v_4,4),(v_5,2),(v_2,0)\}$
v_3	$\{(v_4,2),(v_3,0)\}$
v_4	$\{(v_4,0)\}$
v_5	$\{(v_7,8),(v_6,5),(v_5,0)\}$
v_6	$\{(v_6,0)\}$
v_7	$\{(v_7,0)\}$

Fig. 2: The $2HL$ index and BL index of G.

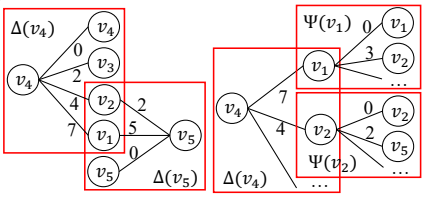


Fig. 3: Query on the $2HL$ index and BL index.

Example 3. *The BL index Ψ of G is shown in Fig. 2. Given a k-NK query (v_4, w_2, k), as shown on the right in Fig. 3, we find $\Delta(v_4)$ first. Since $(v_2, 4) \in \Delta(v_4)$, $(v_5, 2) \in \Psi(v_2)$ and $v_5 \in V_{w_2}$, node v_5 with a distance of 6 might be one of the answers.*

Keyword-lookup tree index. Since BL index may contain nodes that don't contain the keyword w, we need KT index, which is denoted by Υ, to efficiently find entries for nodes that do. Υ consists of keyword-lookup tree index $\Upsilon(v)$ for each node $v \in V$. For $(u, d_{vu}) \in \Psi(v)$, $\Upsilon(v)$ divides the entries in $\Psi(v)$ into fragments. Each leaf node contains the hash value of all keywords for a node, and each tree node contains the hash value of a subset of keywords. A hash function H is used in this process. For each keyword w, the hash function $H(w)$ sets exactly one of the h bit hash value to 1. Given a node set X and a keyword w, if $H(X) \wedge H(w) = 0$, no node in X contains the keyword w, otherwise may contain w.

We store $\Upsilon(v)$ for $\Psi(v)$ in an array of length $2\,|\Psi(v)| - 1$. The number of unused spaces in the array that stores $\Upsilon(v)$ is $bit(|\Psi(v)|) - 1$, where $bit(x)$ is the number of 1's in the binary representation of x. We refer the reader to [10] for more construction and search details.

Example 4. *Fig. 4 shows the $\Upsilon(v_1)$ of node v_1 in G. The leaf node n_2 contains the hash value $H(n_2) = H(w_1) \vee H(w_2)$ of all keywords in W_{v_4}, while the non-leaf node n_1 contains the hash value $H(n_1) = H(n_0) \vee H(n_2)$ of tree nodes n_0 and n_2. Given the keyword w_1, we compute $H(w_1)$ first. Since $H(n_1) \wedge H(w_1) \neq 0$, a node in $\{v_7, v_1\}$ may contain w_1. We calculate that $2\,|\Psi(v_1)| - 1 = 13$ and $bit(|\Psi(v_1)|) = 3$. As shown in Fig. 5, the unused space denoted by $\vdash$ is 2.*

III. Problem Formulation

This section presents the system model and the security model of the privacy-preserving k-NK querying.

A. System Model

We adopt the general system model [22] for the secure k-NK querying. As shown in Fig. 6, our model contains two entities, named a *user* and a *cloud sever* (sever for short).

The user is the owner of graph data, who constructs the secure searchable index for the graph and outsources the encrypted graph and index to the sever. When the user wants

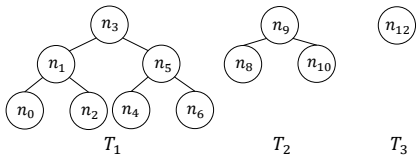


Fig. 4: An example of $\Upsilon(v_1)$.

0	1	2	3	4	5	6
$H(n_0)$	$H(n_1)$	$H(n_2)$	$H(n_3)$	$H(n_4)$	$H(n_5)$	$H(n_6)$

7	8	9	10	11	12
$\vdash$	$H(n_8)$	$H(n_9)$	$H(n_{10})$	$\vdash$	$H(n_{12})$

Fig. 5: The array for keeping the forest of $\Upsilon(v_1)$.

to perform a k-NK query, he first generates the corresponding token and sends it to the sever. Upon receiving user's query token, the sever queries on the encrypted index according to the pre-designed query algorithms. The user decrypts the result returned by the sever to get the final answer. We define the framework of the graph encryption scheme as follows:

Definition 2. *(GRAPH ENCRYPTION).* A graph encryption scheme Π is a collection of three polynomial-time algorithms $\Pi = (KeyGen, Setup, Query)$ that work as follows:

- $K \leftarrow KeyGen(\lambda)$: is a secret key generation algorithm that takes as input a security parameter λ and outputs a secret key K .
- $I \leftarrow Setup(K, G, doc(V))$: is a graph encryption algorithm that takes as input a secret keys K, a graph G and a keyword list $doc(V)$, and outputs the secure index I.
- $R \leftarrow Query(I, (K, q))$: is a twoparty protocol between a user who holds (K, q) and a server who holds I. After executing this protocol, the server returns an encrypted result R. The user decrypts R to get Ans.

B. Security Model

We define the security of our graph encryption scheme using the standard simulation-based model which has been widely adopted by secure computation works [22–25]. In this paper, we consider an honest-but-curious adversary who has access to all encrypted data and queries from the user. Now, we present the formal CQA2-security definitions as follows.

Definition 3. *(CQA2-security model). Let $\Pi = (KeyGen, Setup, Query)$ be a graph encryption scheme and consider the following probabilistic experiments where $\mathcal{A}$ is a semi-honest adversary, $\mathcal{S}$ is a simulator, and $\mathcal{L}_{Setup}$ and $\mathcal{L}_{Query}$ are leakage functions which capture the leaked information during Setup and Query, respectively.*

Real$_{\Pi,\mathcal{A}}(\lambda)$:

- $\mathcal{A}$ outputs a graph G.
- The challenger begins by running $KeyGen(\lambda)$ to create a secret key K, and then computes the encrypted index I by $Setup(K, G, doc(V))$. The challenger sends the encrypted index I to $\mathcal{A}$.
- $\mathcal{A}$ adaptively issues k-NK queries. For each query q, $\mathcal{A}$ and the challenger execute $Query(I, (K, q))$.
- $\mathcal{A}$ computes a bit $b \in \{0, 1\}$ as the output of the experiment.

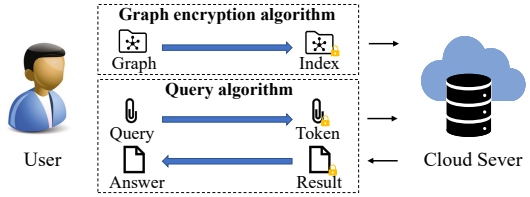


Fig. 6: The system model of secure k-NK query scheme.

Ideal$_{\Pi,\mathcal{A},\mathcal{S}}(\lambda)$:

- $\mathcal{A}$ outputs a graph G.
- Given the leakage function $\mathcal{L}_{Setup}(G)$, $\mathcal{S}$ simulates a secure graph index I^* and sends it to $\mathcal{A}$.
- $\mathcal{A}$ adaptively issues k-NK queries. $\mathcal{S}$ is given the leakage function $\mathcal{L}_{Query}(G, Q)$ for each query q. $\mathcal{A}$ and $\mathcal{S}$ execute a simulation of $Query$.
- $\mathcal{A}$ computes a bit $b \in \{0, 1\}$ as the output of the experiment.

We say that the graph encryption scheme $\Pi = (KeyGen, Setup, Query)$ is $(\mathcal{L}_{Setup}, \mathcal{L}_{Query})$-secure against the adaptive chosen-query attack, if for the adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ satisfying

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

where $negl(\lambda)$ is a negligible function.

IV. Construction Of Aton

In this section, we introduce the verifiable graph encryption scheme Aton for the secure k-NK queries.

In our design, we use four particular pseudorandom functions g, h, f and F and a symmetric-key encryption (SKE) scheme. The parameters of g, h, f and F are illustrated in (1),

$$g : \{0,1\}^{\lambda} \times \{0,1\}^{*} \to \{0,1\}^{\lambda} \tag{1a}$$
$$h : \{0,1\}^{\lambda} \times \{0,1\}^{*} \to \{0,1\}^{z+2\lambda} \tag{1b}$$
$$f : \{0,1\}^{\lambda} \times \{0,1\}^{*} \to \{0,1\}^{2r+2\lambda} \tag{1c}$$
$$F : \{0,1\}^{\lambda} \times \{0,1\}^{*} \to 2^{i}(i \in t) \tag{1d}$$

where λ is the security parameter, z and r are the output lengths of the SKE and ORE encryptions and F is the PRF that sets exactly one of the t bit hash value to 1.

A. Secure k-NK Index Construction

In the key generation phase, given the security parameter λ, the user randomly generates three secret key K_1, K_2, K_3 and K_4 for PRFs, ORE, SKE and OE respectively.

According to the index structure introduced in Section II, the k-NK query can be divided into forward query (FQ) and forward backward query (FBQ) for keywords of different frequencies. To implement a k-NK query, we need to precompute a secure index structure, which consists of four parts: low-frequency keyword (LF) index, secure 2-hop labeling (SL) index, 2-hop labeling backward (SBL) index and keyword tree (SKT) index.

Given a k-NK query (v, w, k) in FQ, since w is not frequent, we can first find all the nodes containing w in V_w before

Algorithm 1: LF Index Generation Algorithm

Input: A secret key K, and low frequency keyword set W
Output: A secure LF index $\widetilde{\Theta}$

1 Initialize a dictionary $\widetilde{\Theta}$
2 **for** *each* $w \in W$ **do**
3 Set $T_w^1 = g(K_1, w||1)$, $T_w^2 = g(K_1, w||2)$
4 Initialize a counter $c = 0$
5 **for** *each* $v \in V_w$ **do**
6 Compute V =SKE$(K_3, v||1)$
7 Set $T_{wv}^1 = g(T_w^1, c)$, $T_{wv}^2 = h(T_w^2, c)$
8 Set $T_v^1 = g(K_1, v||3)$, $T_v^2 = g(K_1, v||4)$
9 Set $\widetilde{\Theta}[T_{wv}^1] = (V||T_v^1||T_v^2) \oplus T_{wv}^2$
10 Set $c = c + 1$
11 **return** $\widetilde{\Theta}$

Algorithm 2: SL Index Generation Algorithm

Input: A secret key K and an graph $G = (V, E)$
Output: A secure SL index $\widetilde{\Delta}$

1 Generate the $2HL$ index Δ of G
2 Initialize a dictionary $\widetilde{\Delta}$
3 **for** *each* $v \in V$ **do**
4 Set $T_v^1 = g(K_1, v||3)$, $T_v^2 = g(K_1, v||4)$, $T_v^3 = g(K_1, v||5)$
5 Initialize a counter $c = 0$
6 **for** *each* $(u, d) \in \Delta(v)$ **do**
7 Compute U =SKE$(K_3, u||2)$, $(D, \hat{D})$ =OE(K_2, d, K_4)
8 Set $T_u^1 = g(K_1, u||6)$
9 Set $T_{vu}^1 = g(T_v^1, c)$, $T_{vu}^2 = f(T_v^2, c)$, $T_{vu}^3 = h(T_v^3, c)$
10 Set $\widetilde{\Delta}[T_{vu}^1] = (U||D||\hat{D}||(T_u^1 \oplus T_{vu}^3)) \oplus T_{vu}^2$
11 Set $c = c + 1$
12 **return** $\widetilde{\Delta}$

calculating the distance value. We do this by constructing two dictionaries LF and SL indexes, which are denoted by $\widetilde{\Theta}$ and $\widetilde{\Delta}$ as described in Algorithm 1 and 2. To prevent leaking the size of keyword list and sketch before querying, we separate each entry in the plain index and ensure that they are stored separately in the dictionary, with a size of one. More precisely, we utilize a counter c and generate the unique T_w^1 and T_w^2 for each keyword w (line 3) and the unique T_v^1 and T_v^2 for each entity in $\Delta(v)$ (line 8). The T_w^1 (or T_v^1) indicates the position that the entity will be stored in $\widetilde{\Theta}$ (or $\widetilde{\Delta}$). T_v^1 and T_v^2 (line 8) are used to link to $\widetilde{\Delta}$, because the same encryption is used in $\widetilde{\Delta}$. Similarly, T_u^1 in Algorithm 2 (line 9) is also used to connect to another index, as we will explain later. T_w^2 (or T_v^2, T_v^3) is used for second-round encryption of the encrypted value by XOR operation. Through these operations, we guarantee that the size of each V_w and sketch, the link to $\widetilde{\Delta}$ in $\widetilde{\Theta}$, the number of common nodes in different sketches and order information of distance in $\widetilde{\Delta}$ will not be disclosed before the query.

If $|V_w|$ is large, FQ becomes inefficient because there are many candidates in V_w need to be checked. In addition to SL index, we achieve FBQ by constructing two dictionaries SBL index and SKT index, which are denoted by $\widetilde{\Psi}$ and $\widetilde{\Upsilon}$. To ensure that FBQ on encrypted graphs can be implemented, we still store $\widetilde{\Psi}$ and $\widetilde{\Upsilon}$ in arrays. The encryption process of BL index is shown in Algorithm 3. For any $u \in V, (x, d) \in \Psi(u)$, use SKE to encrypt node x and OE to encrypt d where OE is introduced in Section V. T_u^1 (line 5) is where each term $\widetilde{\Psi}(u)$ is stored in $\widetilde{\Psi}$. SKT index $\widetilde{\Upsilon}$ is constructed similarly to Υ, but we use the PRF F instead of H when encrypting

Algorithm 3: SBL Index Generation Algorithm

Input: A secret key K and 2-hop label index Δ of G
Output: A secure SBL index $\widetilde{\Psi}$

Generate the BL index Ψ of G
Initialize a dictionary $\widetilde{\Psi}$
for *each* $u \in V$ **do**
  Initialize an array $\widetilde{\Psi}(u)$
  Set $T_u^1 = g(K_1, u||6)$
  for *each* $(x, d) \in \Psi(u)$ **do**
    Compute $X =$SKE$(K_3, x||1)$, $(D, \hat{D}) =$OE(K_2, d, K_4)
    Insert $(X||D||\hat{D})$ into $\widetilde{\Psi}(u)$ in the original order
  Set $\widetilde{\Psi}[T_u^1] = \widetilde{\Psi}(u)$
return $\widetilde{\Psi}$

keywords. We also store each term $\widetilde{\Upsilon}(u)$ in $\widetilde{\Upsilon}$ in location T_u^1. In $Setup$ algorithm, we finally get the secret k-NK index structure $I = (\widetilde{\Theta}, \widetilde{\Delta}, \widetilde{\Psi}, \widetilde{\Upsilon})$.

B. Secure k-NK Querying

$Query$ contains two different algorithms FQ and FBQ. The FQ algorithm in Algorithm 4 works as follows. Given a k-NK query (v, w, k), the user first generates the corresponding query token τ and sends it to the sever (lines 1-2). After receiving τ, the sever finds the $\widetilde{\Theta}(w)$ of the keyword w through T_1 and T_2 (lines 4-10), and constructs $\widetilde{\Delta}(T)$ by finding $\widetilde{\Delta}(u)$ according to T_u^1 and T_u^2 in $\widetilde{\Theta}(w)$ (lines 11-20). Then the sever finds $\widetilde{\Delta}(v)$ of node v in $\widetilde{\Theta}$ according to T_3 and T_4 (lines 21-27). For each $\widetilde{\Delta}(u) \in \widetilde{\Delta}(T)$, the sever calculates the shortest distance $d_G vu$ and puts it into C (lines 28-31). Finally, the sever finds the k items with the smallest distance in C to form the query result R and returns R to the user. The Algorithm 7 is used in both calculating the shortest distance and sorting.

Next, we introduce how to answer a k-NK query by FBQ algorithm in Algorithm 5. Similar to Algorithm 4, the user generates a token τ and sends it to the sever (lines 1-2). After receiving τ, the cloud first obtains $\widetilde{\Delta}(v)$ of node v and the link T_u to the relevant $\widetilde{\Psi}(u)$ and $\widetilde{\Upsilon}(u)$ according to T_2, T_3 and T_4 (lines 4-11). Then sever initializes a double-ended priority queue PQ of size k to store candidates (line 12). For each item in $\widetilde{\Delta}(v)$, given X_{j-1}, sever finds the next shortest term $(X_j, D3||D4)$ containing the keyword w in $\widetilde{\Psi}(u)$ through the function $KTnext$ (line 14). $KTnext$ traverses $\widetilde{\Upsilon}[T_{u_i}]$ and checks whether there has the j-th node containing the keyword w through the bitwise AND operation with T_1.

The function $KTnext$ in this paper is the same as the function $KTsearchNext$ in [10], except we use PRF F instead of hash function H when checking the keywords. Therefore, we skip further details due to space limitations. The sever then inserts some candidates to PQ (line 16). $InsertPQ$ is shown in Algorithm 6. After PQ is initialized, the sever pops the minimum item $(U^*, X_j^*, D1^*||D2^*||D3^*||D4^*)$ from PQ, and inserts X_j^* into R. When an item is popped, the sever looks for the next X_{j+1} for each item linked to it (lines 17-26). When $|R| = k$ or PQ is empty, the sever stops the above process and returns R. Finally, the user decrypts R to get Ans, which contains the identifier for each node and is sorted by distance in non-descending order.

Algorithm 4: Forward Query Algorithm

Input: The $user$'s inputs are K and the query(v, w, k). The $sever$'s input is the encrypted index I
Output: The $sever$'s output is R and the $user$'s output is Ans

1 $user$ generates $T_1 = g(K_1, w||1)$, $T_2 = g(K_1, w||2)$, $T_3 = g(K_1, v||3)$ and $T_4 = g(K_1, v||4)$
2 $user$ sends $\tau = (T_1, T_2, T_3, T_4, k)$ to sever
3 $sever$ parses τ as (T_1, T_2, T_3, T_4, k)
4 $sever$ initializes a set $\widetilde{\Theta}(w)$ and a counter c=0
5 $sever$ sets $T_w^1 = g(T_1, c)$, $T_w^2 = h(T_2, c)$
6 **while** $\widetilde{\Theta}[T_w^1] \neq \perp$ **do**
7 $sever$ computes $V||T_u^1||T_u^2 = \widetilde{\Theta}[T_w^1] \oplus T_w^2$
8 $sever$ inserts $V||T_u^1||T_u^2$ into $\widetilde{\Theta}(w)$
9 $sever$ sets $c = c + 1$
10 $sever$ sets $T_w^1 = g(T_1, c)$, $T_w^2 = h(T_2, c)$
11 $sever$ initializes a set $\widetilde{\Delta}(T)$
12 **for** *each* $V||T_u^1||T_u^2 \in \widetilde{\Theta}(w)$ **do**
13 $sever$ initializes a set $\widetilde{\Psi}(u)$ and a counter c=0
14 $sever$ sets $T_{uv}^1 = g(T_u^1, c)$, $T_{uv}^2 = g(T_u^2, c)$
15 **while** $\widetilde{\Delta}[T_{uv}^1] \neq \perp$ **do**
16 $sever$ computes $V||D||\hat{D} = \widetilde{\Delta}[T_{uv}^1] \oplus T_{uv}^2$
17 $sever$ inserts $V||D||\hat{D}$ into $\widetilde{\Delta}(u)$
18 $sever$ sets $c = c + 1$
19 $sever$ sets $T_{uv}^1 = g(T_u^1, c)$, $T_{uv}^2 = g(T_u^2, c)$
20 $sever$ inserts $\widetilde{\Delta}(u)$ into $\widetilde{\Delta}(T)$
21 $sever$ initializes a set $\widetilde{\Delta}(v)$ and a counter c=0
22 $sever$ sets $T_v^1 = g(T_3, c)$, $T_v^2 = f(T_4, c)$
23 **while** $\widetilde{\Delta}[T_v^1] \neq \perp$ **do**
24 $sever$ computes $V||D||\hat{D} = \widetilde{\Delta}[T_v^1] \oplus T_v^2$.
25 $sever$ inserts $V||D||\hat{D}$ into $\widetilde{\Delta}(v)$.
26 $sever$ sets $c = c + 1$.
27 $sever$ sets $T_v^1 = g(T_3, c)$, $T_v^2 = f(T_4, c)$
28 $sever$ initializes a set C and a set R
29 **for** *each* $\widetilde{\Delta}(u) \in \widetilde{\Delta}(T)$ **do**
30 $sever$ inserts $(V, d_G vu)$ into C that $d_G vu$ contains $D_1||\hat{D_1}||D_2||\hat{D_2}$ where $(V_1||D_1||\hat{D_1}) \in \widetilde{\Delta}(v)$, $(V_2||D_2||\hat{D_2}) \in \widetilde{\Delta}(u)$, and $V_1 = V_2$
31 $sever$ sorts C according to $d_G vu$ and inserts the top-k item into R and returns R to the $user$
32 $user$ decrypts R with K
33 **return** decrypted value of R as Ans

Combining FQ and FBQ. Since FQ and FBQ are effective for low-frequency and high-frequency keywords respectively, a frequency threshold t needs to be set to effectively combine the two schemes. The setting of threshold t means that the t-most frequent keywords are processed by FBQ, and others are processed by FQ. The selection of the threshold on plaintext data is analyzed in detail in [10] ans it is considered that the top $\sqrt{|W|}$ keywords are expected to contain about half of the $|doc(V)|$ keyword occurrences. Similarly, in this paper, we set threshold t to $\sqrt{|W|}$, but specific analysis is needed for different datasets in order to make the combined scheme have the highest efficiency.

Extend to directed graphs. Our scheme can also support k-NK queries on directed graphs. First we adopt $2HL$ index Δ that supports directed graphs. The underlying Δ of a directed graph is divided into Δ_{in} and Δ_{out}. Therefore, BL index Ψ is built based only on Δ_{in}. We calculate the distance by Δ_{in} and Δ_{out} in FQ , and in FBQ we calculate by Δ_{out} and Ψ.

V. Ciphertext Sum Comparison Scheme

In this section, we introduce the ciphertext sum comparison scheme used in Section IV.

Algorithm 5: Forward Backward Query Algorithm

Input: The $user$'s inputs are K and the query (v, w, k). The $sever$'s input is the encrypted index I
Output: The $sever$'s output is R and the $user$'s output is Ans

$user$ generates $T_1 = F(K_1, w)$,$T_2 = g(K_1, v||3)$, $T_3 = g(K_1, v||4)$ and $T_4 = g(K_1, v||5)$
$user$ sends $\tau = (T_1, T_2, T_3, T_4, k)$ to sever
$sever$ parses τ as (T_1, T_2, T_3, T_4, k)
$sever$ initializes a set $\widetilde{\Delta}(v)$ and a counter c=0
$sever$ sets $T_v^1 = g(T_2, c)$, $T_v^2 = f(T_3, c)$, $T_v^3 = g(T_4, c)$
while $\widetilde{\Delta}[T_v^1] \neq \perp$ **do**
  $sever$ computes $U||D||\hat{D}||T_u^1 = \widetilde{\Delta}[T_v^1] \oplus T_v^2$
  $sever$ computes $T_u = T_u^1 \oplus T_v^3$
  $sever$ inserts $U||D||\hat{D}||T_u$ into $\widetilde{\Delta}(v)$
  $sever$ sets $c = c + 1$.
  $sever$ sets $T_v^1 = g(T_2, c)$, $T_v^2 = f(T_3, c)$, $T_v^3 = g(T_4, c)$
$sever$ initializes a double-ended priority queue PQ where the minimum item is represented by $*$ and the maximum item by $\#$ and a set R
for *each* $U_i||D_1||\hat{D_1}||T_{u_i} \in \widetilde{\Delta}(v)$ **do**
  $sever$ sets $(X_1, D2||\hat{D_2}) = KTnext(T_1, T_{u_i}, \infty)$
  if $(X_1, D_2||\hat{D_2}) \neq \perp$ **then**
    $sever$ executes $InsertPQ(U, X_1, D_1||\hat{D_1}||D_2||\hat{D_2})$
while $|R| < k$ *and* $PQ \neq \perp$ **do**
  $sever$ pops $(U^*, X_j^*, D_1^*||\hat{D_1^*}||D_2^*||\hat{D_2^*})$ from PQ
  $sever$ inserts X_j^* into R
  for *each* $(U_i, X_j, D_1^{'}||\hat{D_1^{'}}||D_2^{'}||\hat{D_2^{'}})$ *linked to* $X_j = X_j^*$ **do**
    while $j < k$ **do**
      $sever$ sets $j = j + 1$
      $sever$ sets $(X_j, D_2^{''}||\hat{D_2^{''}}) = KTnext(T_1, T_{u_i}, X_{j-1})$.
      if $(X_j, D_2^{''}||\hat{D_2^{''}}) \neq \perp$ *and* $X_j \notin R$ **then**
        $sever$ executes $InsertPQ(U_i, X_j, D_1^{'}||\hat{D_1^{'}}||D_2^{''}||\hat{D_2^{''}})$
        break.
$sever$ returns R to the $user$
$user$ decrypts R with K
return decrypted value of R as Ans

Algorithm 6: InsertPQ

Input: $U, X, D_1||\hat{D_1}||D_2||\hat{D_2}$
Output: PQ

if $\exists(U^{'}, X, D_1^{'}||\hat{D_1^{'}}||D_2^{'}||\hat{D_2^{'}}) \in PQ$ **then**
  if $CMP(D_1^{'}||\hat{D_1^{'}}||D_2^{'}||\hat{D_2^{'}}, D_1||\hat{D_1}||D_2||\hat{D_2}) == 0$ **then**
    replace $(U^{'}, X, D_1^{'}||\hat{D_1^{'}}||D_2^{'}||\hat{D_2^{'}})$ with $(U, X, D_1||\hat{D_1}||D_2||\hat{D_2})$
  link $(U, X, D_1||\hat{D_1}||D_2||\hat{D_2})$ with X
else
  if $|PQ| < k - |R|$ **then**
    $PQ = PQ \cup (U, X, D_1||\hat{D_1}||D_2||\hat{D_2})$
  else
    if $CMP(D_1^{\#}||\hat{D_1^{\#}}||D_2^{\#}||\hat{D_2^{\#}}, D_1||\hat{D_1}||D_2||\hat{D_2}) == 0$ **then**
      replace $(U^{\#}, X, D_1^{\#}||\hat{D_1^{\#}}||D_2^{\#}||\hat{D_2^{\#}})$ with $(U, X, D_1||\hat{D_1}||D_2||\hat{D_2})$

Given a k-NK query, we need to compare the different sum of two ciphertexts when calculating the shortest distance and obtaining the top-k value. Since the value is stored in ciphertext, we need a scheme to compare the different sum of two ciphertexts. There are some solutions can solve this problem, however they still face great limitations [16, 17, 19].

Algorithm 7: Comparison Algorithm CMP

Input: $D_1||\hat{D_1}||D_2||\hat{D_2}$ and $D_3||\hat{D_3}||D_4||\hat{D_4}$
Output: 0 or 1

if $D_1 \leq D_3$ *and* $D_2 \leq D_4$ *or* $D_1 \leq D_4$ *and* $D_2 \leq D_3$ **then**
  return 1
else if $D_3 \leq D_1$ *and* $D_4 \leq D_2$ *or* $D_3 \leq D_2$ *and* $D_4 \leq D_1$ **then**
  return 0
else
  Assume that $D_1 \leq D_3 \leq D_4 \leq D_2$
  if $D_4 \leq \hat{D_2}$ *or* $\hat{D_4} \leq D_1$ *and* $D_3 \leq \hat{D_2}$ **then**
    return 0
  else
    return 1

The scheme [16] based on OPE has been proved to be unsafe. The comparison scheme proposed by Lu et al. [17] requires multiple additional interactions and is very inefficient due to the use of homomorphic encryption and the scheme [19] has low accuracy. To keep the search process efficient and accurate, we need a secure scheme that can compare ciphertexts sum.

When we want to compare the sum of a and b and the sum of c and d, if $a < c$ and $b < d$ (or $a > c$ and $b > d$), we know $a + b < c + d$ (or $a + b > c + d$) obviously. Based on this observation, most of the candidates can be filtered out. In order to improve the accuracy of the results, our scheme is introduced for comparison in the case of $a < c < d < b$. Our scheme takes any safe order-revealing encryption (ORE) algorithm to encrypt the distance value. In an ORE scheme, the encryption algorithm produces ciphertexts that preserve the order of their plaintexts. Our scheme is defined as follows:

Definition 4. *(Order Encryption (OE)). Given a private key K_{ORE} of the ORE scheme, a private key K and a distance d, $OE(K_{ORE}, d, K)$ will return $(enc, e\hat{n}c)$, where $enc = ORE.Enc(Kd + r_1)$ and $e\hat{n}c = ORE.Enc(\frac{Kd - r_2}{2})$. r_1 and r_2 are two random numbers that satisfy $r_1 \leq r_2$ and $r_2 \ll K$, where the symbol "$\ll$" means much greater than.*

By listing, we know that for the case of $a < c < d < b$, there are 14 kinds of sorting about $a, b, c, d, a/2, b/2, c/2, d/2$ such as $a/2 < a < c/2 < c < d/2 < d < b/2 < b$ and so on. We find that when satisfying $d < b/2$, or both $d/2 < a$ and $c < b/2$, there must be $a+b > c+d$. The above conditions are satisfied in 6 out of 14 cases, so the probability of comparing $a + b$ and $c + d$ with certainty is $\frac{3}{7}$. In theory, there is a $\frac{2}{3} + \frac{1}{3} * \frac{3}{7} = 80.98\%$ chance that our scheme will produce an exact comparison in each comparison. The existence of K, r_1 and r_2 can expand ciphertext domain and make inference attack difficult.

Now we prove that the existence of K, r_1 and r_2 will not affect the accuracy of the scheme. For four integers a, b, c and d, if $d < b/2$, then $Kd + r_{d_1} - \frac{Kb - r_{b_2}}{2} = \frac{K(2d-b) + (2r_{d_1} + r_{b_2})}{2}$. Becuase $2d - b < 0, r_{d_1} \leq r_{b_2}$ and $r_{b_2} \ll K$, we have $Kd + r_{d_1} < \frac{Kb - r_{b_2}}{2}$. Therefore, $enc_d = ORE.Enc(Kd + r_{d_1}) < e\hat{n}c_b = ORE.Enc(\frac{Kb - r_{b_2}}{2})$, which means that the result of the comparison on encrypted data is the same as the result on plaintext. In the case of $d/2 < a$ and $c < b/2$, we can prove

that $enc_d < e\hat{n}c_a$ and $enc_c < e\hat{n}c_b$ in the same way, and skip the proof here.

When we want to compare D_1+D_2 and D_3+D_4, as shown in Algorithm 7, we judge whether $D_1 \leq D_3$ and $D_2 \leq D_4$ (or $D_3 \leq D_1$ and $D_4 \leq D_2$) (lines 1-4) to do the first filter. If it fails, $\hat{D_1}$, $\hat{D_2}$, $\hat{D_3}$ and $\hat{D_4}$ are used for the second filter (lines 6-10). The CMP returns 1 if $D_1 + D_2 \leq D_3 + D_4$, otherwise returns 0.

VI. Security Analysis

This section presents the security analysis on Aton. We first describe the leakage functions of it and then prove that our scheme is secure under the CQA2-security model.

Setup leakage. The leakage function $\mathcal{L}_{Setup}$ leaks information from static index I of graph G, where $I = (\widetilde{\Theta}, \widetilde{\Delta}, \widetilde{\Psi}, \widetilde{\Upsilon})$. Given a graph G, the leakage function

$$\mathcal{L}_{Setup} = (|\widetilde{\Theta}|, |\widetilde{\Delta}|, n, t_v, t_v^{'}, Inf_{\widetilde{\Upsilon}(v)}, Ord)$$

where $|\widetilde{\Theta}|$ and $|\widetilde{\Delta}|$ are the size of LF and SL indexes consists of all entities in $\widetilde{\Theta}$ and $\widetilde{\Delta}$, n is the total number of nodes in G, t_v and $t_v^{'}$ are the size of $\widetilde{\Psi}(v)$ and $\widetilde{\Upsilon}(v)$, $Inf_{\Upsilon(v)}$ is a list that contains the number of 1 of each node $u \in \Upsilon(v)$ and Ord is a $n \times n$ matrix represents the distance order relationship leaked in the $\widetilde{\Psi}$ with each entry (i, j) being 2, 1, -1 or -2. For each pair of distances $d_i||\hat{d_i}$ and $d_j||\hat{d_j}$, its order relationship of $d_i \leq \hat{d_j}$, $\hat{d_j} < d_i \leq d_j$, $d_j > d_i > \hat{d_j}$ and $d_i > d_j$ can be represented by 2, 1, -1 and -2, respectively. Since we use the private key K and random numbers to confuse distances, the server cannot infer the real values from their order information revealed by Ord.

Note that we don't carry out second-round encryption on SBL index, so Ord gets leaked. However, this leak doesn't reveal the order information of SL index because we encrypt the nodes in a different way when constructing SBL and SL indexes (line 7 of Algorithm 2 and 3).

Query leakage. The query leakage $\mathcal{L}_{Query}$ consists of the query pattern leakage and the sketch pattern leakage. The query pattern leakage reveals the repetition information among all queries and the sketch pattern leakage reveals information between sketches associated with query nodes. Note that k in each k-NK query only affects the size of the returned set R and doesn't reveal other information, so we don't discuss it.

Definition 5. *(QUERY PATTERN LEAKAGE).* Let $\boldsymbol{q} = (q_1, q_2, \ldots, q_r)$ be a non-empty query sequence containing r queries. Set each query q_i to (v_i, w_i, k). For two queries $q_i = (v_i, w_i, k)$ and $q_j = (v_j, w_j, k)$, define $Sim(q_i, q_j) = (v_i = v_j, w_i = w_j)$, i.e., whether each q_i and q_j matches. The query pattern leakage function $\mathcal{L}_{QP}(\boldsymbol{q})$ returns an $r \times r$ matrix, in which each entry $(i,\ j)$ equals $Sim(q_i, q_j)$. Note that $\mathcal{L}_{QP}(\boldsymbol{q})$ does not leak the identities of the queried nodes.

Definition 6. *(SKETCH PATTERN LEAKAGE).* Given an index $I = (\widetilde{\Theta}, \widetilde{\Delta}, \widetilde{\Psi}, \widetilde{\Upsilon})$ and a query $q = (v, w, k)$, the sketch pattern leakage function $\mathcal{L}_{SP}(I, q)$ is defined as $(\Gamma, \Omega, \Lambda)$. Γ is a quad containing lists A, B, C, and D where A is the $doc(v)$ about w, B is the sketch associated to v, and C and D are entries in $\widetilde{\Psi}(u)$ and $\widetilde{\Upsilon}(u)$ associated to B. Ω is a pair (Y, Z), where $Y = \text{SKE}(K_1, v||2){:}(v, d, \hat{d}) \in \widetilde{\Delta}$ and $Z = \text{SKE}(K_1, v||2){:}(v, d, \hat{d}) \in \widetilde{\Delta}$ are multi-sets. Λ is a (U, X) pair, where $U = \text{SKE}(K_1, u||2){:}(u, d, \hat{d}) \in \widetilde{\Delta}$ and $X = \text{SKE}(K_1, u||1){:}(u, d, \hat{d}) \in \widetilde{\Psi}$ are multi-sets.

Thus, $\mathcal{L}_{Query} = (\mathcal{L}_{QP}(\boldsymbol{q}), \mathcal{L}_{SP}(I, q))$.

Theorem 2. *If the cryptography primitives g, h, f, F, ORE and the SKE are secure, then the proposed graph encryption scheme $\Pi = (KeyGen, Setup, Query)$ is $(\mathcal{L}_{Setup}, \mathcal{L}_{Query})$-secure against the adaptive chosen-query attack.*

Proof. The key idea is constructing a simulator $\mathcal{S}$ who uses $\mathcal{L}_{Setup}$ and $\mathcal{L}_{Query}$ to construct a fake encrypted index $I^* = (\widetilde{\Theta}^*, \widetilde{\Delta}^*, \widetilde{\Psi}^*, \widetilde{\Upsilon}^*)$ and a list of query q^*. If no PPT adversary $\mathcal{A}$ can distinguish between the two games **Real** and **Ideal**, we can say that our graph encryption scheme is $(\mathcal{L}_{Setup}, \mathcal{L}_{Query})$-secure against the adaptive chosen-query attack.

Simulating I^*. $\mathcal{S}$ first generates fake secret keys K^*. $\mathcal{S}$ randomly selects R_1 as the number of low-frequency keywords. Given leakage function $\mathcal{L}_{Setup}$, for each keyword w_i ($1 \leq i \leq R_1$), $\mathcal{S}$ randomly chooses r_i for w_i with $\sum_1^{R_1} r_i = |\widetilde{\Theta}|$ and samples $l_i \leftarrow \{0,1\}^\lambda$ and $\eta_i \leftarrow \{0,1\}^\lambda$ uniformly without repetition. Then for all $0 \leq i < r_i$, $\mathcal{S}$ computes $l_{r_i} = g(l_i, r_i)$ and $\eta_{r_i} = h(\eta_i, r_i)$, where g and h are two particular pseudo-random functions. $\mathcal{S}$ encrypts each node v in the V_{w_i} of w_i by computing $V^* = SKE(K_3^*, v||1)$, $T_v^{1*} = g(K_1^*, v||3)$, and $T_v^{2*} = g(K_1^*, v||4)$. Set $\widetilde{\Theta}^*[l_{r_i}] = \eta_{r_i} \oplus (V^*||T_v^{1*}||T_v^{2*})$. Similarly, for each node v_i ($1 \leq i \leq n$), $\mathcal{S}$ randomly chooses $r_i^{'}$ for v_i with $\sum_1^n r_i^{'} = |\widetilde{\Delta}|$ and samples $l_i^{'} \leftarrow \{0,1\}^\lambda$, $\eta_i \leftarrow \{0,1\}^\lambda$ and $\rho_i \leftarrow \{0,1\}^\lambda$ uniformly without repetition. For all $0 \leq i < r_i^{'}$, $\mathcal{S}$ computes $l_{r_i}^{'} = g(l_i^{'}, r_i^{'})$, $\eta_{r_i}^{'} = f(\eta_i^{'}, r_i^{'})$ and $\rho_{r_i}^{'} = h(\rho_i^{'}, r_i^{'})$. For each vertex u in the $L(v_i)$, $\mathcal{S}$ encrypts it by computing $U^* = SKE(K_3^*, u||2)$, and generates a random distance value d and obtains ciphertexts $D^*||\hat{D}^*$ by OE. It computes $T_u^{1*} = g(K_1^*, u||6)$ and sets $\widetilde{\Delta}^*[l_{r_i}^{'}] = (U^*||D^*||\hat{D}^*||(T_u^{1*} \oplus \rho_{r_i}^{'})) \oplus \eta_{r_i}^{'}$. Now, $\mathcal{S}$ has obtained fake SKL^* and $\widetilde{\Delta}^*$.

To simulate fake $\widetilde{\Psi}^*$ and $\widetilde{\Upsilon}^*$, for each node u_i ($1 \leq i \leq n$), $\mathcal{S}$ computes $T_u^{1*} = g(K_1^*, u||6)$. For all $0 \leq j < t_{u_i}$ it encrypts each node $x_j \in BL(u_i)$ by computing $X^* = SKE(K_3^*, x_j||1)$. Then, it generates integers d and obtains ciphertexts $D^*||\hat{D}^*$ by OE according to Ord. Set $\widetilde{\Psi}^*[T_{u_i}^{1*}] = Arr[u_i]$ where $Arr[u_i]$ is an array that holds t_{u_i} ciphertexts $X^*||D^*||\hat{D}^*$. Similarly, for each node x_j, $\mathcal{S}$ computes $H_{x_j} = \bigvee_{o=1}^{num} F(w_o)$ where num is obtained from $Inf_{\Upsilon(u_i)}$. Set $\widetilde{\Upsilon}^*[T_{u_i}^{1*}] = Arr^{'}[u_i]$ where $Arr^{'}[u_i]$ is an array that holds $t_{u_i}^{'}$ ciphertexts H_{x_j}. $\mathcal{S}$ finally obtains the fake $I^* = \{SKL^*, \widetilde{\Delta}^*, \widetilde{\Psi}^*, \widetilde{\Upsilon}^*\}$.

Simulating q^*. Given the leakage function $\mathcal{L}_{Query} = (\mathcal{L}_{QP}(\boldsymbol{q}), \mathcal{L}_{SP}(I, q), \mathcal{L}_{DP}(I, q))$, $\mathcal{S}$ first checks if either of node v or keyword w appeared in any previous query. If v or w appeared previously, $\mathcal{S}$ sets $Token_v$ or $Token_w$ to the values that were previously used. Otherwise, it computes $Token_v$ based on some previously unused $l_i^{'}$, $\eta_i^{'}$ and $\rho_{r_i}^{'}$ and then remembers the association among $l_i^{'}$, $\eta_i^{'}$, $\rho_{r_i}^{'}$ and v. It

TABLE III: The Graph Datasets Used in Our Experiments

Dataset	$\|V\|$	$\|E\|$	$\|doc(V)\|$	$\|W\|$
D1: ego-Facebook	4,039	88,234	74,522	1,311
D2: Twith-DE	9,498	153,138	193,725	3,169
D3: Facebook LPPN	22,470	171,002	314,583	4,714

computes $Token_w$ according to some previously unused l_i and η_i. It then remembers the association among $l_i^{'}$, $\eta_i^{'}$, and w. Given the query token $(Token_v, Token_w)$, $\mathcal{S}$ first checks if the query has been queried before. If yes, $\mathcal{S}$ returns the value that was previously used as the query result. Otherwise, $\mathcal{S}$ simulates the query procedure according to the information in $\mathcal{L}_{SP}(I, q)$ and returns the query result.

Since the cryptography primitives g, h, f, F, ORE and SKE are secure, the fake index structure I^* and the query sequence q^* are indistinguishable from the real ones. Therefore, no PPT adversary $\mathcal{A}$ can distinguish the two games **Real** and **Ideal**. Thus, we have

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

where $negl(\lambda)$ is a negligible function. □

VII. Performance Evaluation

In this section, we present the evaluation of the proposed Aton through experiments on real-world datasets.

A. Experimental Setup

Testbed. All test programs are written in C++ and run on a 64-bit Windows machine with Intel Core i7-9700K (3.60GHz) CPU and 32GB RAM. We construct the $2HL$ index mentioned in [20], the BL and KT indexes mentioned in [10]. The ORE in our implementation follows the methods described in [21]. SKE and PRFs are implemented using AES and HMAC from the OpenSSL library. To ensure basic safety, we set the security parameter $\lambda = 128$.

Graph datasets. The real datasets used in our experiments are listed in Table III, which are publicly available from the Standford SNAP website[2]. Ego-Facebook is a social network, where each node with keywords such as ID represents a user and each edge represents friendship between users. Twith-DE is a twitch user-user network of gamers where node keywords are extracted based on the player's location and streaming habits, etc. Facebook LPPN is a page-page graph, where nodes represent pages and node keywords are extracted from the site descriptions. We generate a distance for each edge, the value of which follows a uniform distribution between 1 and 100.

Query sets. For each dataset, we randomly generate 500 queries. In a query (v, w, k), the node v is randomly picked from the node set, the keyword w is selected following the keyword distribution and k varies from 1 to 128. Take social network as an example, people are more concerned about several people nearby, so setting k from 1 to 128 is enough.

Methods to compare. First, we compare Aton with the one over unencrypted graphs. Since GREK [16] and PPknk [18] are the only two methods that can directly solve the k-NK querying problem over encrypted graphs, we make an overall comparison between Aton and them. To achieve shortest distances querying over encrypted graphs, ENPS proposes an OPE scheme, which solves the problem of ciphertext sum comparison just like OE in Aton. Therefore, we compare the two ciphertext sum comparison schemes. To enable ENPS to solve k-NK querying, we replace the index structure of ENPS with that of Aton while preserving OPE.

[2]http://snap.stanford.edu/data/

TABLE IV: Comparison of Index Construction Overheads

Metrics		D1	D2	D3
Plain Graph	Size (MB)	15.16	67.10	493.93
	Time (mins)	3.24	53.72	686.70
GREK [16]	Size (MB)	19.716	59.41	698.25
	Time (mins)	3.75	55.91	694.10
PPknk [18]	Size (MB)	48.37	200.89	1,304.83
	Time (mins)	3.59	54.02	691.13
ENPS [19]	Size (MB)	33.32	105.63	708.67
	Time (mins)	4.97	58.69	710.51
Aton	Size (MB)	35.15	110.62	749.87
	Time (mins)	5.45	60.29	717.51

B. Evaluation of Index Setup

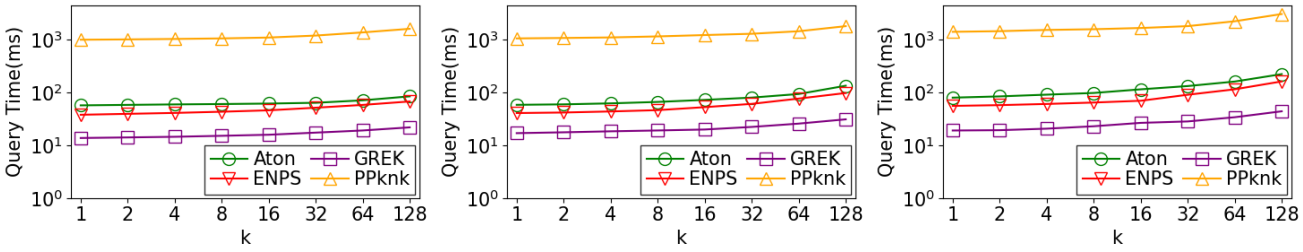
The index size and construction time for each scheme are shown in Table IV.

Index size. The index size of GREK, Aton and ENPS vary according to the value of thresholds. Through continuous experiments, we select the best thresholds. For different datasets, we set the thresholds for GREK as 120, 150 and 200, and the Aton and ENPS as 30, 40 and 50. According to Table IV, we can see that the index size of PPknk is the largest. The index size of Aton is larger than GREK and ENPS, because additional $|\Psi(v)|$-1 space is required when building the BL index $\Psi(v)$ into the keyword tree index. It is noted that storage problem is no longer a bottleneck now with the development of the cloud computing.

Construction time. The index construction time of Plain Graph is the index generation time, where building the $2HL$ index takes up most of the time. The most important observation is that the construction time of Aton is slightly higher than the one of Plain Graph. PPknk has the shortest construction time, since it uses the shortest path tree answering distance queries. For this reason, it cannot support accurate queries. Aton and ENPS take longer than GREK because the OPE algorithm designed in GREK is efficient at the expense of security. Compared to ENPS, Aton takes longer because we encrypt each distance value twice with ORE, and ENPS encrypts each distance once. Since the index construction is offline, it is acceptable to take longer to ensure security.

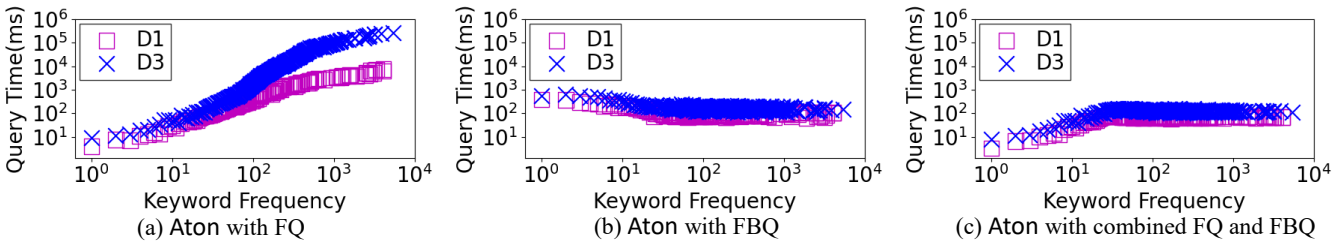
C. Evaluation of Query

Query efficiency. The query time includes the query token generation time and the interval between the user submitting the token and receiving the result. The query times under different k for each scheme are shown in Fig. 7. PPknk has the longest query time due to the use of the Paillier cryptosystem. With the increase of k, the query time of Aton grows very



(a) Query time on D1 (b) Query time on D2 (c) Query time on D3

Fig. 7: Average Query Time under different values of k.



(a) Aton with FQ (b) Aton with FBQ (c) Aton with combined FQ and FBQ

Fig. 8: Query time with different keyword frequencies.

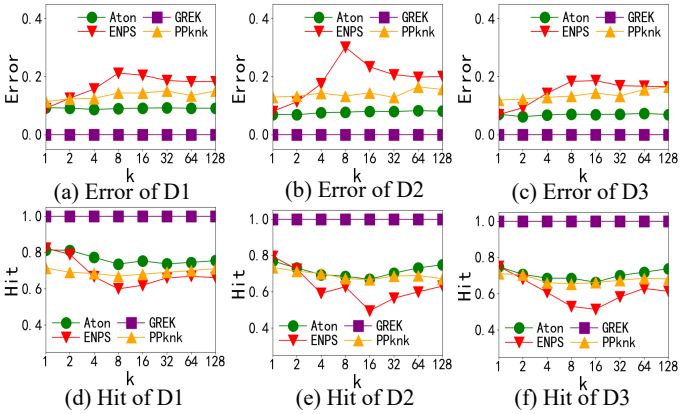
slowly. Aton can complete in one second even when k= 128, which shows the effectiveness of Aton.

Fig. 8 shows the query time of Aton with different keyword frequencies under k=32. The query time for FQ increases linearly with keyword frequency, so FQ is less efficient when the keyword frequency is high. The query time for FBQ is stable but FBQ cannot process low-frequency keywords as quickly as FQ. For hight-frequency keywords, FBQ is slower than FQ-FBQ because it has a larger keyword set in the index. Thus, using FQ or FBQ alone don't run as fast as FQ-FBQ.

Query accuracy. We set two parameters to evaluate the accuracy of the scheme: $error$ and hit. For a k-NK query, we set $R = \{(u_1, d_1), (u_2, d_2), ..., (u_k, d_k)\}$ as the exact result, where the distance values are in non-descending order, and $R^{'} = \{(u_1^{'}, d_1^{'}), (u_2^{'}, d_2^{'}), ..., (u_k^{'}, d_k^{'})\}$ as the approximate result. The $error$ represents the average error between R and $R^{'}$ and hit represents the sorting accuracy of distance in $R^{'}$, which are defined as:

$$error = \frac{\sum_{i=1}^{k} |(d_i^{'} - d_i)/d_i|}{k}, \quad hit = \frac{\sum_{i=1}^{k} (d_i^{'} \leq d_i)?1:0}{k}$$

The smaller the $error$ is, the closer the value of approximate result is to the exact result, while the larger the hit is, the closer the sorting of the approximate result is to the exact result. Fig. 9 shows the $error$ and hit of each scheme with varying k. The $error$ and hit of GREK are 0 and 1, but GREK sacrifices security. We set the distance interval itv of ENPS to 8. It can be found that the $error$ of Aton is below 0.1, and the hit is around 0.7 for all k values. Compared with ENPS and PPknk, Aton has higher hit and lower $error$ in different datasets, which means that Aton has higher accuracy and stability. The inaccuracy of PPknk is due to the underlying index it chooses, regardless of the encryption method. It should be noted that increasing itv can improve the accuracy of ENPS, but more privacy will be disclosed at the same time.



(a) Error of D1 (b) Error of D2 (c) Error of D3

(d) Hit of D1 (e) Hit of D2 (f) Hit of D3

Fig. 9: $Error$ and Hit under different values of k.

VIII. Related Work

In this section, we briefly review the related work on k-NK querying over plain graphs and encrypted graphs.

In the field of plain graphs, Bahmani et al. [9] explore the approximate k-NK query on undirected graphs through the shortest path tree. Jiang et al. [10] propose a precise algorithm to solve the k-NK problem by using an exact $2HL$ index and the keyword tree. V-Tree [11] accelerates the query by dividing the large rode network into interconnected regional subnets. However, these schemes do not consider security issues, and cannot be directly applied to encrypted graphs.

Querying encrypted graphs has drawn much attention recently. Some work [12] study the k-NK query for encrypted geospatial data. However, these schemes are implemented based on coordinates and privacy protection is mostly achieved through technologies such as local sensitive hash and location confusion, which are only applicable to spatial data and cannot be generalized to general graphs.

Wu et al. [13] realize adjacency search that supports synonym query but does not support the specified k nearest nodes

query. Arthy et al. [14] propose a k-NK query scheme for encrypted graphs of social media, but this scheme can only be used on undirected graphs with no weight. Another scheme [15] performs secure query operations on transitive closure matrices of directed graphs, but does not support operations on weighted graphs. Liu et al. [16] consider k-NK query over graph for privacy protection. However, their scheme has defects on security, where the order-preserving encryption scheme constructed in this scheme is not secure and is easy to be cracked to obtain the key [26]. Teng et al. [18] propose a dual-cloud framework to solve the k-NK problem, but the efficiency is low due to the use of Paillier cryptosystem.

Although there are respectable studies on k-NK queries, a privacy-preserving and efficient solution is still desirable. In this paper, we propose a secure and efficient encryption scheme for k-NK queries on directed weighted graphs which can produce more accurate results in linear time compared with existing solutions.

IX. Conclusion

In this paper, we presented a novel graph encryption scheme Aton that supports k-NK queries over encrypted graphs, which can answer a query efficiently with high accuracy. Based on the existing order-revealing encryption scheme, we proposed a practical ciphertext sum comparison scheme to select the top-k answers that satisfy the keyword constraints. This scheme can also be used in other scenarios that need to compare the sum of ciphertexts. Through rigorous security analysis, our scheme was proved to be CQA-2 secure. We evaluated Aton on three real-world graph datasets, and the experimental results demonstrated its accuracy and effectiveness. In future work, we will focus on designing new modules to further improve query accuracy.

Acknowledgments

This work is partially supported by National Key RD Program of China with No. 2020YFB1006101, Beijing Nova Program with No. Z201100006820006, NSFC Projects with No. 61972039, 61932016 and 61872041, Beijing Natural Science Foundation with No. 4192050, Open Research Projects of Zhejiang Lab under Grant 2020AA3AB04, China National Funds for Distinguished Young Scientists with No. 61825204, Beijing Outstanding Young Scientist Program with No. BJJWZYJH01201910003011.

References

[1] G. V. Demirci, H. Ferhatosmanoglu, and C. Aykanat, "Cascade-aware partitioning of large graph databases," *The VLDB Journal*, vol. 28, no. 3, p. 329–350, Jun. 2019.

[2] F. Petroni, L. Querzoni, K. Daudjee, S. Kamali, and G. Iacoboni, "Hdrf: Stream-based partitioning for power-law graphs." New York, NY, USA: Association for Computing Machinery, 2015, p. 243–252.

[3] "Powergraph," http://www.powergraph.ru/.

[4] "Snap," http://snap-graph.sourceforge.net/.

[5] M. Qiao, L. Qin, H. Cheng, J. X. Yu, and W. Tian, "Top-k nearest keyword search on large graphs," *Proc. VLDB Endow.*, vol. 6, no. 10, p. 901–912, Aug. 2013.

[6] V. Chang, Y.-H. Kuo, and M. Ramachandran, "Cloud computing adoption framework: A security framework for business clouds," *Future Generation Computer Systems*, vol. 57, pp. 24 – 41, 2016.

[7] M. Shen, X. Tang, L. Zhu, X. Du, and M. Guizani, "Privacy-preserving support vector machine training over blockchain-based encrypted iot data in smart cities," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7702–7712, 2019.

[8] M. Shen, Y. Deng, L. Zhu, X. Du, and N. Guizani, "Privacy-preserving image retrieval for medical iot systems: A blockchain-based approach," *IEEE Network*, vol. 33, no. 5, pp. 27–33, 2019.

[9] B. Bahmani and A. Goel, "Partitioned multi-indexing: Bringing order to social search," in *Proceedings of the 21st International Conference on World Wide Web*, New York, NY, USA, 2012, p. 399–408.

[10] M. Jiang, A. W. Fu, and R. C. Wong, "Exact top-k nearest keyword search in large networks," in *Proceedings of the 2015 ACM SIGMOD, Melbourne, Victoria, Australia*. ACM, 2015, pp. 393–404.

[11] B. Shen, Y. Zhao, G. Li, W. Zheng, Y. Qin, B. Yuan, and Y. Rao, "V-tree: Efficient knn search on moving objects with road-network constraints," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, 2017, pp. 609–620.

[12] L. Chen, Y. Li, J. Xu, and C. S. Jensen, "Direction-aware why-not spatial keyword top-k queries," in *33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017*, 2017, pp. 107–110.

[13] B. Wu, Z. Zhao, Z. Cui, Z. Mei, and Z. Wu, "Secure and efficient adjacency search supporting synonym query on encrypted graph in the cloud," *IEEE Access*, vol. 7, pp. 133 716–133 724, 2019.

[14] R. Arthy, E. Daniel, T. G. Maran, and M. Praveen, "A hybrid secure keyword search scheme in encrypted graph for social media database," in *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)*, 2020, pp. 1000–1004.

[15] S. Ramezanian, T. Meskanen, and V. Niemi, "Privacy preserving shortest path queries on directed graph," in *22nd Conference of Open Innovations Association, FRUCT 2018, Jyväskylä, Finland, May 15-18, 2018*. IEEE, 2018, pp. 217–223.

[16] C. Liu, L. Zhu, and J. Chen, "Graph encryption for top-k nearest keyword search queries on cloud," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 4, pp. 371–381, 2017.

[17] W. Lu, J. Zhou, and J. Sakuma, "Non-interactive and output expressive private comparison from homomorphic encryption," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ser. ASIACCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 67–74.

[18] Y. Teng, X. Cheng, S. Su, and R. Bi, "Privacy-preserving top-k nearest keyword search on outsourced graphs," in *2016 IEEE Trustcom/BigDataSE/ISPA*, 2016, pp. 815–822.

[19] C. Liu, L. Zhu, X. He, and J. Chen, "Enabling privacy-preserving shortest distance queries on encrypted graph data," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2018.

[20] T. Akiba, Y. Iwata, and Y. Yoshida, "Fast exact shortest-path distance queries on large networks by pruned landmark labeling," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD'13. New York, NY, USA: Association for Computing Machinery, 2013, p. 349–360.

[21] K. Lewi and D. J. Wu, "Order-revealing encryption: New constructions, applications, and lower bounds," ser. CCS'16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1167–1178.

[22] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *ASIACRYPT 2010*, ser. Lecture Notes in Computer Science, M. Abe, Ed., vol. 6477. Springer, 2010, pp. 577–594.

[23] M. Shen, B. Ma, L. Zhu, X. Du, and K. Xu, "Secure phrase search for intelligent processing of encrypted data in cloud-based iot," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1998–2008, 2018.

[24] X. Meng, S. Kamara, K. Nissim, and G. Kollios, "Grecs: Graph encryption for approximate shortest distance queries," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS'15. New York, NY, USA: Association for Computing Machinery, 2015, p. 504–517.

[25] M. Shen, B. Ma, L. Zhu, R. Mijumbi, X. Du, and J. Hu, "Cloud-based approximate constrained shortest distance queries over encrypted graphs with privacy protection," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 4, pp. 940–953, 2018.

[26] D. Liu and S. Wang, "Programmable order-preserving secure index for encrypted database query," in *2012 IEEE Fifth International Conference on Cloud Computing*, 2012, pp. 502–509.