Privacy-Preserving Top-k Query Processing in Distributed Systems

Sakina Mahboubi, Reza Akbarinia, Patrick Valduriez

Privacy-Preserving Top-k Query Processing in Distributed Systems

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

INRIA & LIRMM, University of Montpellier, France,

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, that 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.

1 Introduction

We consider a distributed system where users can outsource their sensitive data and issue top-k queries. A top-k query is an important kind of query that allows the user to get the k data items that are most relevant to the query. 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 critical for many 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 [6] 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.

In this paper, 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 this paper is organized as follows. Section 2 gives the problem definition. Section 3 describes SD-TOPK system. Section 4 presents the performance evaluation results. Section 5 discusses related work, and Section 6 concludes.

2 Problem Definition

In this section, we 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*) [6]. 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.

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 $LFPC$). Formally, a function f is LFPC if $f = a_1x_1 + a_2x_2 + \ldots + a_mx_m$ where each coefficient $a_i \geq 0$ for $1 \leq i \leq m$. Many functions such as SUM, COUNT, AVG and MAX are in the class of LFPC functions.

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 preserving processing solutions [10].

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

In this section, we present our system, called SD-TOPK, that encrypts and outsources the user data in a distributed system, and is capable to efficiently evaluate top-k queries over the distributed encrypted data.

The rest of this section is organized as follows. We first describe the architecture of our outsourcing system. Then, we present our method for encrypting the data items and storing them in the distributed system. Afterwards, we propose our algorithm for processing top-k queries over the encrypted data.

3.1 System Architecture

The architecture of our outsourcing system has two main components:

- **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 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.

3.2 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*. With the *deterministic* scheme, for two equal inputs, the same ciphertexts (encrypted values) are generated. We use this scheme 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.

The *probabilistic* scheme is used to encrypt the local scores (attribute values) of data items. With the *probabilistic* encryption, for the same plaintexts different ciphertexts are generated, but the decryption function returns the same plaintext for them. Thus, for example if two data items have the same local scores in a sorted list, their encrypted scores may be different. The probabilistic encryption is the strongest type of encryption.

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 place of the data items inside each bucket is chosen randomly*. This prevents the nodes to know the order of data items inside the buckets.

3.3 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. We call this node as *coordinator*. The coordinator may be the node that initially receives the user's query or 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. Therefor, 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. □

In the set Y returned by the above 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, 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 4), 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 strengthen the security of our system, we obfuscate the bucket boundaries as follows. We choose two random numbers a and c. These numbers are kept secret in the trusted client. Before sending the encrypted database to the nodes of the distributed system, the trusted client multiplies the lower (and upper) bounds of buckets by a secret number a, and then adds the secret number c to the result. These obfuscated bucket boundaries are sent to the nodes together with the encrypted IDs and scores.

4 Performance Evaluation

In this section, we first describe the experimental setup, and then report the results of our experiments.

4.1 Setup

We implemented SD-TOPK and performed experiments on real and synthetic datasets. As in some previous work on privacy (*e.g.*, [10]), 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 [10], 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 two algorithms based on the TA algorithm [6]: *Remote-TA* and *Block-TA*. In Remote-TA, the trusted client retrieves the encrypted data from the sorted lists of the distributed system one by one using sorted access, and for each retrieved data d, it retrieves the encrypted local scores of d from the other lists, decrypts the read local scores, computes the TA threshold, and checks if it can stop or not (as in TA). Block-TA is like Remote-TA, except that the encrypted data items are read block by block. For the TA-based algorithms, we sort the encrypted data items in each list based on their initial order (*i.e.*, their order in plaintext).

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.

4.2 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 1 shows how response time evolves, with increasing n, while the other parameters are set as default values described in Section 4. 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 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.

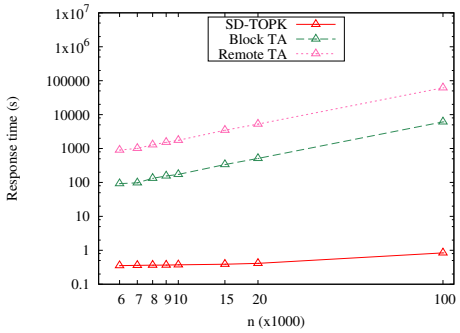

Fig. 1: Response time vs. number of database tuples

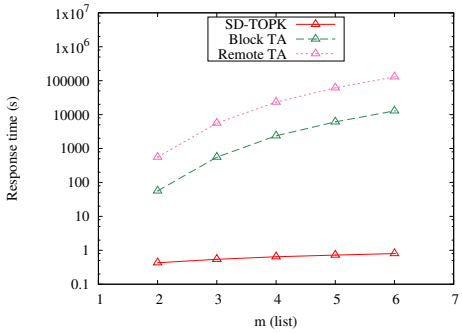

Fig. 2: Response time vs. number of lists

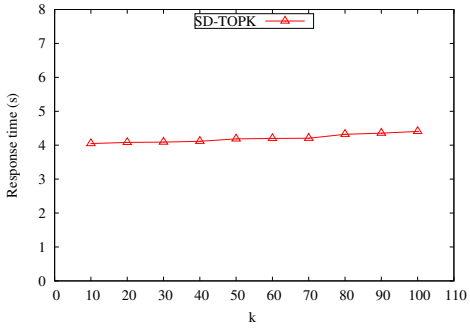

Fig. 3: Response time vs. k

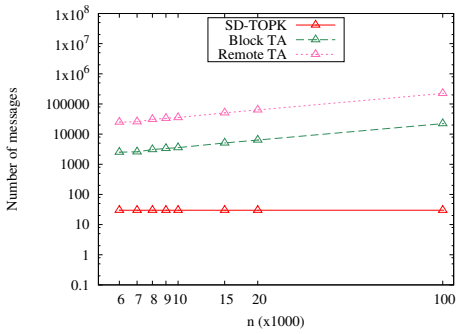

Fig. 4: Number of communicated messages vs. number of database tuples

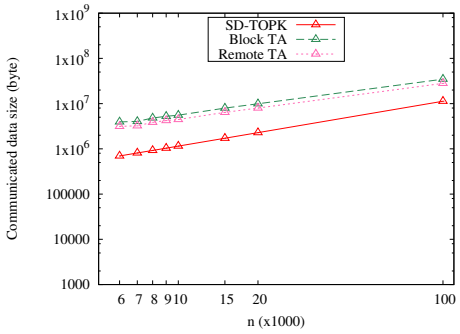

Fig. 5: Size of communicated data (in bytes) vs. number of database tuples

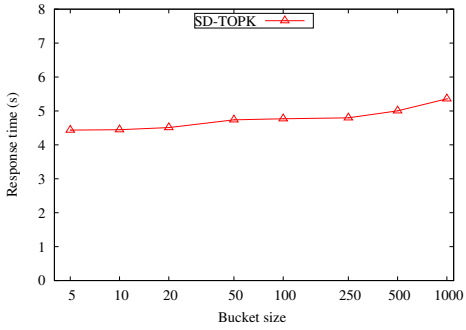

Fig. 6: Response time vs. bucket size

4.3 Effect of the Number of Lists

Figure 2 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 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.

4.4 Effect of k

Figure 3 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).

4.5 Effect of Bucket Size

Figure 6 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.

4.6 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 4 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 5 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.

4.7 Filtering Rate

We study the efficiency of the filtering algorithm of SD-TOPK by using different datasets. The results are shown in Table 1. 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 1: False positive elimination by the filtering algorithm of SD-TOPK over different databases

5 Related Work

In the literature, there has been some research work to process keyword queries over encrypted data , *e.g.*, [2, 13]. For example [2] and [13] 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.*, [5, 3, 4, 11, 15]. 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.*, [8, 7]. For example, in [8], 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. In [9], Kim et al. propose an approach for preserving the privacy of data access patterns during top-k query processing. In [14], 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. Thus 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.

CryptDB [12] 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) [1] 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.
In [16], the authors propose an approach for top-k query processing over encrypted data. The proposed approach assumes the existence of two non-colluding nodes s_1 and s_2 in two different clouds. One of the nodes, say s_2, has the decryption keys, and the other one, say s_1, stores the data. Top-k query processing proceeds by using the TA algorithm and accessing the encrypted data in s_1, such that after reading each data in s_1, its encrypted local scores are sent to the node s_2 (using a special protocol) where they are decrypted and compared with the TA threshold. Our assumptions about the distributed system are different. In our solution, we do not need to trust any node, and during the top-k query processing, we do not decrypt the encrypted data in the nodes of the system. In addition, the solution in [16] needs a lot of communications between cloud nodes (*i.e.*, at least two messages for each sorted/random access, which is even more than the TA-based algorithms compared with SD-TOPK).

6 Conclusion

In this paper, we proposed SD-TOPK, an efficient system that encrypts and outsources user data in a distributed system, and 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.

Acknowledgement

The research leading to these results has received funding from the European Union's Horizon 2020 - The EU Framework Programme for Research and Innovation 2014-2020, under grant agreement No. 732051

Bibliography

[1] P. Cao and Z. Wang. Efficient top-k query calculation in distributed networks. In *Proc. of ACM PODC*, pages 206–215, 2004.

[2] Y-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *ACNS*, pages 442–455, 2005.

[3] S. Choi, G. Ghinita, H-S. Lim, and E. Bertino. Secure knn query processing in untrusted cloud environments. *IEEE TKDE*, 26(11):2818–2831, 2014.

[4] X. Ding, P. Liu, and H. Jin. Privacy-preserving multi-keyword top-k similarity search over encrypted data. *IEEE TDSC*, (99):1–14, 2017.

[5] Y. Elmehdwi, B K. Samanthula, and W Jiang. Secure k-nearest neighbor query over encrypted data in outsourced environments. In *Proc. of IEEE ICDE*, pages 664–675, 2014.

[6] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[7] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu. Secure multidimensional range queries over outsourced data. *J. VLDB*, 21(3):333–358, 2012.

[8] B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In *VLDB*, pages 720–731, 2004.

[9] Hyeong-Il Kim, Hyeong-Jin Kim, and Jae-Woo Chang. A privacy-preserving top-k query processing algorithm in the cloud computing. In *Economics of Grids, Clouds, Systems, and Services (GECON)*, pages 277–292, 2016.

[10] R. Li, Alex X. Liu, Ann L. Wang, and B. Bruhadeshwar. Fast range query processing with strong privacy protection for cloud computing. *PVLDB*, 7(14):1953–1964, 2014.

[11] Xiaojing Liao and Jianzhong Li. Privacy-preserving and secure top-k query in two-tier wireless sensor network. In *Global Communications Conference (GLOBECOM)*, pages 335–341, 2012.

[12] Raluca A. Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: processing queries on an encrypted database. *Commun. ACM*, 55(9):103–111, 2012.

[13] D. Xiaodong Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE S&P*, pages 44–55, 2000.

[14] Jaideep Vaidya and Chris Clifton. Privacy-preserving top-k queries. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 545–546. IEEE, 2005.

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

[16] Haohan Zhu Xianrui Meng and George Kollios. Top-k query processing on encrypted databases with strong security guarantees. *arXiv:1510.05175v2*, 2016.