Privacy-Preserving Top-k Query Processing in Distributed Systems

Sakina Mahboubi[1], Reza Akbarinia[2], and Patrick Valduriez[2]

[1] University of Batna, Algeria
[2] INRIA & LIRMM, Univ. Montpellier, France
firstname.lastname@inria.fr

Abstract. We consider a distributed system that stores user sensitive data across multiple nodes. In this context, we address the problem of privacy-preserving top-k query processing. We propose a novel system, called SD-TOPK, which is able to evaluate top-k queries over encrypted distributed data without needing to decrypt the data in the nodes where they are stored. We implemented and evaluated our system over synthetic and real databases. The results show excellent performance for SD-TOPK compared to baseline approaches.

Keywords: Privacy Preserving · Top-k Query · Distributed System.

1 Introduction

Top-k queries are important for many centralized and distributed applications such as information retrieval [32], sensor networks [38], data stream management systems [35], crowdsourcing [7], spatial data analysis [29], social networks [16], etc. A top-k query allows the user to get the k data items that are most relevant to the query.

We consider a distributed system where users can outsource their *sensitive* data and issue top-k queries. The user data are encrypted (for privacy reasons) and distributed (for performance reasons) across multiple nodes. In this context, we address the problem of privacy-preserving top-k query processing.

Privacy preserving top-k query processing is a critical requirement for some distributed applications that outsource *sensitive* data. For example, consider a university that outsources the students database in a public cloud, in Infrastructure-as-a-Service (IaaS) mode, with non-trusted nodes. The database is vertically partitioned (for performance reasons) and encrypted. Then, an interesting top-k query over the encrypted distributed data is the following: return the k students that have the worst averages in some given courses.

There are different approaches for processing top-k queries over *plaintext* (non encrypted) data. One of the best known approaches is TA [14] that works on sorted lists of attribute values. However, there is no efficient solution capable of evaluating efficiently top-k queries over encrypted data in distributed systems with non-trusted nodes.

A naive solution is to retrieve the encrypted database from the distributed system to the user machine that keeps the secret keys, decrypt it, and then evaluate the top-k query over *plaintext* data. This solution is not efficient, because it does not allow us to take advantage of the distributed system power for evaluating queries.

In this paper[3], we propose a system, called SD-TOPK (Secure Distributed TOPK), that encrypts and stores user data in a distributed system, and is able to evaluate top-k queries over the encrypted data. SD-TOPK comes with a novel top-k query processing algorithm that finds a set of encrypted data that is proven to contain the top-k data items. This is done without having to decrypt the data in the nodes where they are stored. In addition, we propose a powerful filtering algorithm that removes the false positives as much as possible without data decryption. We implemented and evaluated the performance of our system over synthetic and real databases. The results show excellent performance for SD-TOPK compared to TA-based approaches. They show the efficiency of our filtering algorithm that eliminates almost all false positives in the distributed system, and reduces significantly the communication cost between the distributed system and the user.

The rest of the paper is organized as follows. Section 2 gives the problem definition. Section 3 describes the architecture of SD-TOPK system. In Section 4, we present two TA-based algorithms for top-k query processing over encrypted data. In Section 5, we describe the SD-TOPK system, and analyze its security in Section 6. Section 7 presents the performance evaluation results. Section 8 discusses related work, and Section 9 concludes.

2 Background and Problem Definition

In this section, we give a background about top-k queries, and define the problem which we address.

2.1 Top-k Queries

By a top-k query, the user specifies a number k, and the system should return the k most relevant answers. The relevance degree of the answers to the query is determined by a *scoring function*. A common method for efficient top-k query processing is to run the algorithms over *sorted lists* (also called *inverted lists*) [14]. Let us define them formally.

Let D be a set of n data items, then the sorted lists are m lists $L_1, L_2, ..., L_m$, such that each list L_i contains every data item $d \in D$ in the form of a pair $(id(d), s_i(d))$ where $id(d)$ is the identification of d and $s_i(d)$ is a value that denotes the *local score* (attribute value) of d in L_i. The data items in each list L_i are sorted in descending order of their local scores. For example, in a relational table, each sorted list represents a sorted column of the table where the local score of a data item is its attribute value in that column.

[3] This journal paper is a major extension of [23].

Let f be a scoring function given by the user in the top-k query. For each data item $d \in D$ an *overall score*, denoted by $ov(d)$, is calculated by applying the function f on the local scores of d. Formally, we have $ov(d) = f(s_1(d), s_2(d), ..., s_m(d))$.

The result of a top-k query is the set of k elements that have the highest overall scores among all elements of the database. In this work, we assume that the scoring function is in the class of linear functions with positive coefficients (denoted by $LFPC$).

The sorted lists model can be used for top-k query processing in many applications. For example, suppose we want to find the top-k tuples in a relational table according to some scoring function over its attributes. To answer such query, it is sufficient to have a sorted list for the values of each attribute, and return the k tuples whose overall scores in the lists are the highest.

2.2 Distributed System and Adversary Model

We suppose that the sorted lists are stored in the nodes of a *distributed system*. We make no specific assumption about the distributed system architecture which can be very general, *e.g.*, a cluster of nodes. Formally, let P be the set of the nodes in the distributed system. Each sorted list L_i is kept in a node $p \in P$. We call p the *owner* of L_i.

We consider the *honest-but-curious* adversary model for the nodes of the distributed system. In this model, the adversary is inquisitive to learn the sensitive data without introducing any modification in the data or protocols. This model is widely used in many privacy preserving solutions [21].

2.3 Problem Statement

The problem we attack in this paper is top-k query processing over encrypted data in distributed systems.

Let D be a database composed of n data items. We want to encrypt the data items contained in D, and store the encrypted data items in a distributed system. Then, our goal is to develop a distributed algorithm A that given any top-k query q (including a scoring function f) returns the k data items that have the highest overall scores with regard to f. This should be done without decrypting the data items in the nodes of the distributed system, while minimizing the response time and the communication cost of the query execution.

3 SD-TOPK System Architecture

The architecture of SD-TOPK has two main components (see Figure 1):

- **Trusted client.** It is responsible for encrypting the user data, decrypting the results and controlling the user accesses. The security keys used for data encryption/decryption are managed by this part of the system. When a query

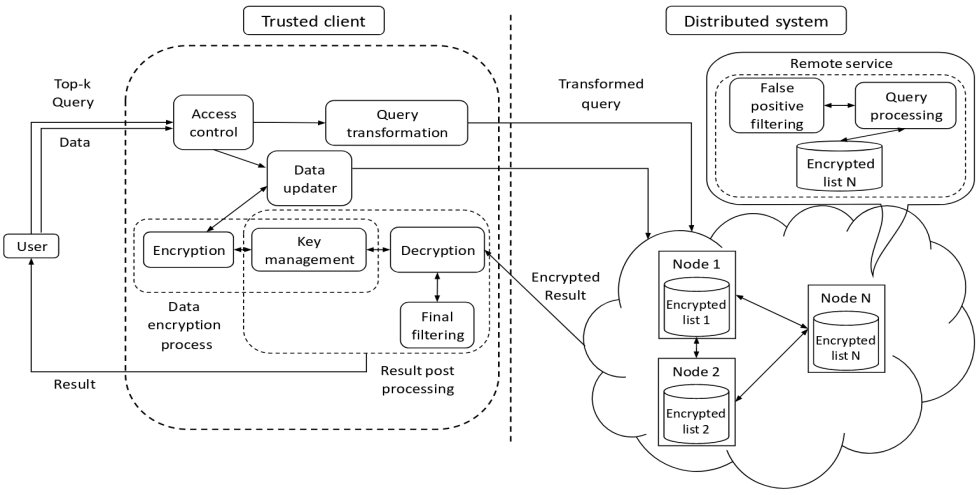

Fig. 1: SD-TOPK architecture

is issued by a user, the trusted client checks the access rights of the user. If the user does not have the required rights to see the query results, then her demand is rejected. Otherwise, the query is transformed to a query that can be executed over the encrypted data.
Note that the trusted client component should be installed in a trusted location, *e.g.*, the machine(s) of the person/organization that outsources the data.

– **Remote service.** It is installed in the nodes of the distributed system, and is responsible for storing the encrypted data, executing the queries provided by the trusted client, and returning the results. This component does not keep any security key, thus cannot decrypt the encrypted data in the distributed system.

In the trusted client of SD-TOPK, we use two types of schemes. *Deterministic* encryption is an encryption scheme that, for two equal inputs, generates the same ciphertexts. With a *probabilistic* encryption, for the same plaintexts different ciphertexts can be generated, but the decryption function returns the same plaintext for them [26]. The details of data encryption in SD-TOPK are described in Section 5.1.

4 TA-Based Algorithms

In this section, we present two basic approaches based on the TA algorithm for top-k query processing over encrypted data in distributed environments: Remote-TA and Block-TA. Our main contribution, i.e. SD-TOPK, will be presented in the next section.

4.1 Data Encryption

To be able to execute TA-based algorithms over encrypted data, the trusted client stores the database in the nodes of the distributed system as follows. It encrypts the pairs $\langle d, s_i(d)\rangle$ of the sorted lists using two encryption schemes: 1) *deterministic* to encrypt data identifier d; 2)*probabilistic* to encrypt local score of the data, *i.e.*, $s_i(d)$. The encrypted pairs are sorted in the same order as their initial order. After encrypting the pairs, the trusted client sends each encrypted sorted list to one node of the distributed system, which is called the *owner* of the list.

4.2 Remote-TA

Given a top-k query containing a number k and a scoring function f, Remote-TA proceeds as follows:

1. The trusted client asks the list owners to return the encrypted pairs (encrypted data id and score) which are in position j of the lists (initially $j = 1$). The list owners return the asked data.
2. The trusted client decrypts the received encrypted scores and calculates a threshold TH by applying the scoring function on the decrypted scores.
3. Let S be the set of encrypted data items returned from the position j of the lists. The trusted client demands the list owners to return the encrypted scores of data items in S. Each list owner does random access in its list to find the encrypted scores of each data item in S, then sends them to the trusted client.
4. Trusted client decrypts each returned data item d and calculates its overall score $ov(d) = f(s_1(d), s_2(d), ..., s_m(d))$. Then, it checks if among the yet received data items there are at least k data items that have an overall score greater than or equal to TH. If this is the case, then it stops the algorithm and returns the k received data items that have the highest overall scores to the user. Otherwise, it increases j by one and restarts from step 1.

4.3 Block-TA

Block-TA is an improvement of the Remote-TA algorithm where the encrypted data items are read block by block. Indeed, in order to minimize the communication cost, in the first step of Block-TA, the trusted client asks blocks of predefined size from the list owners. After decrypting the data items of the block, the trusted client computes the threshold by applying the scoring function on the scores of the last data items in the blocks, and stops if among yet received data items there are at least k data items with overall scores higher than or equal to the threshold. Otherwise it retrieves the next block, and so on.

Example. Consider Table 1 that represents a database composed of three encrypted lists. There are three sorted lists, and each list L_i is stored in a node. The user asks for the top-4 data items in the database with SUM as scoring function.

List 1		List 2		List 3	
encrypted data item	encrypted local score	encrypted data item	encrypted local score	encrypted data item	encrypted local score
$E(d_3)$	$E(30)$	$E(d_3)$	$E(29)$	$E(d_6)$	$E(27)$
$E(d_1)$	$E(27)$	$E(d_6)$	$E(28)$	$E(d_3)$	$E(25)$
$E(d_6)$	$E(26)$	$E(d_2)$	$E(26)$	$E(d_2)$	$E(22)$
$E(d_5)$	$E(24)$	$E(d_1)$	$E(24)$	$E(d_5)$	$E(21)$
$E(d_8)$	$E(20)$	$E(d_7)$	$E(21)$	$E(d_1)$	$E(20)$
$E(d_2)$	$E(15)$	$E(d_4)$	$E(19)$	$E(d_9)$	$E(18)$
$E(d_4)$	$E(14)$	$E(d_5)$	$E(16)$	$E(d_8)$	$E(17)$
$E(d_7)$	$E(12)$	$E(d_9)$	$E(13)$	$E(d_7)$	$E(14)$
$E(d_9)$	$E(11)$	$E(d_8)$	$E(10)$	$E(d_4)$	$E(11)$
...	...	...	...	...	...

Table 1: Example of an encrypted database

Let us run Block-TA on the database of Table 1 by using blocks of size 4. The trusted client asks the list owners to return the 4 first data items in each list. The owner of the list L_1 returns d_3, d_1, d_6 and d_5. It also returns $E(24)$ which is the encrypted score of the last data item in the block. This is used later to calculate the threshold. The owner of L_2 returns d_3, d_6, d_2, d_1 and $E(24)$. The owner of L_3 returns d_6, d_3, d_2, d_5 and $E(21)$. Then, the trusted client decrypts the received data and calculates the threshold $TH = 24 + 24 + 21 = 69$. The client asks for the encrypted score of the returned data items d_1, d_2, d_3, d_5 and d_6. When it gets the asked scores, it decrypts them and calculates the overall score of each data item: $ov(d_1) = 71$, $ov(d_2) = 63$, $ov(d_3) = 84$, $ov(d_5) = 61$, $ov(d_6) = 81$. The client finds that only d_1, d_3 and d_6 have an overall score greater than or equal to the threshold TH, so it asks the next block of four data items in each list. The owner of L_1 returns d_8, d_2, d_4, d_7 and $E(12)$. The owner of L_2 returns d_7, d_4, d_5, d_9 and $E(13)$, and that of L_3 returns d_1, d_9, d_8, d_7 and $E(14)$. The trusted client calculates the threshold $TH = 12 + 13 + 14 = 39$. Then, the client asks each node to return the encrypted score of the data items and calculates their overall score: $ov(d_4) = 44, ov(d_7) = 47, ov(d_8) = 47, ov(d_9) = 42$. The trusted client finds that there are at least 4 data items with overall scores greater than or equal to TH. Thus, it stops communicating with the list owners, and returns to the user the data items d_1, d_2, d_3, d_6 that have the biggest overall score among the data items received from the list owners.

5 SD-TOPK

The TA-based algorithms, presented in the previous section, evaluate correctly the top-k queries over encrypted data. But, as shown by our experiments re-

ported in Section 7, there may be a high number of false positives which are sent from the distributed system nodes to the client, and this renders these algorithms inefficient in practice. In this section, we present the SD-TOPK system, designed for efficient processing top-k queries over encrypted data in distributed systems. As shown by our experiments, SD-TOPK is much more efficient than the TA-based algorithms.

The rest of this section is organized as follows. We first describe SD-TOPK's method for encrypting the data items and storing them in the distributed system. Afterwards, we propose an efficient algorithm for processing top-k queries over the encrypted data. Then, we propose an algorithm for removing the false positives from the results of the top-k query processing algorithm, without decrypting the data. Afterwards, we discuss how we can update the encrypted data in the distributed system. Finally, we propose a technique to enforce the security of data outsourcing in SD-TOPK.

5.1 Data Encryption and Outsourcing

Before outsourcing a database, SD-TOPK creates sorted lists for all important attributes, *i.e.*, those that may be used in the top-k queries. Then, each sorted list is partitioned into buckets. There are several methods for partitioning a sorted list, for example dividing the attribute domain of the list to almost equal intervals or creating buckets with equal sizes. In the current implementation of our system, we use the latter method, *i.e.*, we create buckets with almost the same size where the bucket size is configurable by the system administrator.

Let b_1, b_2, ..., b_t be the created buckets for a sorted list L_j. Each bucket b_i has a lower bound, denoted by $min(b_i)$, and an upper bound, denoted by $max(b_i)$. A data item d is in the bucket b_i, if and only if its local score (attribute value) in the list L_j is between the lower and upper bounds of the bucket, *i.e.*, $min(b_i) \leq s_j(d) < max(b_i)$.

We use two types of encryption schemes (methods) for encrypting the data item ids and the local scores of the sorted lists: *deterministic* and *probabilistic*. The *deterministic* scheme is used to encrypt the ID of the data items. This allows us to have the same encrypted ID for each data item in all sorted lists.

We use the *probabilistic* scheme to encrypt the local scores (attribute values) of data items. By using *probabilistic* encryption, if two data items have the same local scores in a sorted list, their encrypted scores may be different. This protects the scores against the frequency attacks [24] that decrypt order-preserved encrypted lists given auxiliary information about the database, such as the distribution of the plaintext values.

After encrypting the data IDs and local scores of each list L_i, the trusted client puts them in their bucket (chosen based on the local score). Then, the trusted client sends the buckets of each sorted list to one node in the distributed system. The buckets are stored in the nodes according to their lower bound order. However, there is no order for the data items inside each bucket, *i.e.*, *the position of the data items inside each bucket is chosen randomly*. This prevents the nodes to know the order of data items inside the buckets.

5.2 Top-k Query Processing Algorithm

The main idea behind top-k query processing in SD-TOPK is to use the bucket boundaries and a new technique to decide when to stop reading the encrypted data from the lists.

For each top-k query, one of the nodes of the distributed system performs the coordination between the nodes to execute the query. The coordinator may be the node that initially receives the user's query or it can be randomly chosen among the system nodes.

Let us describe our top-k query processing algorithm. Given a top-k query with a number k and a scoring function f that is linear with positive coefficients, *i.e.*, it is in the form of $f = a_1x_1 + a_2x_2 + \ldots + a_mx_m$. SD-TOPK chooses a node as *coordinator*, and then the following steps are performed to answer the query:

1. The coordinator broadcasts the query in parallel to the nodes, and asks each node to return the buckets that contain the k first data items in its list. Each node returns the encrypted identifier of the first k data items, as well as the lower bound of their including buckets.
2. For each returned data item d, the coordinator calculates its *minimum overall score* defined as follows: $ov_{min}(d) = f(v_1(d), v_2(d), ..., v_m(d))$ where $v_i(d)$ is the lower bound of the bucket that contains d in the list L_i. If d has not been returned to the coordinator by the owner of a list L_j then $v_j(d) = 0$.
3. The coordinator sorts the received data items according to their minimum overall score, and chooses the data item d' that has the k^{th} minimum overall score denoted by δ. Then, it uses the minimum overall score of d' to calculate a threshold θ as follows: $\theta = \frac{\delta}{\sum_{i=1}^{m} a_i}$ where $a_1, \ldots, a_m$ are the coefficients in the scoring function.
4. The coordinator broadcasts θ in parallel to the nodes. Each node returns to the coordinator the buckets that have upper bounds greater than or equal to θ.
5. Let Y be the set of all data items that are sent to the coordinator by at least one node. We call Y the set of *candidate items*. The coordinator sends the encrypted id of all data items contained in Y to the nodes, and they return the encrypted score of each data item contained in Y.
6. Finally, the coordinator returns to the trusted client the candidate items and their encrypted local scores.

When the trusted client receives the candidate items, it decrypts them using the secret keys. Then, it calculates for each candidate d its overall score, extracts the k data items that have the highest overall scores, and returns them to the user.

The following theorem shows that the output of the above algorithm contains the encrypted top-k data items.

Theorem 1. *Given a top-k query with a scoring function f that is linear with positive coefficients, then the output of the top-k algorithm of SD-TOPK contains the encrypted top-k results.*

Proof. Let the scoring function be $f = a_1x_1 + a_2x_2 + \ldots + a_mx_m$. Let Y be the output of the algorithm, *i.e.*, the set of candidate items. To prove the theorem, it is sufficient to show that each data item d that has not been sent to the coordinator in the 4th step of the algorithm, has an overall score less than or equal to the overall score of at least k data items in Y. Let θ be the threshold value that is sent to the nodes in the 4th step of the algorithm. For each list L_i, let s_i be the local score of d in the list L_i. The overall score of d is computed as $ov(d) = a_1s_1 + \ldots + a_ms_m$. Since d has not been sent to the coordinator, from the 4th step of the algorithm we know that $s_i < \theta$. Thus, we have $ov(d) < a_1 \times \theta + \ldots + a_m \times \theta = \sum_{i=1}^{m} a_i \times \theta$. From the 3rd step of the algorithm, we know that $\theta = \frac{\delta}{\sum_{i=1}^{m} a_i}$. Thus, we have $ov(d) < \delta$. In other words, the overall score of d is less than the minimum overall score of the data item d' that is the k^{th} data item found in the 3rd step of the algorithm. Therefore, the overall score of d is less than at least k data items found by the top-k algorithm of SD-TOPK, so d cannot be among the top-k results.

5.3 Filtering Algorithm

In the set Y returned by the top-k query processing algorithm, in addition to the top-k results there may be false positives. Below, we propose a filtering algorithm to eliminate most of them in the distributed system nodes, without decrypting the data items.

Given the set of candidate data items Y, the filtering algorithm executed by the coordinator proceeds as follows:

1. Calculate the *minimum overall score* of all candidate data items, sort them according to their minimum overall score, and take the k^{th} *minimum overall score* denoted by δ_2.
2. Calculate the *maximum overall score* of all candidate data items, and eliminate those with *maximum overall score* less than $< \delta_2$. The *maximum overall score* of a data item d is computed as follows: $ov_{max}(d) = f(v_1(d), v_2(d), ..., v_m(d))$ where $v_i(d)$ is the upper bound of the bucket that contains d in the list L_i. If d has not been returned to the coordinator by the node that keeps L_i then $v_i(d)$ is equal to the lower bound of the last bucket received from that node.

The above algorithm eliminates almost all false positives (see the experimental results on filtering rate in Section 7.8), and by doing that it improves significantly the response time of the queries because the eliminated false positives do not need to be communicated to the trusted client and should not be decrypted.

To prove the correctness of the filtering algorithm, we need the following lemmas.

Lemma 1. *Given a top-k query with a scoring function f that is linear with positive coefficients. The minimum overall score of any data item d is less than or equal to its overall score.*

Proof. The minimum overall score of a data item d is calculated by applying the scoring function on the lower bound of the buckets in which d is involved. Let b_i be the bucket that contains d in the list L_i. Let s_i be the local score of d in L_i. Since $d \in b_i$, its local score is higher than or equal to the lower bound of b_i, i.e. $min(b_i) \leq s_i$. Since f is monotonic, we have $f(min(b_1), ..., min(b_m)) \leq f(s_1, ..., s_m)$. Therefore, the minimum overall score of d is less than or equal to its overall score.

Lemma 2. *Given a top-k query with a scoring function f that is linear with positive coefficients, then the maximum overall score of any data item d is greater than or equal to its overall score.*

Proof. The proof can be done in a similar way as lemma 1

The following theorem shows that the filtering algorithm works correctly, i.e., the removed data are only false positives.

Theorem 2. *Given a top-k query with a scoring function f that is linear with positive coefficients, then any data item removed by the filtering algorithm cannot belong to the top-k results.*

Proof. The proof can be done by considering the fact that any removed data item d has a maximum overall score that is lower than the minimum overall score of at least k data items. Thus, according to Lemmas 1 and 2 the overall score of d is less than or equal to that of at least k data items that are not removed. Therefore, we can eliminate d.

5.4 Example

To illustrate SD-TOPK, we use the database shown in Table 4 with a top-4 query and SUM as scoring function. In this example, we suppose that a node n_0 is the coordinator. It sends a messages to all list owners (e.g., n_1, n_2, n_3) and asks for the 4 first data items in each list (since $k = 4$), and the minimum of the buckets in which they are. The node n_1 returns $\langle d_1, 24.6\rangle$ $\langle d_3, 24.6\rangle$ $\langle d_6, 24.6\rangle\langle d_2, 14.8\rangle$. The node n_2 returns $\langle d_6, 25.5\rangle$ $\langle d_3, 25.5\rangle$ $\langle d_2, 25.5\rangle$ $\langle d_1, 18\rangle$ and the node n_3 returns $\langle d_2, 21.9\ \rangle$ $\langle d_3, 21.9\rangle$ $\langle d_6, 21.9\rangle$ $\langle d_5, 17.7\rangle$. The coordinator calculates the minimum overall score of the returned data items by using the minimum of their buckets. It finds that $ov_{min}(d_1) = 24.6 + 18 = 42.6$, $ov_{min}(d_2) = 14.8 + 25.5 + 21.9 = 62.2$, $ov_{min}(d_3) = 24.6 + 25.5 + 21.9 = 72$, $ov_{min}(d_5) = 17.7$ and $ov_{min}(d_6) = 72$. After sorting the minimum overall scores, the coordinator finds that the 4th minimum overall score is 42.6 (that of d_3), so it sets $\delta = 42.6$. Then, it calculates $\theta = \delta/3 = 14.2$. Afterwards, it asks each node to return the encrypted id of the data items that are in buckets b_i such that $max(b_i) \geq \theta$. The data items received from list owners are called candidate data items. The coordinator calculates for the candidate items their minimum overall score: $ov_{min}(d_1) = 60.3$, $ov_{min}(d_2) = 62.2$, $ov_{min}(d_3) = 72$, $ov_{min}(d_4) = 38.7$, $ov_{min}(d_5) = 41.5$, $ov_{min}(d_6) = 72$,

List 1			List 2			List 3		
bucket ID	encrypted data item	encrypted local score	bucket ID	encrypted data item	encrypted local score	bucket ID	encrypted data item	encrypted local score
B_{11}	$E(d_1)$	$E(27)$	B_{21}	$E(d_6)$	$E(28)$	B_{31}	$E(d_2)$	$E(22)$
B_{11}	$E(d_3)$	$E(30)$	B_{21}	$E(d_3)$	$E(29)$	B_{31}	$E(d_3)$	$E(25)$
B_{11}	$E(d_6)$	$E(26)$	B_{21}	$E(d_2)$	$E(26)$	B_{31}	$E(d_6)$	$E(27)$
B_{12}	$E(d_2)$	$E(15)$	B_{22}	$E(d_1)$	$E(24)$	B_{32}	$E(d_5)$	$E(21)$
B_{12}	$E(d_8)$	$E(20)$	B_{22}	$E(d_7)$	$E(21)$	B_{32}	$E(d_1)$	$E(20)$
B_{12}	$E(d_5)$	$E(24)$	B_{22}	$E(d_4)$	$E(19)$	B_{32}	$E(d_9)$	$E(18)$
B_{13}	$E(d_4)$	$E(14)$	B_{23}	$E(d_5)$	$E(16)$	B_{33}	$E(d_8)$	$E(17)$
B_{13}	$E(d_9)$	$E(11)$	B_{23}	$E(d_9)$	$E(13)$	B_{33}	$E(d_7)$	$E(14)$
B_{13}	$E(d_7)$	$E(12)$	B_{23}	$E(d_8)$	$E(10)$	B_{33}	$E(d_4)$	$E(11)$
...	...	...	...	...	...	...	...	...

Table 2: Encrypted database, with 3 data items in each bucket. The encrypted scores inside buckets are not sorted. The boundaries (minimum and maximum) of buckets are shown below

List 1			List 2			List 3		
bucket ID	min	max	bucket ID	min	max	bucket ID	min	max
B_{11}	24.6	32	B_{21}	25.5	31	B_{31}	21.9	28
B_{12}	14.8	24.1	B_{22}	18	24.1	B_{32}	17.7	21.5
B_{13}	10.7	14.2	B_{23}	9	16.5	B_{33}	10	17.3

Table 3: Bucket boundaries

Table 4: Example of an encrypted database, and the information about its buckets.

$ov_{min}(d_7) = 38.7$, $ov_{min}(d_8) = 33.8$ and $ov_{min}(d_9) = 37.4$. It also calculates $\delta_2 = 60.3$, that is the kth minimum overall score among candidate data items (defined in the filtering algorithm). Then, it calculates the maximum overall scores of the candidate data items: $ov_{max}(d_1) = 77.6$, $ov_{max}(d_2) = 83.1$, $ov_{max}(d_3) = 91$, $ov_{max}(d_4) = 55.6$, $ov_{max}(d_5) = 62.1$, $ov_{max}(d_6) = 91$, $ov_{max}(d_7) = 55.6$, $ov_{max}(d_8) = 57.9$ and $ov_{max}(d_9) = 52.2$. According to the filtering algorithm, the coordinator eliminates the data items that have a maximum overall score less than 60.3. Then, it remains five data items in the set of candidate items $Y = \{d_1, d_2, d_3, d_5, d_6\}$. The coordinator asks the list owners to return all encrypted scores of the candidate items and sends them to the trusted client. When the client receives the data items, it decrypts them and calculates their real overall score: $ov(d_1) = 71, ov(d_2) = 63, ov(d_3) = 84, ov(d_5) = 61, ov(d_6) = 81$. Finally, the trusted client finds that the top-4 data items are d_1, d_2, d_3 and d_6. It returns them to the user who had issued the query.

5.5 Update Management

In our system, updating a data item d in the nodes is done by deleting the old encrypted scores (attribute values) of d and then inserting its new scores.

To delete the old encrypted scores of d from the outsourced database, the trusted client encrypts the ID of d using the key that has been used for encrypting the data IDs, and then asks the nodes of the distributed system to find the encrypted ID in the buckets of their lists and then remove the pairs encrypted score and encrypted ID of d from the lists.

Inserting the new scores of d is done as follows. The trusted client uses the metadata of the buckets (*i.e.*, the lower and upper bounds), and for each list L_i, it calculates the bucket of the list to which the data score s_i should be stored. Let b_i be the corresponding bucket of s_i. The trusted client encrypts the ID and scores of d by using the encryption schemes that are used for encrypting the ID and scores. Then, it asks the nodes to put the encrypted ID and encrypted scores of d in the corresponding buckets.

5.6 Obfuscating Bucket Boundaries

A drawback of the basic version of SD-TOPK, presented until now, is that the limits (i.e., lower and upper bounds) of the buckets are disclosed to the nodes of the distributed system. To strengthen the security of our system, we change the bucket limits as follows. We choose two random numbers a and c. These numbers must be kept secret in the trusted client. Before encrypting the database, the lower and upper bounds of each bucket b_i are obfuscated (modified) as follows:

$$min(b_i) := min(b_i) \times a + c \tag{1}$$

$$max(b_i) := max(b_i) \times a + c \tag{2}$$

Thus, the trusted client multiplies the lower (upper) bounds by the secret number a, and then adds the secret number c to the result. These obfuscated

bucket limits are sent to the nodes of distributed system together with the encrypted IDs and scores. By the above strategy, we can hide the limits of the buckets from the nodes.

The following theorem proves that SD-TOPK works correctly if it uses the obfuscated lower bounds.

Theorem 3. *Assume a top-k query with a scoring function f that is linear with positive coefficients. If we change the lower bound of the buckets by using Equation 1, then the output of SD-TOPK will involve the top-k results.*

Proof. Let the scoring function be $f = a_1x_1 + a_2x_2 + \ldots + a_mx_m$. Let Y be the output of SD-TOPK algorithm, *i.e.*, the set of candidate items. We show that each data item d that has not been sent to the coordinator by the list owners, has an overall score that is less than or equal to the overall score of at least k data items involved in Y. Let b_i be the bucket that contains the data d in the list L_i, and thus $max(b_i) * a + c$ is the new (modified) upper bound of b_i. From the 4^{th} step of Sd-TOPK, we know that in each list L_i we have $max(b_i) * a + c < \theta$. Thus, we have $a_1 \times (max(b_1) * a + c) + \ldots + a_m \times (max(b_m) * a + c) < \sum_{i=1}^{m} a_i \times \theta$. We know that $\theta = \frac{\delta}{\sum_{i=1}^{m} a_i}$. Thus, we have the following equation:

$$a_1 \times (max(b_1) * a + c) + \ldots + a_m \times (max(b_m) * a + c) < \delta \tag{3}$$

Now, let d' be the data item that has the k^{th} minimum overall score in the 3rd step of SD-TOPK. In each list L_i, let b'_i be the bucket that contains d' in L_i, and thus $min(b'_i) * a + c$ is the new (modified) lower bound of b'_i. From the 3rd step of the algorithm, we know that the minimum overall score of d' (computed by using the obfuscated buckets) is equal to δ. Thus, we have $a_1 \times (min(b'_1) * a + c) + \ldots + a_m \times (min(b'_m) * a + c) = \delta$. Thus, we have the following equation:

$$a_1 \times (min(b'_1) * a + c) + \ldots + a_m \times (min(b'_m) * a + c) = \delta \tag{4}$$

By comparing Equations 3 and 4, we have: $a_1 \times (min(b'_1) * a + c) + \ldots + a_m \times (min(b'_m) * a + c) > a_1 \times (max(b_1) * a + c) + \ldots + a_m \times (max(b_m) * a + c)$. Since the numbers a and c are positive, we can write: $a_1 \times (min(b'_1) + \ldots + a_m \times min(b'_m) > a_1 \times max(b_1) + \ldots + a_m \times max(b_m)$. This means that the minimum overall score of the data item d' is higher than the maximum overall score of d. In other words, the data item d could not be among the top-k results.

The following theorem shows that the filtering algorithm works correctly, if it uses the obfuscated bucket limits.

Theorem 4. *Assume a top-k query with a scoring function f that is linear with positive coefficients. If we change the lower bound of the buckets by using Equation 1, then the filtering algorithm does not remove any top-k result.*

Proof. Let Y be the output of SD-TOPK algorithm, and d' be the data item that has the k^{th} minimum overall score among the data items in the 3rd step of SD-TOPK. In each list L_i, let b'_i and s'_i be the bucket and local score of d'_i in the list.

We do the proof by contradiction. We assume a top-k data item d has been removed by the filtering algorithm, and show that this assumption yields to a contradiction. Let b_i and s_i be the bucket and local score of d_i in the list L_i. Since, d has been removed from the list, its maximum overall score using the modified limits is lower than or equal to minimum overall score of d'. Thus, we have: $a_1 \times (max(b_1) \times a + c), ..., a_m \times (max(b_m) \times a + c) \leq a_1 \times (min(b'_1) \times a + c), ..., a_m \times (min(b'_1) \times a + c)$. Since the parameters a and c are positive, we have: $a_1 \times max(b_1), ..., a_m \times max(b_m) \leq a_1 \times min(b'_1), ..., a_m \times min(b'_1)$. This means that the maximum overall score of the data item d is lower than the minimum overall score of d'. Thus, d cannot be a top-k result.

6 Security Analysis and Improvement

In this section, we analyze the different types of information that can be leaked to the adversary (the nodes of the distributed system), and for each type of leakage, we propose some techniques to reduce the risk of disclosing sensitive data.

6.1 Partial Order Leakage

In SD-TOPK, we use the bucketization technique for managing the data in the distributed system. Inside the buckets, no information is leaked because the data items are not ordered and the local scores are encrypted using a probabilistic scheme. But a partial order is leaked about the data items that are in different buckets (since the buckets are ordered).

Even a partial order leakage may help the adversary to obtain rough information about the sensitive data of individuals if he has some background information about the data. For example, if the adversary A knows that the age of a target person u is very high, then A may find the bucket containing u in the list corresponding to age (*i.e.*, the first bucket of the list). Then, by guessing the ID of u in the bucket (e.g., if the size of the bucket is too small), A may find the bucket of u in the salary's list, and then estimate her salary with some confidence probability. *We show that this probability (i.e., the risk of privacy violation) is very low, when the size of buckets is not small.*

Let u be an individual (data item) in the database, and assume that the adversary A knows the value of u in some attribute a. We want to compute the confidence probability that A finds the bucket containing u's value in a sensitive attribute s. Let us denote this confidence probability by $P(b_{s,u}|a)$. We assume that if A finds the bucket of u in the list representing s, then he can make a good estimation of u's value, *e.g.*, using some background knowledge about the values of attribute s.

To find the bucket of u in the sensitive attribute s, the adversary A needs to perform the following steps: 1) guessing the lists that represent a and s; 2) finding the bucket of u in the list representing a; 3) guessing the ID of u in the found bucket; 4) searching u's ID in the list representing s, and finding its bucket.

Let $P(L_1 = a \wedge L_2 = s)$ be the probability that A guesses correctly the lists representing the attributes a and s. Let m be the number of lists in the database. In our system, the metadata of sorted lists (*e.g.*, their identification) is encrypted, and they have the same size and format. Thus, the probability of finding the correct list of an attribute is $\frac{1}{m}$. Therefore, the probability of correctly guessing the lists representing both attributes a and s is:

$$P(L_1 = a \wedge L_2 = s) = \frac{1}{m \times (m-1)} \tag{5}$$

If the adversary A finds correctly the list representing a, then we assume that A is able to find the bucket containing u by using the background knowledge about the value of u in a (and some statistical information). After finding the bucket, say b, the adversary needs to guess the ID of u. Let $size(b)$ be the number of encrypted values in the bucket b. Then, the probability of finding u' ID in the bucket b, denoted as $P(ID = u)$, is:

$$P(ID = u) = \frac{1}{|size(b)|} \tag{6}$$

If A guesses correctly the ID of u in the bucket b, then he can find the bucket containing the ID in the list representing the attribute s (if he guesses correctly the list of s), and then he can roughly estimate the u's value in s.

The following theorem provides a formula that calculates the probability that an adversary finds the bucket of an individual u in the sensitive attribute s by knowing the value of u in an attribute a.

Theorem 5. *Let s be a sensitive attribute, $size(b)$ be the size of the buckets, and m be the number of sorted lists (attributes). Let $P(b_{s,u}|a)$ be the probability that the adversary detects correctly the bucket of an individual u in the sensitive attribute s by knowing the value of u in an attribute a. Then, $P(b_{s,u}|a)$ is:*

$$P(b_{s,u}|a) \leq \frac{1}{size(b) \times m \times (m-1)} \tag{7}$$

Proof. The proof can be done using Equations 5 and 6.

The above theorem shows that when the size of the buckets is not small, the probability of privacy violation is negligible.

When the bucket size is one (which is equivalent of preserving the total order), the risk of privacy violation is the highest. We advise at least the size of 10 for the buckets, in order to make the privacy violation risk very low. For example, if the number of sorted lists (*i.e.*, attributes) is $m = 5$, with the bucket size of 10, the privacy violation risk $P(b_{s,u}|a)$ is less than 0.005.

However, note that choosing very big buckets increases the response time of query processing (see Section 7.5). Therefore, the size of the buckets should be taken based on the user privacy requirements (*e.g.*, the maximum acceptable probability of privacy violation) and performance (*e.g.*, response time).

6.2 Obfuscating the Number of Asked Results

In the basic version of our approach, the information about the number of asked results, *i.e.*, k, is disclosed to the cloud provider. We can obfuscate this information as follows. The trusted client generates a random integer s between 0 and a predefined (small) value, and adds it to k. Then, it sends $k' = k + s$ to the cloud as the number of required results. After receiving the encrypted results from the cloud, the trusted client filters the result set and sends only k results to the user.

6.3 Obfuscating the Database Size

Another information which is leaked is the database size (*i.e.*, the number of tuples). This leakage can be avoided by adding dummy tuples to the database before sending it to the cloud. But, we have to be careful not to add dummy tuples which could be returned to the user as a result of top-k queries. For this, we can proceed as follows. Let n' be the number of dummy data items that we want to add to the lists. In each list L_i, let s_i be the last local score in the list. We generate n' random data IDs. Then, for each list L_i, we generate n' random scores smaller than s_i, and assign them randomly to the n' data IDs. Afterwards, we add the generated data IDs and their local scores to the sorted lists. Since the local scores of the dummy data items are smaller than any real data item, they have no chance to be returned as a result of top-k queries.

7 Performance Evaluation

In this section, we first describe the experimental setup, and then present the performance evaluation results.

7.1 Setup

We implemented SD-TOPK and performed experiments on real and synthetic datasets. As in some previous work on privacy (*e.g.*, [21]), we use the Gowalla database, which is a location-based social networking dataset collected from users locations. The database contains 6 million tuples where each tuple represents user number, time, user geographic position, etc. In our experiments, we are interested in the attribute time, which is the second value in each tuple. As in [21], we decomposed this attribute into 6 attributes (year, month, day, hour, minute, second), and then created a database with the values of those attributes.

In addition to the real dataset, we have also generated random datasets using uniform and Gaussian distributions.

We compared SD-TOPK with the two TA-Based algorithms: *Remote-TA* and *Block-TA*.

In the experiments, the number of nodes is equal to the number of lists, *i.e.,* each node stores one of the lists. The coordinator of SD-TOPK is one of the nodes of the system (randomly chosen).

We study the effect of several parameters: 1) n: the number of data items in the database; 2) m: the number of lists; 3) k: the number of required top items; 4) $bsize$: the number of data items in the buckets (or blocks) in SD-TOPK and Block-TA. The default value for n is 2M items. Unless otherwise specified, m is 5, k is 50, and $bsize$ is 10. The default database is the synthetic uniform database, and the latency of the messages is around 50 ms.

To evaluate the performance of SD-TOPK, we measured the following metrics:

- **Response time:** includes top-k query processing time, communication time, filtering time, and the result post-processing time (*e.g.*, decryption).
- **Filtering rate:** the number of false positives eliminated by the filtering algorithm in the distributed system.
- **Communication cost:** we measure two metrics: 1) the number of messages communicated between the nodes to answer a top-k query; 2) the total number of bytes communicated to answer a top-k query.

7.2 Effect of Database Size

In this section, we compare the response time of SD-TOPK, Remote-TA and Block-TA, while varying the number of data items, *i.e.*, n.

Figure 2 shows how response time evolves, with increasing n, while the other parameters are set as default values described in Section 7.1. Note that the results are shown in logarithmic scale. The response time of all approaches increases with increasing the database size. SD-TOPK is the best; its response time is at least two orders of magnitude better than the other algorithms. This high difference between SD-TOPK and TA-based algorithms is mainly due to the high number of encrypted data items that should be decrypted by TA-based algorithms in the trusted client, and also the messages needed for communicating them. Block-TA performs better than Remote-TA, because of reading the lists in blocks, thus it needs less number of messages.

7.3 Effect of the Number of Lists

Figure 3 shows the response time of SD-TOPK and TA-based algorithms when varying m (i.e., the number of attributes in the scoring function), and the other parameters set as default values. We observe that the response time of SD-TOPK increases slightly comparing to Remote-TA and Block-TA when the number of

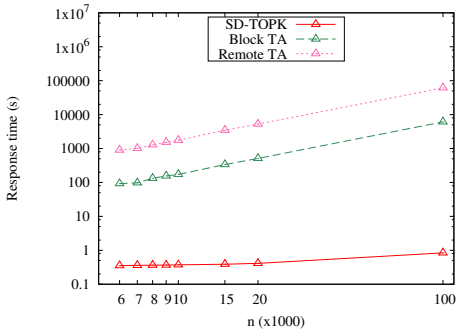

Fig. 2: Response time vs. number of database tuples

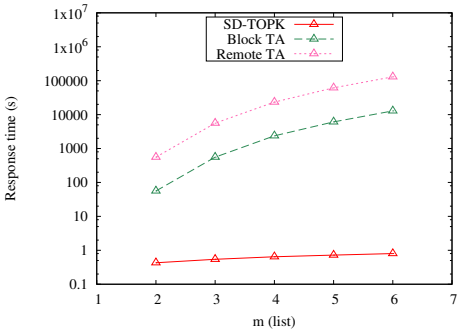

Fig. 3: Response time vs. number of lists

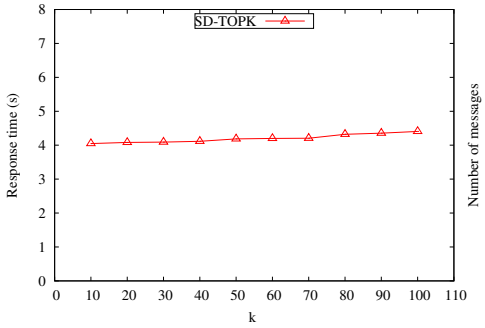

Fig. 4: Response time vs. k

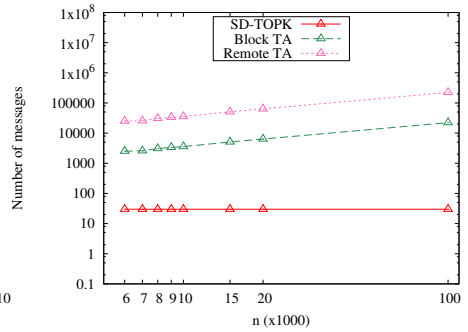

Fig. 5: Number of communicated messages vs. number of database tuples

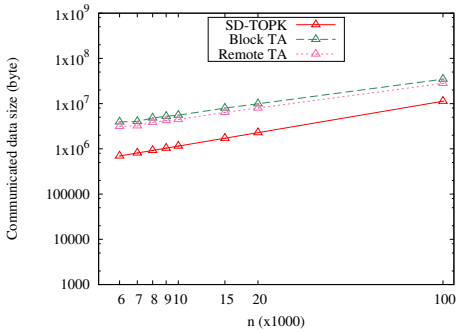

Fig. 6: Size of communicated data (in bytes) vs. number of database tuples

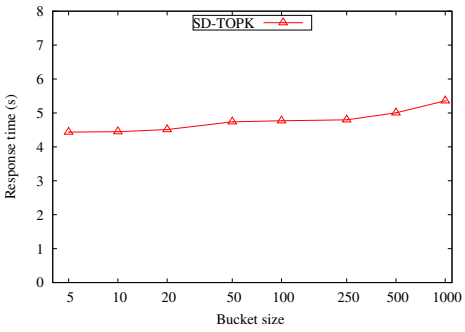

Fig. 7: Response time vs. bucket size

lists increases. The reason is that when we increase the number of lists, more data (sent by the nodes) should be processed by the coordinator for finding the candidate items.

7.4 Effect of k

Figure 4 shows the response times of SD-TOPK with increasing k, and the other parameters set as default values. We observe that with increasing k the response time increases slightly. The reason is that when k increases, SD-TOPK needs to get more data items from the list owner nodes in each step. In addition, increasing k augments the number of data items that the trusted client needs to decrypt (because at least k data items are decrypted by the trusted client).

7.5 Effect of Bucket Size

Figure 7 reports the response time of SD-TOPK when varying the size of buckets, and the other parameters set as default values. We observe that the response time increases slightly when the bucket size increases. The reason is that increasing the bucket size increases the number of data items to be considered in the different steps of SD-TOP algorithm. It also increases the number of false positives to be removed by the filtering algorithm.

7.6 Communication Cost

We measure the communication cost of SD-TOPK, Remote-TA and Block-TA in terms of the total number of messages exchanged between the different nodes of the distributed system and the size of the exchanged data.

Figure 5 shows the number of communicated messages while increasing the number of tuples and fixing the other parameters to the default values. We observe that SD-TOPK needs to exchange a small number of messages comparing to the others approaches. The reason is that SD-TOPK runs in only some rounds of communication, and does not depend on the database size. But for the TA-based algorithms, the number of messages depends on the position where they stop in the lists, and that position depends on the database size.

Figure 6 illustrates the size of the communicated data in bytes, while increasing the number of tuples in the database and setting the other parameters to the default values. We note that the size of the communicated data increases with the database size. The amount of data transferred by SD-TOPK is less than that of Remote-TA and Block-TA. The reason is that SD-TOPK uses the obfuscated bucket boundaries to check the top-k data items and these boundaries have a size less than the encrypted scores used by other algorithms.

7.7 Performance over Different Datasets

We study the effect of the datasets on the performance of SD-TOPK, Remote-TA and Block-TA using different datasets: synthetic datasets with uniform and

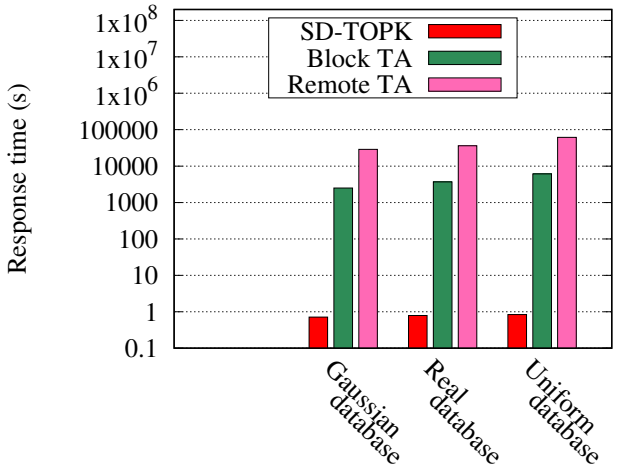

Fig. 8: Response time using different databases

Gaussian distributions, and real dataset (Gowalla). Figure 8 shows the response time of the approaches over different datasets, while other parameters are set as default values. We see that the performance of all approaches over the Gaussian database is better than real and uniform databases. The reason is that with the latter databases, the algorithms need to go deeper into the lists to be sure that they have found the top-k results.

7.8 Filtering Rate

We study the efficiency of the filtering algorithm of SD-TOPK by using different datasets. The results are shown in Table 5. The results show that the filtering algorithm is very efficient over all the tested datasets. However, there is a little difference in the filtering rates because of the local score distributions. For example, in the Gaussian distribution, the local scores of many data items are very close to each other, thus the filtering rate decreases in this dataset.

	Uniform dataset	Real dataset	Gaussian dataset
filtering Rate	100%	99.995%	99.991%

Table 5: False positive elimination by the filtering algorithm of SD-TOPK over different databases

8 Related Work

Efficient processing of top-k queries is important for many applications such as information retrieval [32], sensor networks [38], data stream management systems [28, 35], crowdsourcing [7], string matching [34], spatial data analysis [29], temporal databases [25], graph databases [16, 19], uncertain data ranking [11, 30], etc.

One of the most efficient algorithms for top-k query processing is the TA algorithm [13], which models the general problem of top-k using lists of data items sorted by their local scores and proposes a simple and efficient algorithm. Several TA-based algorithms have been proposed for processing top-k queries in different environments, e.g. [3, 1, 8]. However, all these algorithms assume that the data scores are available as plaintext, and not encrypted.

In the literature, there has been some research work to process keyword queries over encrypted data , *e.g.*, [5, 31]. For example [5] and [31] propose matching techniques to search words in encrypted documents. However, the proposed techniques cannot be used to answer top-k queries. There have been also some solutions proposed for secure kNN similarity search, *e.g.*, [12, 6, 10, 22, 36]. The problem is to find k points in the search space that are the nearest to a given point. This problem should not be confused with the top-k problem in which the given scoring function plays an important role, such that on the same database and with the same k, if the user changes the scoring function, then the output may change. Thus, the proposed solutions proposed for kNN cannot deal with the top-k problem.

The bucketization technique (*i.e.*, creating buckets) has been used in the literature for answering range queries over encrypted data, *e.g.*, [18, 17, 27]. For example, in [18], Hore et al. use this technique, and propose optimal solutions for distributing the encrypted data in the buckets in order to guarantee a good performance for range queries. However, the techniques for range queries no longer apply to top-k queries, because in a top-k query, we should find the k tuples that are most relevant to the query using a given scoring function. In a range query, the objective is simpler, i.e. only to find the tuples whose attribute values are in a given range.

In [20], Kim et al. propose an approach for preserving the privacy of data access patterns during top-k query processing. In [33], Vaidya et. al. propose a privacy preserving method for top-k selection from the data shared by individuals in a distributed system. Their objective is to avoid disclosing the data of each node to other nodes. But, their assumption about the nodes is different from ours, because they can trust the node that stores the data (this is why the data are not crypted), but in our system we trust no node of the distributed system.

When we think about top-k query processing on encrypted data, the first idea that comes to mind is the utilization of a fully homomorphic encryption cryptosystem, e.g. [15], which allows one to do arithmetic operations over encrypted data. Using this type of encryption would allow to compute the overall score of data items over encrypted data. However, fully homomorphic encryption

is still impractical for query processing over large databases, particularly due to the prohibitive computational time [9, 2].

CryptDB [26] is a system designed for processing SQL like queries over encrypted data. It is capable to execute several types of queries, *e.g.*, exact-match, join and range queries. However, top-k queries are not supported by CryptDB.

The Three Phase Uniform Threshold (TPUT) [4] is an efficient algorithm to answer top-k queries in distributed systems. Like our SD-TOPK algorithm, it is done in few round-trips between the nodes of the distributed system. However, TPUT can be used only with the queries in which the scoring function is SUM, whereas our algorithm can be used for a large range of scoring functions. In addition, our algorithm finds top-k results over encrypted data, while TPUT can be used only over plaintext data.

Meng et al [37] propose a solution for processing top-k queries over encrypted data in clouds. They assume the existence of two non-colluding nodes, one of which can decrypt the data (using the decryption key) and execute a TA-based algorithm. Our assumptions about the nodes of the distributed system are different, as we do not trust any node.

9 Conclusion

In this paper, we proposed SD-TOPK, an efficient system to encrypt and outsource user data in a distributed system. SD-TOPK is able to evaluate top-k queries over encrypted data, without decrypting them in the nodes of the system. We evaluated the performance of our solution over synthetic and real databases. The results show excellent response time and communication cost for SD-TOPK. They show that the response time of SD-TOPK can be several order of magnitude better than that of the TA-based algorithms. This is mainly due to its optimized top-k query processing and filtering algorithms. The results also show a significant gain in communication cost of SD-TOPK compared to the other algorithms. They also show the efficiency of the filtering algorithm that eliminates almost all false positives in the distributed system.

References

1. Akbarinia, R., Pacitti, E., Valduriez, P.: Best position algorithms for efficient top-k query processing. Inf. Syst. **36**(6), 973–989 (2011)
2. Barhamgi, M., Bandara, A.K., Yu, Y., Belhajjame, K., Nuseibeh, B.: Protecting privacy in the cloud: Current practices, future directions. IEEE Computer **49**(2), 68–72 (2016). https://doi.org/10.1109/MC.2016.59, https://doi.org/10.1109/MC.2016.59
3. Bast, H., Majumdar, D., Schenkel, R., Theobald, M., Weikum, G.: Io-top-k: Index-access optimized top-k query processing. In: Proc. of International Conference on Very Large Databases (VLDB). pp. 475–486 (2006)
4. Cao, P., Wang, Z.: Efficient top-k query calculation in distributed networks. In: Proc. of ACM PODC. pp. 206–215 (2004)

5. Chang, Y., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In: ACNS. pp. 442–455 (2005)
6. Choi, S., Ghinita, G., Lim, H., Bertino, E.: Secure knn query processing in untrusted cloud environments. IEEE TKDE **26**(11), 2818–2831 (2014)
7. Ciceri, E., Fraternali, P., Martinenghi, D., Tagliasacchi, M.: Crowdsourcing for top-k query processing over uncertain data. IEEE TKDE **28**(1), 41–53 (2016)
8. Das, G., Gunopulos, D., Koudas, N., Tsirogiannis, D.: Answering top-k queries using views. In: Proc. of International Conference on Very Large Databases (VLDB). pp. 451–462 (2006)
9. Demertzis, I., Papadopoulos, S., Papapetrou, O., Deligiannakis, A., Garofalakis, M.N.: Practical private range search revisited. In: Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016. pp. 185–198 (2016)
10. Ding, X., Liu, P., Jin, H.: Privacy-preserving multi-keyword top-k similarity search over encrypted data. IEEE TDSC (99), 1–14 (2017)
11. Dylla, M., Miliaraki, I., Theobald, M.: Top-k query processing in probabilistic databases with non-materialized views. In: Proc. of IEEE International Conference on Data Engineering (ICDE). pp. 122–133 (2013)
12. Elmehdwi, Y., Samanthula, B.K., Jiang, W.: Secure k-nearest neighbor query over encrypted data in outsourced environments. In: Proc. of IEEE ICDE. pp. 664–675 (2014)
13. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. In: Proc. of ACM PODS (2001)
14. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci. **66**(4), 614–656 (2003)
15. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: ACM STOC. pp. 169–178 (2009)
16. Gupta, M., Gao, J., Yan, X., Cam, H., Han, J.: Top-k interesting subgraph discovery in information networks. In: IEEE ICDE. pp. 820–831 (2014)
17. Hore, B., Mehrotra, S., Canim, M., Kantarcioglu, M.: Secure multidimensional range queries over outsourced data. J. VLDB **21**(3), 333–358 (2012)
18. Hore, B., Mehrotra, S., Tsudik, G.: A privacy-preserving index for range queries. In: Proc. of International Conference on Very Large Databases (VLDB). pp. 720–731 (2004)
19. Khemmarat, S., Gao, L.: Fast top-k path-based relevance query on massive graphs. In: Proc. of IEEE International Conference on Data Engineering (ICDE). pp. 316–327 (2014)
20. Kim, H., Kim, H., Chang, J.: A privacy-preserving top-k query processing algorithm in the cloud computing. In: Economics of Grids, Clouds, Systems, and Services (GECON). pp. 277–292 (2016)
21. Li, R., Liu, A.X., Wang, A.L., Bruhadeshwar, B.: Fast range query processing with strong privacy protection for cloud computing. PVLDB **7**(14), 1953–1964 (2014)
22. Liao, X., Li, J.: Privacy-preserving and secure top-k query in two-tier wireless sensor network. In: Global Communications Conference (GLOBECOM). pp. 335–341 (2012)
23. Mahboubi, S., Akbarinia, R., Valduriez, P.: Privacy-preserving top-k query processing in distributed systems. In: 24th International Conference on Parallel and Distributed Computing (Euro-Par). pp. 281–292 (2018)
24. Naveed, M., Kamara, S., Wright, C.V.: Inference attacks on property-preserving encrypted databases. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. pp. 644–655. ACM (2015)

25. Pilourdault, J., Leroy, V., Amer-Yahia, S.: Distributed evaluation of top-k temporal joins. In: ACM SIGMOD. pp. 1027–1039 (2016)
26. Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: Cryptdb: processing queries on an encrypted database. Commun. ACM **55**(9), 103–111 (2012)
27. Sahin, C., Allard, T., Akbarinia, R., Abbadi, A.E., Pacitti, E.: A differentially private index for range query processing in clouds. In: ICDE Conf. (2018)
28. Shen, Z., Cheema, M.A., Lin, X., Zhang, W., Wang, H.: Efficiently monitoring top-k pairs over sliding windows. In: Proc. of IEEE International Conference on Data Engineering (ICDE). pp. 798–809 (2012)
29. Shi, J., Wu, D., Mamoulis, N.: Top-k relevant semantic place retrieval on spatial RDF data. In: ACM SIGMOD. pp. 1977–1990 (2016)
30. Soliman, M.A., Ilyas, I.F., Chang, K.C.: Top-k query processing in uncertain databases. In: Proc. of IEEE International Conference on Data Engineering (ICDE). pp. 896–905 (2007)
31. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: IEEE S&P. pp. 44–55 (2000)
32. U, L.H., Mamoulis, N., Berberich, K., Bedathur, S.J.: Durable top-k search in document archives. In: Proc. of ACM International Conference on Management of Data (SIGMOD). pp. 555–566 (2010)
33. Vaidya, J., Clifton, C.: Privacy-preserving top-k queries. In: Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on. pp. 545–546. IEEE (2005)
34. Wang, J., Li, G., Deng, D., Zhang, Y., Feng, J.: Two birds with one stone: An efficient hierarchical framework for top-k and threshold-based string similarity search. In: Proc. of IEEE International Conference on Data Engineering (ICDE). pp. 519–530 (2015)
35. Wang, X., Zhang, Y., Zhang, W., Lin, X., Huang, Z.: SKYPE: top-k spatial-keyword publish/subscribe over sliding window. PVLDB **9**(7), 588–599 (2016)
36. Wong, W.K., Cheung, D.W., Kao, B., Mamoulis, N.: Secure knn computation on encrypted databases. In: ACM SIGMOD. pp. 139–152 (2009)
37. Xianrui Meng, H.Z., Kollios, G.: Top-k query processing on encrypted databases with strong security guarantees. In: ICDE Conf. (2018)
38. Yang, H., Chung, C., Kim, M.: An efficient top-k query processing framework in mobile sensor networks. Data Knowl. Eng. **102**, 78–95 (2016)