Top-k Query Processing Over Outsourced Encrypted Data

Sakina Mahboubi, Reza Akbarinia, Patrick Valduriez

To cite this version:

Sakina Mahboubi, Reza Akbarinia, Patrick Valduriez. Top-k Query Processing Over Outsourced Encrypted Data. [Research Report] RR-9053, INRIA Sophia Antipolis - Méditerranée. 2017, pp.24. lirmm-01502142v2

HAL Id: lirmm-01502142
https://hal-lirmm.ccsd.cnrs.fr/lirmm-01502142v2

Submitted on 23 May 2017

Top-k Query Processing Over Outsourced Encrypted Data

Sakina Mahboubi, Reza Akbarinia, Patrick Valduriez

RESEARCH REPORT
N° 9053
April 2017
Project-Teams Zenith

ISSN 0249-6399   ISRN INRIA/RR--9053--FR+ENG

Top-k Query Processing Over Outsourced Encrypted Data

Sakina Mahboubi[*], Reza Akbarinia[†], Patrick Valduriez[‡]

Project-Teams Zenith

Research Report n° 9053 — April 2017 — 24 pages

Abstract: Nowadays, cloud data outsourcing provides users and companies with powerful capabilities to store and process their data in third-party data centers. However, the privacy of the outsourced data is not guaranteed by the cloud providers. One solution for protecting the user data against security attacks is to encrypt the data before being sent to the cloud servers. Then, the main problem is to evaluate user queries over the encrypted data.
In this paper, we address the problem of top-k query processing over encrypted data, and propose an efficient approach called BuckTop. Our approach uses the bucketization technique to manage the encrypted data in the remote server. It includes a top-k query processing algorithm that works on the encrypted data of the buckets, and returns a set that contains the encrypted top-k results. It also has a filtering algorithm that efficiently eliminates the false positives in the server side.
We implemented BuckTop, and compared its response time for processing top-k queries over encrypted data with that of the TA algorithm over original (plaintext) data. Our results show excellent performance gains. They show that the response time of BuckTop over encrypted data is close to TA over plaintext data.

Key-words: top-k query, cloud, security, encrypted data

[*] Email: sakina.mahboubi@inria.fr
[†] Email: reza.akbarinia@inria.fr
[‡] Email: patrick.valduriez@inria.f

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Traitement de requêtes Top-k sur les donnés cryptées

Résumé : Aujourd'hui cloud computing fournit aux utilisateurs et aux entreprises des capacites puissantes pour stocker et traiter leurs donnees. Cependant, la confidentialité des données externalisées n'est pas garantie par les fournisseurs de cloud. Une solution pour protéger les données utilisateur contre les attaques de sécurité consiste à chiffrer les données avant d'être envoyée aux serveurs. Ensuite, le problème principal est d'évaluer les requêtes des utilisateurs sur les données cryptées.

Dans ce travail, nous abordons le problème du traitement des requêtes top-k sur les données chiffrées et proposons une approche efficace appelée BuckTop. Notre approche utilise la technique de bucketization pour gérer les données cryptées dans le serveur distant. Il comprend un algorithme de traitement de requêtes top-k qui fonctionne sur les données cryptées des seaux et renvoie un ensemble qui contient les résultats top-k cryptés. Il a également un algorithme de filtrage qui élimine efficacement les faux positifs du côté du serveur.

Nous avons mis en place BuckTop et comparé son temps de réponse pour traiter les requêtes top-k sur des données cryptées avec celles de l'algorithme TA sur des données originales (en clair). Nos résultats affichent d'excellents gains de performance. Ils montrent que le temps de réponse de BuckTop sur les données chiffrées est proche de TA sur les données en clair.

Mots-clés : top-l, cloud, sécurité

Contents

1 Introduction

Nowadays, cloud data outsourcing provides users and companies with powerful capabilities to store and process their data in third-party data centers. However, when a user stores her data in a public cloud, she loses the physical access control to the data. Thus, potentially sensitive data gets at risk of security attacks, e.g., from the employees of the cloud provider. According to a recent report published by the Cloud Security Alliance [9], security attacks are one of the main concerns for the cloud users.

One solution for protecting the user data against attacks is to encrypt the data before sending them to the cloud servers. Then, the challenge is to answer user queries over encrypted data. A naive solution for answering queries is to retrieve the encrypted database from the cloud to the client, decrypt it, and then evaluate the query over *plaintext (non encrypted)* data. This solution is not practical, in particular for large databases.

In this work, we are interested in processing top-k queries over encrypted data. These queries have attracted much attention in several areas of information technology such as sensor networks [42], information retrieval [37], data stream management systems [39, 31], spatial data analysis [5, 32, 33], and graph databases [29, 21, 18]. A top-k query allows the user to specify a number k, and the system returns the k tuples which are most relevant to the query. The relevance degree of tuples to the query is determined by a *scoring function*.

There have been many different approaches proposed for processing top-k queries over plaintext data. One of the best known approaches is TA [15] that works on sorted lists of attribute values. TA can find efficiently the top-k results because of a smart strategy for deciding when to stop reading the database. However, TA and all other efficient top-k approaches developed so far assume the existence of local scores of the data items (i.e. their attribute values) in plaintext, and there is no efficient solution capable of evaluating efficiently top-k queries over encrypted data.

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. [16], which allows one to do arithmetic operations over encrypted data. Using this type of encryption allows to compute the overall score of data items over encrypted data. However, existing fully homomorphic encryption methods are very expensive in terms of encryption and decryption time. In addition, they do not allow to compare the encrypted data, and to find the top-k results.

In this research report, we propose a new approach, called *BuckTop*, that efficiently evaluates top-k queries over encrypted data. BuckTop uses the bucketization technique to partition the encrypted data into a set of buckets before sending them to the server. This work includes the following contributions:

- A top-k query processing algorithm that works on the encrypted data of the buckets, and returns a set, which is proved to contain the encrypted data corresponding to the top-k results.
- An efficient filtering algorithm that filters in the server significantly the false positives included in the set returned by the top-k query processing algorithm. We prove theoretically the correctness of the filtering algorithm.
- A novel approach to obfuscate the boundaries of the buckets that contain the data scores, thus increasing the security of the buckets.

We implemented our approach, and compared its response time over encrypted data with a base algorithm and also with TA over original (plaintext) data. The experimental results show excellent performance gains for BuckTop. For example, the results show that the response time of BuckTop over encrypted data is close to TA over plaintext data. The results also illustrate that more than 99.9 % of the false positives can be eliminated in the server by BuckTop's filtering algorithm.

The rest of this report is organized as follows. In Section 2, we give the problem definition. In Section 3, we describe briefly the bucketization technique and some encryption schemes which we use in the report. In Section 4, we describe the system architecture. Then, in Section 5, we propose our BuckTop approach for top-k query processing over encrypted data. In Section 6, we report the performance evaluation results. Section 7 discusses related work, and Section 8 concludes.

2 Problem Definition

In this section, we first describe the adversary model which we consider, and then define the top-k queries. Finally, we state the problem which we address.

2.1 Adversary Model

In this work, we consider the honest-but-curious adversary model, where the adversary is inquisitive to learn the sensitive data without introducing any modification in the data or protocols. This model is widely used as the adversary model for many solutions proposed for secure processing of the different queries [25].

2.2 Top-k Queries

By a top-k query, the user specifies a number k, and the system should return the k most relevant answers to the users. 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* [15]. Let us define them formally.

Let D be a set of n data items, and $L_1, L_2, ..., L_m$ be m lists such that each list L_i contains n pairs of the form $(d, s_i(d))$ where $d \in D$ and $s_i(d)$ is a non-negative real number that denotes the *local score* of d in L_i. Each list L_i is sorted in descending order of its local scores.

The set of m lists is called a *database*. The *overall score* of each data item d is calculated as $ov(d) = f(s_1(d), s_2(d), ..., s_m(d))$ where f is a given *scoring function*. In other words, the overall score is the output of f where the input is the local scores of d in all lists. We assume that the function f is monotonic, thus if $x_i \leq x'_i$ then $f(x_1, x_2, ..., x_m) \leq f(x'_1, x'_2, ..., x'_m)$. Many popular aggregation functions such as MIN, MAX, AVG are monotonic.

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. Formally, the top-k query result is a set of elements $D' \subseteq D$ such that $|D'|= k$, and $\forall d' \in D' \wedge \forall d \in (D - D') \Rightarrow ov(d') \geq ov(d)$.

The sorted lists model for top-k query processing is simple and general. 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 this query, it is sufficient to have a sorted (indexed) list of the values of each attribute involved in the scoring function, and return the k tuples whose overall scores in the lists are the highest.

For processing top-k queries over sorted lists, two modes of access are usually used [15]. The first is the *sorted (sequential) access* that allows us to sequentially access the next data item in the sorted list. This access begins with the first item in the list. The second is the *random access* by which we look up a given data item in the list.

2.3 Problem Statement

The problem, which we address is top-k query processing over encrypted data. Let D be a database, and $e(D)$ be its encrypted version such that each data $c \in e(D)$ is the ciphertext of a data $d \in D$, i.e. $c = Enc(d)$ where $Enc()$ is an encryption function. The database $e(D)$ is stored in a remote server. Given a number k and a scoring function f, our goal is to develop an algorithm A, such that when A is executed over the database $e(D)$, its output contains the ciphertexts of the top-k results, i.e. those that can be found by executing a correct top-k algorithm over the database D.

A naive approach for top-k query processing over the encrypted database $e(D)$ is to retrieve it from the remote server, decrypt all of its data, run an existing top-k algorithm over the plaintext data, and return the top-k results to the user. However, this approach is not practical, particularly for very large databases.

3 Background

In this section, we first introduce the bucketization technique, and then describe different encryption methods used in our work.

3.1 Bucketization Technique

In this work, we take advantage of the bucketization technique for managing the encrypted data in the server. This technique consists of partitioning the data (e.g., attribute values) into buckets. There are several methods for partitioning the values of an attribute, for example dividing the attribute domain to almost equal intervals or creating partitions with equal sizes. We use one of the existing approaches, e.g. one of those described in [19], for creating the buckets around the scores of the sorted lists, and storing each data score in its corresponding bucket.

3.2 Encryption Schemes

In our approach, we use *symmetric encryption* schemes that use a single key, called *private key*, for both encryption and decryption operations. The sender uses the private key to encrypt the plaintext and sends the ciphertext to the receiver. The receiver applies the same key to decrypt the message.

We distinguish two types of symmetric schemes: *deterministic* and *probabilistic*. *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. A probabilistic encryption scheme can be developed by simply adding some random bits to the plaintext, and then encrypting the result using a deterministic encryption scheme. The random bits are discarded after decrypting the ciphertext.

4 System Architecture and a Basic Top-k Approach

In this section, we first present the architecture of our system. Then, we propose a basic top-k query processing approach, called EncFA.

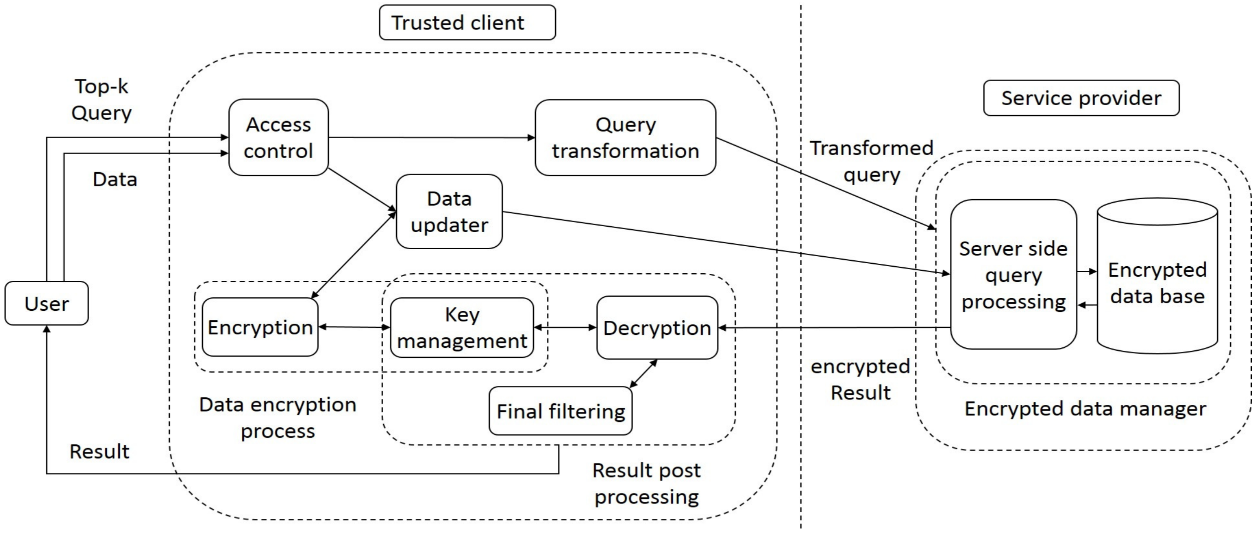

Figure 1: System architecture

4.1 Architecture

The architecture of our outsourcing system is composed of two parts (see Figure 1):

- **Trusted client.** It is responsible for encrypting the user data, decrypting the data 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 his demand is rejected. Otherwise, the issued query is transformed to a query that can be executed over the encrypted data. For example, suppose we have a relation R with the following attributes: att_1, att_2,â¦, att_m and the user issues the following query:

  *SELECT * FROM R ORDERED BY* $f(att_1, , att_m)$ *STOP AFTER k*

  This query is transformed to:

  *SELECT * FROM E(R) ORDERED BY F(E(att*$_1$*) , ..., E(att*$_m$*)) STOP AFTER k.*

  The transformed query is sent to the service provider, and the received results are post-processed, e.g. decrypted, before being returned to the user.

- **Service provider.** It stores the encrypted data, executes on them the algorithms provided by the trusted client, and returns the results to it.

4.2 EncFA: A Basic Top-k Query Processing Approach

Here, we propose a simple approach, called EncFA, that uses the FA algorithm [13] in the server for processing top-k queries over encrypted data. Our main contribution, called BuckTop, is presented in the next section.

Let us first describe how the local scores are encrypted. By EncFA, the trusted client uses a *probabilistic* encryption scheme to encrypt the local scores (attribute values) of the data items in the lists. It also uses a *deterministic* scheme for encrypting the ID of data items. Then, it sends the encrypted data IDs and scores to the service provider in the same order as they were before being encrypted. Notice that all we need is that the lists in the server be sorted in the same order they were before encryption. We do not need to compare the encrypted scores, thus no need for an order preserving encryption (OPE) scheme.

Given a top-k query Q with a scoring function f, EncFA uses the FA algorithm for processing top-k queries as follows. It performs sorted access in parallel to each sorted list, and maintains the encrypted data items and their encrypted local scores in a set Y. If there are at least k data items in Y such that each of them has been seen in each of the m lists, then it stops doing sorted access to the lists.

Then, for each data item d involved in Y, and each list L_i, it performs random access to L_i to find the encrypted local scores of d in L_i (if it has not been seen yet). EncFA sends Y to the trusted client which decrypts the local scores of each item $d \in Y$, computes their overall scores, and sends to the user the k items with the highest overall scores.

Theorem 1 *Given a top-k query with a monotonic scoring function, EncFA returns a set that includes the encrypted top-k elements.*

The proof is given in Appendix A.

The main idea behind EncFA is that the FA algorithm is agnostic with respect to the scoring function, thus it works correctly even on the encrypted data if they are sorted in the same order as the plaintext data. Notice that many other algorithms do not have this characteristic, e.g. the TA algorithm.

5 BuckTop: An Efficient Approach for Top-k Query Processing over Encrypted Data

The EncFA approach, presented in the previous section, evaluates correctly the top-k queries over encrypted data. But, as shown by our experiments reported in Section 6, there may be a high number of false positives which are sent from the server to the client, and this renders EncFA inefficient in practice. This is why, we propose the BuckTop approach which is much more efficient than EncFA.

Our BuckTop approach manages the encrypted data in the server using the bucketization technique. The data storage and query processing by BuckTop is done in four main steps: 1) bucket creation and data encryption; 2) top-k query processing over encrypted data in the server; 3) result filtering in the server; 4) result post-processing in the trusted client.

5.1 Bucket Creation and Data Encryption

Before sending the data to the service provider, the trusted client partitions each sorted list into buckets using a bucketization mechanism. Let b_1, b_2, ..., b_t be the created buckets for a list L_j. Each bucket b_i has a lower bound, denoted by $min(b_i)$, and an upper bound, denoted by $max(b_i)$. The lower bound of the bucket b_i is equal to the upper bound of the next bucket, i.e., $min(b_i) = max(b_{i+1})$. A data item d is in the bucket b_i, if and only if its local score in the list L_j is between the lower and upper bounds of the bucket, i.e. $min(b_i) \leq s_j(d) < max(b_i)$.

The trusted client chooses two symmetric encryption schemes. The first one is deterministic used to encrypt the ID of the data items in each database list. The second encryption scheme is probabilistic used to encrypt the local scores of data items.

After encrypting the data IDs and local scores, the trusted client creates the buckets in each list, puts the encrypted local scores in their corresponding bucket, and sends the buckets and their involved encrypted data to the service provider. The buckets are sent 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.*

For the ease of presentation, in the basic version of BuckTop, we assume that the lower and upper bounds of the buckets are known by the service provider, i.e. they are sent to the service provider when sending the buckets. In Section 5.5, this assumption is relaxed by obfuscating the lower/upper bounds using secret numbers.

5.2 Top-k Query Processing

Given a top-k query Q including a number k and a scoring function f, the top-k query processing algorithm of BuckTop performs the following steps in the service provider for processing Q:

1. Do sorted access in parallel to the lists, to read a bucket from each list. For each data item d seen in a bucket in a list, do random access in the other lists to find the encrypted score and the bucket of d in each list. Let b_1, b_2, ..., b_m be the buckets involving d in the lists L_1, L_2, ..., L_m. Compute a minimum overall score for d, denoted as $min\_ovl(d)$, by applying the scoring function on the lower bound of the buckets that contain d in different lists. Formally, $min\_ovl(d) = f(min(b_1), min(b_2), ..., min(b_m))$. Then, store d, its encrypted local scores, and its $min\_ ovl$ score in a set Y.

2. For each list L_i, let b'_i be the last bucket seen under sorted access in L_i up to now. Let threshold be defined as follows: $TH = f(min(b'_1), min(b'_2), ..., min(b'_m))$. In other words, the threshold is computed by applying the scoring function on the lower bounds of the last seen buckets in the lists.

3. If the set Y contains k elements that have a $min\_ ovl$ score higher than the threshold, then stop doing sorted access in the lists. Otherwise, go to step one.

At the end of the algorithm, the set Y includes the encrypted top-k data items (see the proof below). This set can be sent to the trusted client, which decrypts its involved elements, computes the overall scores of the items, removes the false positives (i.e., the items that are in Y but not among the top-k results), and returns the top-k items to the user.

Let us prove that the BuckTop top-k query processing works correctly. We first show that the minimum overall score of any data item d, i.e. $min\_ovl(d)$, which is computed based on the lower bound of its buckets, is less than or equal to its overall score. We also show that the maximum overall score of d, i.e. $max\_ovl(d)$, is higher than or equal to its overall score.

Lemma 1 *Given a monotonic scoring function f, 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 monotonic scoring function f, 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 output of BuckTop contains the encrypted top-k data items.

Theorem 2 *Given a top-k query with a monotonic scoring function f, the output of BuckTop contains the encrypted top-k results.*

Proof. Let Y be output of the BuckTop top-k query processing algorithm, i.e. the set that contains all the encrypted data items seen under sorted access when the algorithm ends. We show that each data item d that is not in Y ($d \notin Y$), has an overall score that is less than or equal to the overall score of at least k data items in Y. For each list L_i, let s_i be the local score of d in the list L_i. Let b'_i be the last bucket seen under sorted access in the list L_i, i.e. when the algorithm ends. Since d is not in Y, it has not been seen under sorted access in the lists. Thus, its involving buckets are after the buckets seen under sorted access by the algorithm. Therefore, we have $s_i < min(b'_i)$ for $1 \leq i \leq m$, i.e. the local score of d in each list L_i is less than the lower bound of the last bucket read under sorted access in L_i. Since the scoring function is monotonic, we have $f(s_1, ..., s_m) < f(min(b'_1), min(b'_2), ..., min(b'_m)) = TH$. Thus, the overall score of d is less than the threshold used in the BuckTop algorithm. When the algorithm stops, there are at least k data items in Y whose minimum overall scores are greater than or equal to the threshold. Thus, by using Lemma 1, their overall scores are at least the threshold. Therefore, their overall score is greater than or equal to that of the data item d. □

5.3 Filtering

In the set Y returned by BuckTop, the number of false positives may be high. Here, we propose a server side algorithm to filter from Y the data items that have no chance to be among the top-k items. As shown by our experimental results, this filtering algorithm can eliminate most of the false positives (more than 99% in the different tested datasets).

In the filtering algorithm, we use the *maximum overall score*, denoted by max_ovl of each data item. This score is computed by applying the scoring function over the upper bound of the buckets involving the data item in the lists. The filtering algorithm proceeds as follows:

1. Let $Y' \subseteq Y$ be the k data items in Y that have the highest min_ovl scores. Let d' be the data item that has the minimum min_ovl score in Y'.

2. For each item $d \in Y$, compute its maximum overall score, i.e. $max\_ovl(d)$, by applying the scoring function on the upper bound of the buckets involving d in the lists. Formally, let $max(b_i)$ be the upper bound of the bucket involving d in the list L_i. Then, $max\_ovl(d) = f(max(b_1), max(b_2), ..., max(b_m))$. If the maximum overall score of d is less than or equal to the minimum overall score of d', then remove d from Y. In other words, if $max\_ovl(d) \leq min\_ovl(d') \Rightarrow Y = Y - \{d\}$

After running the filtering algorithm, the service provider sends Y to the trusted client which decrypts its involved elements and filters the remaining false positives (if any).

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

Theorem 3 *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, by using Lemmas 1 and 2, the overall score of d is less than or equal to that of at least k data items. Therefore, we can eliminate d. □

5.4 Example

Consider the encrypted database shown in Figure 2, and a top-k query with $k = 3$ and a scoring function that computes the sum of the local scores in the sorted lists.

List 1			List 2			List 3		
bucket ID	enc data item	enc local score	bucket ID	enc data item	enc local score	bucket ID	enc data item	enc local score
B_{11}	$E(d1)$	$E(27)$	B_{21}	$E(d6)$	$E(28)$	B_{31}	$E(d2)$	$E(22)$
B_{11}	$E(d3)$	$E(30)$	B_{21}	$E(d3)$	$E(29)$	B_{31}	$E(d3)$	$E(25)$
B_{11}	$E(d6)$	$E(26)$	B_{21}	$E(d2)$	$E(26)$	B_{31}	$E(d6)$	$E(27)$
B_{12}	$E(d2)$	$E(15)$	B_{22}	$E(d1)$	$E(24)$	B_{32}	$E(d5)$	$E(21)$
B_{12}	$E(d8)$	$E(20)$	B_{22}	$E(d7)$	$E(21)$	B_{32}	$E(d1)$	$E(20)$
B_{12}	$E(d5)$	$E(24)$	B_{22}	$E(d4)$	$E(19)$	B_{32}	$E(d9)$	$E(18)$
B_{13}	$E(d4)$	$E(14)$	B_{23}	$E(d5)$	$E(16)$	B_{33}	$E(d8)$	$E(17)$
B_{13}	$E(d7)$	$E(12)$	B_{23}	$E(d9)$	$E(13)$	B_{33}	$E(d7)$	$E(14)$
B_{13}	$E(d9)$	$E(11)$	B_{23}	$E(d8)$	$E(10)$	B_{33}	$E(d4)$	$E(11)$
...	...	...	...	...	...	...	...	...

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

Figure 2: Example of an encrypted database

Read Buckets	data items	min_ovl	TH	continue?
1	$E(d3)$	72	/	/
	$E(d1)$	60.3	/	/
	$E(d6)$	72	/	/
	$E(d2)$	62.2	72	YES
2	$E(d5)$	41.5	/	/
	$E(d8)$	33.8	/	/
	$E(d7)$	38.7	/	/
	$E(d4)$	38.7	/	/
	$E(d9)$	37.4	50.5	NO

Table 1: Applying BuckTop top-k query processing algorithm over the database of Figure 2

Table 1 shows the result of applying the BuckTop top-k query processing algorithm over the database of Figure 2. BuckTop stops sorted accesses after reading the 2nd bucket of each list. In the 2nd bucket, the threshold is 50.5 and the minimum overall score of d_3, d_1, d_6 and d_2 is greater than or equal to the threshold. The set Y contains all the data items that have been seen under sorted access, i.e. $Y = \{E(d3), E(d1), E(d6), E(d2), E(d5), E(d8), E(d7), E(d4), E(d9)\}$. The filtering algorithm is applied on the data items of Y. The k data items that have the highest *min_ ovl* scores in Y are $Y' = \{d_3, d_6, d_2\}$. Among the data items in Y', the minimum *min_ ovl* belongs to d_2, which is $min\_ovl(d_2) = 62.2$. For filtering, we need to compare *max_ ovl* score of each data item in *Y* - *Y'* with $min\_ovl(d_2)$. The result of filtering is shown in Table 2.

False positives	max-ovl	comparison with min-ovl	eliminated?
E(d1)	77.6	$max\_ovl \geq 62.2$	NO
E(d5)	62.1	$max\_ovl \leq 62.2$	YES
E(d8)	57.9	$max\_ovl \leq 62.2$	YES
E(d7)	55.6	$max\_ovl \leq 62.2$	YES
E(d4)	55.6	$max\_ovl \leq 62.2$	YES
E(d9)	52.2	$max\_ovl \leq 62.2$	YES

Table 2: Example of running the filtering algorithm

We find that we can eliminate d_5, d_8, d_7, d_4 and d_9 from Y because they have a *max_ ovl* score less than *min_ ovl* score of d_2. As a result, after the filtering, Y will contain the data items $Y = \{E(d_1), E(d_2), E(d_3), E(d_6)\}$. This set includes only one false positive, i.e. d_2, which will be eliminated in the client side.

5.5 Obfuscating Bucket Limits

A drawback of the basic version of BuckTop, presented until now, is that the limits of the buckets are disclosed to the adversary (server). To strengthen the security of our approach, we change the bucket limits as follows. We choose two random prime numbers a and c. These numbers must be kept secret in the trusted client. Before sending the database to the service provider, 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 prime number a, and then adds the secret prime number c to the result. These obfuscated bucket limits are sent to the service provider together with the encrypted IDs and scores.
By the above strategy, we hide the limits of the buckets from the service provider. But, a question remains to answer: does BuckTop work correctly if it uses the changed lower/upper bounds? The answer to this question is positive.The intuition is that the stop strategy of BuckTop remains valid, if we multiply and add all bucket limits to the same positive numbers.
The following theorem proves that the BuckTop top-k query processing algorithm works correctly if it uses the perturbed lower bounds.

Theorem 4 *Assume a top-k query with a monotonic scoring function f. If we change the lower bound of the buckets by using Equation 1, then the output of BuckTop will involve the top-k results.*

Proof. Let Y be the output of the BuckTop top-k query processing algorithm. We show that each data item d that is not in Y ($d \notin Y$), has an overall score that is less than or equal to the overall score of at least k data items involved in Y. Let $Y' \subseteq Y$ be the k data items whose minimum overall score is higher than the threshold when the algorithm ends. Let $d' \in Y'$ be the data item that has the lowest overall score among the data items involved in Y'. 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. Let $b_1, ..., b_m$ be the last buckets seen by the algorithm before it ends. Then, from the stop condition of BuckTop we have: $TH = f(min(b_1) * a + c, ..., min(b_m) * a + c)$ $\leq f(min(b'_1) * a + c, ..., min(b'_m) * a + c)$
Since f is monotonic and the numbers a and c are positive, we have:

$$f(min(b_1), ..., min(b_m)) \leq f(min(b'_1), ..., min(b'_m)) \tag{3}$$

Before changing the lower bounds, the local score of d' in each list is higher than or equal to the lower bound of its bucket, thus we have:

$$f(min(b'_1), ..., min(b'_m)) \leq f(s'_1, ..., s'_m) \tag{4}$$

By comparing Equations 3 and 4, we have:
$f(min(b_1), ..., min(b_m)) \leq f(s'_1, ..., s'_m) = ovl(d')$
In the right hand side of the above equation, we have the overall score of d'. Now, we show that the left hand side of the above equation is higher than or equal to the overall score of a data d that has not been seen by the algorithm. Let s_i be the (plaintext) local score of d in the list L_i. Since d has not been seen by the algorithm, its bucket in L_i is after b_i that is the last bucket seen by the algorithm. Thus, we have $s_i \leq min(b_i)$ for $1 \leq i \leq m$. Therefore, since f is monotonic, we have:

$$f(s'_1, ..., s'_m) \leq f(min(b_1), ..., min(b_m)) \tag{5}$$

In other words, $ov(d) \leq ov(d')$. Thus, the overall score of any unseen data item d is less than or equal to that of at least k data items involved in Y. Therefore, Y contains the top-k results, and the proof is done. □
Below, we prove that the filtering algorithm works correctly, if it uses the obfuscated bucket limits.

Theorem 5 *Assume a top-k query with a monotonic scoring function f. If we modify the lower bound of the buckets by using Equations 1 and 2, then the filtering algorithm does not remove any top-k result.*

Proof. Let Y be the output of BuckTop algorithm, and $Y' \subseteq Y$ be the k data items in Y that have the highest *min_ovl* scores. Let d' be the data item in Y'. 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 choose a data item d that is a top-k result and has been removed by the filtering algorithm. We show that this assumption yields to a contradiction. In each list L_i, let b_i and s_i be the bucket and local score of d_i in the list. Since, d has been removed from the list, its maximum overall score using the modified limits is lower than or equal to that of d'. Thus, we have:
$f(max(b_1) \times a + b, ..., max(b_m) \times a + b) \leq f(min(b'_1) \times a + c, ..., min(b'_1) \times a + c)$
Since the coefficients a and c are positive, the monotonicity of f implies that:
$f(max(b_1), ..., max(b_m)) \leq f(min(b'_1), ..., min(b'_1))$.
Therefore, we have: $max\_ovl(d) \leq min\_ovl(d')$. Then by using Lemmas1 and 2, we have: $ovl(d) \leq ovl(d')$. In other words, the overall score of d is less than that of any data item involved in Y'. Thus, d is not a top-k result and can be removed. □

5.6 Update Management

Let us explain how the encrypted data can be updated in the server. In our system, updating a data is done by deleting the old data scores and then inserting its new scores. Let d be the data to be updated and the new scores in the lists L_1, ..., L_m are s_1, ..., s_m respectively.
To delete the old scores of d from the server, it is sufficient to encrypt the ID of d using the key which has been used for encrypting the data IDs, and then asking the server to find the encrypted ID in the lists and then remove the pairs $(E(ID), E(score))$ from the lists.
Inserting the new scores is done as follows. The trusted client uses the metadata of the buckets (i.e., the lower bounds), and for each list L_i, it calculates the bucket of the list to which the score s_i should by 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 which are used for encrypting the ID and scores. Afterwards, for each list L_i, it creates the pairs $(E(ID), E(s_i))$, and asks the server to put the pair in the bucket b_i.

5.7 Security Analysis

Let us now analyze the security of BuckTop by considering the information that can be leaked to the adversary. For each case of leakage, we propose some advices to reduce the risk of disclosing sensitive data to the adversary.

5.7.1 Partial Order Leakage in Buckets

In BuckTop, we use the bucketization technique for managing the data in the server. Inside of 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). This may help the adversary to obtain rough information about the sensitive data of individuals if she has some background information about their data. *For example, if an adversary A knows the age of a person u, then A may estimate u's salary with*

some confidence probability. We show that the confidence probability depends inversely on the size of buckets. Notice that we assume that the probabilistic encryption scheme is unbreakable, thus the adversary can not compute the exact value of the u's salary, but just an estimation.

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' 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 she can make a good estimation of u's value, e.g. using some statistical 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 bucketized 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 $P(L_1 = a \wedge L_2 = x)$ be the probability that A guesses correctly the lists representing the attributes a and s. Let m be the number of bucketized 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 bucketized list of an attribute is $\frac{1}{m}$. Therefore, the probability of correctly guessing the lists representing both attributes a and s is:

$$P(L_1 = a \wedge L_2 = x) = \frac{1}{m \times (m-1)} \tag{6}$$

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:

$$P(ID = u) = \frac{1}{|size(b)|} \tag{7}$$

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 she guesses correctly the list of s), and then he can roughly estimate the u's value in s.

Let $P(b_{s,u}|a)$ be the confidence 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. Using the above discussion and Equations 6 and 7, we have:

$$P(b_{s,u}|a) \leq \frac{1}{size(b) \times m \times (m-1)} \tag{8}$$

where m is the number of bucketized lists in the server, and $size(b)$ is the size of the bucket containing u in the list representing a. Equation 8 shows that the bigger the size of the buckets, the higher the security of buckets. Thus, to reduce the risk of disclosing the sensitive data to the adversary, we must avoid using very small bucket sizes. The risk of privacy violation is the highest, when the bucket size is one (which is equivalent to the EncFA approach).

However, notice that choosing very big buckets decreases the performance of query processing, since it increases the number of false positives (see Section 6). Therefore, the size of the buckets should be taken based on the user requirements in terms of privacy (e.g., required confidence probabilities) and performance (e.g. required response time).

5.7.2 Leakage of the Number of Asked Results

In our approach, the information about the number of asked results, i.e. k, is leaked. We can perturb 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 server as the number of required results. After receiving the encrypted results from the service provider, the trusted client filters the result set and sends only k results to the user.

5.7.3 Leakage of 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 that is sent to the service provider. 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 which 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. The above strategy improves the security, without having any impact on the performance of top-k queries.

6 Performance Evaluation

In this section, we evaluate the performance of BuckTop using synthetic and real datasets. We study the effect of different parameters such as the number of data items in each lists, the number of the database lists, the number of data items requested, etc.
In the rest of this section, we first describe the experimental setup, and then report the results of our experiments.

6.1 Experimental Setup

We implemented our top-k query processing approach in Java. We have done our tests on real and synthetic datasets. As in some previous work on encrypted data (*e.g.*, [25]), 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 [25], we decompose this attribute into 6 attributes (year, month, day, hour, minute, second), and then create a database with the following schema R(ID, year, month, date, hour, minute, second), where ID is the tuple identifier. In addition to the real dataset, we have also generated random datasets using uniform and Gaussian distributions.
To the best of our knowledge, in the literature there is no efficient algorithm for processing top-k queries over encrypted data (see Related Work Section). We compare BuckTop with the two following algorithms:

- *TA over plaintext data.* The objective is to show the overhead of running BuckTop over encrypted data compared to an efficient top-k algorithm over plaintext data.
- *OPE.* This is an order preserving encryption approach in which the encrypted scores are kept in the server in the same order as their initial order (*i.e.*, before encryption). For top-k

query processing, the *FA algorithm* [13] is applied on the sorted encrpted data. Since, FA is agnostic with respect to the scoring function, it returns a set that includes the encrypted top-k items. The returned encrypted data items are decrypted in the trusted client, false positives are removed, and the top-k items are returned to the user.

In our experiments, we have two versions of each database: 1) the plaintext database used for running TA; 2) the encrypted database used for running BuckTop and OPE.
For data encryption in BuckTop, we use the two following algorithms: XOR [7] to encrypt the identifiers of the database items, and Blowfish [30] to encrypt the local scores of the database items. For the OPE approach, we simply sort the encrypted data items in each list based on their initial order.
In our performance evaluation, 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 of BuckTop. The default value for n is 2M items. Unless otherwise specified, m is 5, k is 50, and $bsize$ is 20. In our tests,the default database is the synthetic uniform database.
To evaluate the performance of our approach, we measured the following metrics:

- **Server top-k time:** the time required for the server to find the set that includes the top-k results, *i.e.*, the set Y.
- **Total response time:** the total time elapsed between the time when the query is sent to the server and the time when the k decrypted results are returned to the user. This time includes the server top-k time, the filtering, and the result post-processing in the client (*e.g.*, decryption).
- **Filtering rate:** the number of false positives eliminated by the filtering algorithm in the server side.

Our experiments have been done using a server with 16 GB of main memory and Intel Core i7-5500 @ 2.40Ghz as processor.

6.2 Effect of the Number of Data Items

In this section, we compare the performance of TA over plaintext data with BuckTop and OPE over encrypted data, while varying the number of data items, *i.e.*, n.
Figure 3 shows how server top-k time evolves, with increasing n, and the other parameters set as default values described in Section 6.1. The server top-k time of all approaches increases with n. But, OPE takes more time than the two other approaches, because it stops deeper in lists, and thus reads more data.
Figure 4 shows the total response time of BuckTop, OPE and TA while varying n, and the other parameters set as default values. Note that the figure are is in logarithmic scale. TA does not need to decrypt any data, so its response time is almost the same as its server time. The response time of BuckTop is slightly higher than its server top-k time, as in addition to top-k query processing it performs the filtering in the server and also needs to decrypt at least k data items. We see that the response time of OPE is much higher than its server top-k time. The reason is that OPE returns to the trusted client a lot of false positives, which should be decrypted, and removed from the final result set. But, this is not the case for BuckTop as its filtering algorithm removes almost all the false positives in the server (see the results in Section 6.7), thus there is no need to decrypt them.

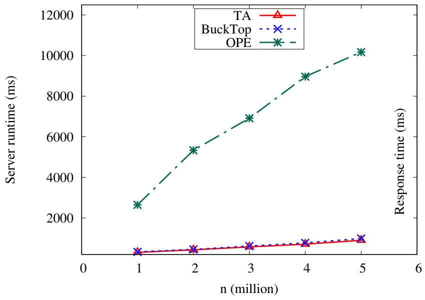

Figure 3: Server top-k time vs. number of database tuples

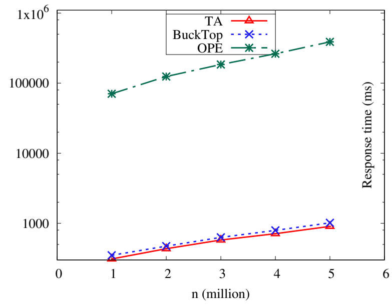

Figure 4: Response time vs. number of database tuples

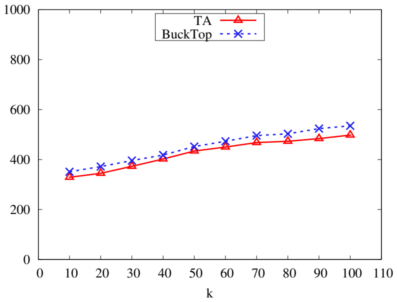

Figure 5: Response time vs. k

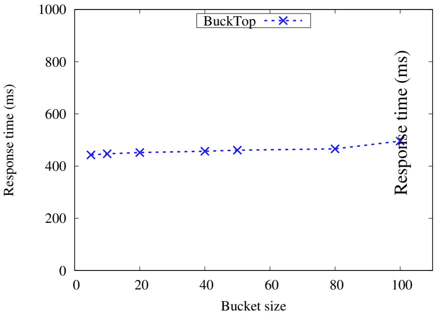

Figure 6: Response time vs. bucket size

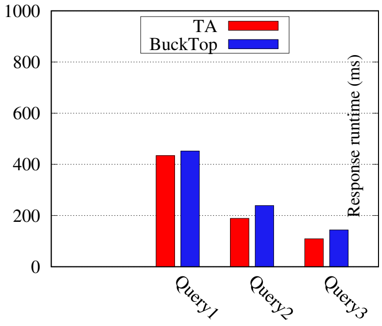

Figure 7: Response time using different queries

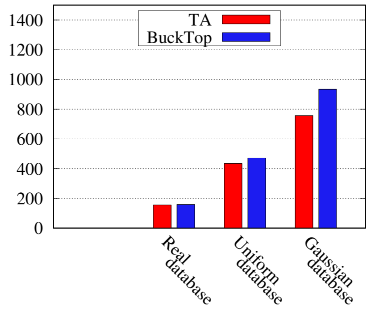

Figure 8: Response time using different datasets

Database size (M)	1	2	3	4	5	6
Number of false positives before filtering	540779	547289	557467	561637	537852	580299
Rate of eliminated false positives	100%	100%	100%	99.99%	99.99%	100%

A: over Uniform dataset

Database size (M)	1	2	3	4	5	6
Number of false positives before filtering	96267	192554	188928	384823	480176	575832
Rate of eliminated false positives	99.98%	99.99%	99.99%	99.99%	99.99%	99.99%

B: over Real dataset

Database size (M)	1	2	3	4	5	6
Number of false positives before filtering	187802	322931	486111	584378	679678	738482
Rate of eliminated false positives	99.94%	99.96%	99.97%	99.98%	99.98%	99.98%

C: over Gaussian dataset

Table 3: False positive elimination by our filtering algorithm in the server over different databases. The filtering algorithm performs an excellent job, this is why we need to decrypt in the client side only a small number of data (*i.e.*, only the top-k data and a very small number of false positives, if any).

6.3 Effect of k

Figure 5 shows the total response times of BuckTop with increasing k, and the other parameters set as default values. We observe that with increasing k the response time increases. The reason is that Bucktop needs to go deeper in the lists to find the top-k results. 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).

6.4 Effect of Bucket Size

Figure 6 reports the response time of BuckTop when varying the size of buckets, and the other parameters set as default values. We observe that the response time increases when the bucket size increases. The reason is that the top-k query processing algorithm of Bucktop reads more data in the lists, because the data are read bucket by bucket. In addition, increasing the bucket size increases the number of false positives to be removed by the filtering algorithm, and eventually decrpting the none eliminated false positives in the client side.

6.5 Effect of Different Queries

We evaluated the effect of the scoring function on the performance of our approach. For this, we tested three different queries with different scoring functions. In the first query, noted as Q_1, the scoring function is $sf_1(s_1 + s_2 + ... + s_n) = s_1 + s_2 + ... + s_n$. In this query, we have the same coefficient (impact) for all scoring attributes. In the 2nd and 3rd queries, there is a higher skew

in the coefficients: in Q_2, we set $sf_2(s_1 + s_2 + ... + s_n) = 1 \times s_1 + 2 \times s_2 + ... + n \times s_n)$, and in Q_3 we set $sf_3(s_1 + s_2 + ... + s_n) = 2^1 \times s_1 + 2^2 \times s_2 + ... + 2^n \times s_n$.
Figure 7 shows the response time of TA and BuckTop using the three queries. Note that in our experiments the default query is Q_1. We observe that for the query Q_3, BuckTop performs better than the other queries. The reason is that in Q_3, only one or two attributes are the dominating factors of the scoring function (*i.e.*, those with very high coefficients). In this case, the top-k processing algorithm takes less time to stop, and this is why the response time of BuckTop with Q_3 is lower than the other queries.

6.6 Performance over Different Datasets

We study the effect of the datasets on the performance of BuckTop and TA using different datasets: synthetic datasets with uniform and Gaussian distributions, and real dataset (Gowalla). Figure 8 shows the response time of TA and BuckTop over different datasets, while other parameters are set as default values. We see that over the uniform and real datasets, BuckTop and TA have approximately the same response times. Over the Gaussian dataset, the response time of BuckTop is a little higher than TA. The reason is that over this dataset the number of false positives is higher than the other datasets, thus more encrypted data should be decrypted by the trusted client.

6.7 Effect of the Filtering Algorithm

BuckTop's filtering algorithm is used to eliminate/reduce the false positives in the server. We study the filtering rate by increasing the size of the dataset. For the uniform synthetic dataset, the results are shown in Table 3-*A* . For datasets with up to three million data items, the filtering method eliminates 100% of the false positives, and the server returns to the trusted client only the k data items that are the result of the query. For larger datasets, the server filters up to 99,99% of the false positives. By using the Gaussian dataset, we obtain the results shown in Table 3-*C*. We see that around 99,94% of false positives are eliminated.
Over the real dataset, Table 3-*B* shows the filtering rate. We observe that the filtering algorithm eliminates 99,99% of false positives. Thus, the filtering algorithm is very efficient over all the tested datasets. However, there is a little difference in the filtering rate for different datasets 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.

7 Related Work

Efficient processing of top-k queries is important for many applications such as information retrieval [37], sensor networks [42], data stream management systems [31, 39], crowdsourcing [8, 44], string matching [22, 38], spatial data analysis [5, 32, 33], temporal databases [28], graph databases [18, 21, 29], uncertain data [11, 34, 35], etc.
A first important paper in top-k query processing is [13], which models the general problem of answering topk queries using lists of data items sorted by their local scores and proposes a simple and efficient algorithm, Faginâs algorithm (FA), which works on sorted lists. One of the most efficient top-k algorithms is the TA algorithm, which was proposed by several groups [14, 17, 27]. Several threshold based algorithms have been proposed for processing top-k queries in different environments, e.g. [2, 1, 10, 23, 26]. However, all these algorithms assume that the data scores are available as plaintext, and not encrypted.

There have been some work to process keyword queries over encrypted data , e.g. [3, 4, 36]. For example [4] and [36] propose matching techniques to search for any word 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 query processing, e.g. [12, 6, 40, 43]. The problem is to find k points in the 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 has been used in the literature for answering range queries over encrypted data, e.g. [20, 19, 24]. For example, in [20], Hore et al. use this technique, and propose optimal solutions for distributing the encrypted data of a database to the buckets in order to guarantee a good performance by reducing the number of false positives while preserving a high security level. The techniques developed in [20, 19, 24] can be used in our system for an optimal distribution of the encrypted data in the buckets.

The only paper which we found about top-k query processing over encrypted data is [41] published in arXiv.org. The proposed architecture assumes the existence of two non-colluding servers s_1 and s_2 in two different clouds. One of the servers, 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 server s_2 (using a special protocol) where they are decrypted and compared with the TA threshold. Our assumptions about the cloud are different. In our solution, we do not need to trust on any remote server, and during the top-k query processing, we do not decrypt the encrypted data in the cloud servers. In addition, the solution in [41] needs a lot of communications between remote servers (i.e., at least two messages after each sorted access). This solution is not efficient and incurs a high latency in the query processing time.

On the whole, to the best of our knowledge, in the literature there is no efficient solution for processing top-k queries over encrypted data. In this work, we proposed such a solution.

8 Conclusion

In this work, we proposed an efficient approach, called BuckTop, for top-k query processing over encrypted data. It uses the bucketization technique to manage the encrypted data in the server. The top-k query processing of BuckTop works on the encrypted data of the buckets, and returns a set involving the top-k results. Bucktop includes also a powerful filtering algorithm that eliminates significantly the false positives from the result set, and reduces the response time and the communication cost of query processing.

We validated our approach through experimentation over synthetic and real datasets. We compared its response time over encrypted data with TA over original (plaintext) data. The experimental results show excellent performance gains for BuckTop. They illustrate that the overhead of using BuckTop for top-k processing over encrypted data is very low, because of efficient top-k processing and filtering in the server. This shows that now the top-k queries can be efficiently executed over encrypted data in untrusted remote severs.

References

[1] R. Akbarinia, E. Pacitti, and P. Valduriez. Best position algorithms for top-k queries. In *VLDB Conf.*, pages 495–506, 2007.

[2] H. Bast, D. Majumdar, R. Schenkel, M. Theobald, and G. Weikum. Io-top-k: Index-access optimized top-k query processing. In *VLDB Conf.*, pages 475–486, 2006.

[3] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 506–522, 2004.

[4] Y. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Applied Cryptography and Network Security (ACNS)*, pages 442–455, 2005.

[5] L. Chen, G. Cong, X. Cao, and K. Tan. Temporal spatial-keyword top-k publish/subscribe. In *ICDE Conf.*, pages 255–266, 2015.

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

[7] R. Churchhouse. *Codes and Ciphers.* Cambridge University Press, 2002.

[8] E. Ciceri, P. Fraternali, D. Martinenghi, and M. Tagliasacchi. Crowdsourcing for top-k query processing over uncertain data. *IEEE Trans. Knowl. Data Eng. (TKDE)*, 28(1):41–53, 2016.

[9] C. Coles and J. Yeoh. Cloud adoption practices and priorities survey report. Technical report, Cloud Security Alliance report, Jan. 2015.

[10] G. Das, D. Gunopulos, N. Koudas, and D. Tsirogiannis. Answering top-k queries using views. In *VLDB Conf.*, pages 451–462, 2006.

[11] M. Dylla, I. Miliaraki, and M. Theobald. Top-k query processing in probabilistic databases with non-materialized views. In *ICDE Conf.*, pages 122–133, 2013.

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

[13] R. Fagin. Combining fuzzy information from multiple systems. *J. Comput. Syst. Sci.*, 58(1):83–99, 1999.

[14] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS Conf.*, 2001.

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

[16] C. Gentry. Fully homomorphic encryption using ideal lattices. In *ACM Symposium on Theory of Computing (STOC)*, pages 169–178, 2009.

[17] U. Güntzer, W. Balke, and W. Kießling. Towards efficient multi-feature queries in heterogeneous environments. In *2001 International Symposium on Information Technology (ITCC)*, pages 622–628, 2001.

[18] M. Gupta, J. Gao, X. Yan, H. Cam, and J. Han. Top-k interesting subgraph discovery in information networks. In *ICDE Conf.*, pages 820–831, 2014.

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

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

[21] S. Khemmarat and L. Gao. Fast top-k path-based relevance query on massive graphs. In *ICDE Conf.*, pages 316–327, 2014.

[22] Y. Kim and K. Shim. Efficient top-k algorithms for approximate substring matching. In *SIGMOD Conf.*, pages 385–396, 2013.

[23] B. Kimelfeld and Y. Sagiv. Finding and approximating top-k answers in keyword proximity search. In *PODS Conf.*, pages 173–182, 2006.

[24] C. Li, M. Hay, G. Miklau, and Y. Wang. A data- and workload-aware query answering algorithm for range queries under differential privacy. *PVLDB*, 7(5):341–352, 2014.

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

[26] S. Michel, P. Triantafillou, and G. Weikum. KLEE: A framework for distributed top-k query algorithms. In *VLDB Conf.*, pages 637–648, 2005.

[27] S. Nepal and M. V. Ramakrishna. Query processing issues in image (multimedia) databases. In *ICDE Conf.*, pages 22–29, 1999.

[28] J. Pilourdault, V. Leroy, and S. Amer-Yahia. Distributed evaluation of top-k temporal joins. In *SIGMOD Conf.*, pages 1027–1039, 2016.

[29] S. Ranu, M. X. Hoang, and A. K. Singh. Answering top-k representative queries on graph databases. In *SIGMOD Conf.*, pages 1163–1174, 2014.

[30] B. Schneier. Description of a new variable-length key, 64-bit block cipher (blowfish). In *Fast Software Encryption workshop, Lecture Notes in Computer Science (809),*, page 191.

[31] Z. Shen, M. A. Cheema, X. Lin, W. Zhang, and H. Wang. Efficiently monitoring top-k pairs over sliding windows. In *ICDE Conf.*, pages 798–809, 2012.

[32] J. Shi, D. Wu, and N. Mamoulis. Top-k relevant semantic place retrieval on spatial RDF data. In *SIGMOD Conf.*, pages 1977–1990, 2016.

[33] A. Skovsgaard and C. S. Jensen. Finding top-k relevant groups of spatial web objects. *VLDB J.*, 24(4):537–555, 2015.

[34] M. A. Soliman, I. F. Ilyas, and K. C. Chang. Top-k query processing in uncertain databases. In *ICDE Conf.*, pages 896–905, 2007.

[35] C. Song, Z. Li, and T. Ge. Top-k oracle: A new way to present top-k tuples for uncertain data. In *ICDE Conf.*, pages 146–157, 2013.

[36] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.

[37] L. H. U, N. Mamoulis, K. Berberich, and S. J. Bedathur. Durable top-k search in document archives. In *SIGMOD Conf.*, pages 555–566, 2010.

[38] J. Wang, G. Li, D. Deng, Y. Zhang, and J. Feng. Two birds with one stone: An efficient hierarchical framework for top-k and threshold-based string similarity search. In *ICDE Conf.*, pages 519–530, 2015.

[39] X. Wang, Y. Zhang, W. Zhang, X. Lin, and Z. Huang. SKYPE: top-k spatial-keyword publish/subscribe over sliding window. *PVLDB*, 9(7):588–599, 2016.

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

[41] H. Z. Xianrui Meng and G. Kollios. Declarative cleaning of inconsistencies in information extraction. *arXiv:1510.05175v2*, 2016.

[42] H. Yang, C. Chung, and M. H. Kim. An efficient top-k query processing framework in mobile sensor networks. *Data Knowl. Eng.*, 102:78–95, 2016.

[43] B. Yao, F. Li, and X. Xiao. Secure nearest neighbor revisited. In *ICDE Conf.*, pages 733–744, 2013.

[44] X. Zhang, G. Li, and J. Feng. Crowdsourced top-k algorithms: An experimental evaluation. *PVLDB*, 9(8):612–623, 2016.

Appendix A: Correctness Proof of EncFA

Here, we prove that the set returned by EncFA contains the encrypted top-k results.

Proof. Let Y be the set of data items, which have been seen by EncFA in some lists before it stops. Let $Y' \subseteq Y$ be set of data items that have been seen in all lists. Let $d' \in Y'$ be the data item whose overall score among the data items in Y' is the minimum. In each list L_i, let s'_i be the real (plaintext) local score of d' in L_i.

We show that any data item d, which has not been seen by EncFA under sorted access, has an overall score that is less than or equal to that of d'. In each list L_i, let s_i be the plaintext local score of d in L_i. Since d has not been seen by the algorithm, and the encrypted data items in the lists are sorted according to their initial order, we have $s_i \leq s'_i$, for $1 \leq i \leq m$. Since, the scoring function f is monotonic, then we have $f(s_1, ..., s_m) \leq f(s'_1, ..., s'_m)$. Thus, the overall score of d is less than or equal to that of d'. Therefore, the set Y contains at least k data items whose overall scores are greater than or equal to that of the unseen data d. □

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399