# Answering Top-k Queries over Outsourced Sensitive Data in the Cloud

Sakina Mahboubi, Reza Akbarinia, Patrick Valduriez

# Answering Top-k Queries over Outsourced Sensitive Data in the Cloud

Sakina Mahboubi, Reza Akbarinia, and Patrick Valduriez

INRIA & LIRMM, University of Montpellier, France
FirstName.LastName@inria.fr

**Abstract.** The cloud 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 is to encrypt it before sending to the cloud. Then, the main problem is to evaluate user queries over the encrypted data.

In this paper, we consider the problem of answering top-k queries over encrypted data. We propose a novel system, called BuckTop, designed to encrypt and outsource the user sensitive data to the cloud. BuckTop comes with a top-k query processing algorithm that is able to process efficiently top-k queries over the encrypted data, without decrypting the data in the cloud data centers.

We implemented BuckTop and compared its performance for processing top-k queries over encrypted data with that of the popular threshold algorithm (TA) over original (plaintext) data. The results show the effectiveness of BuckTop for outsourcing sensitive data in the cloud and answering top-k queries.

**Keywords:** Cloud, Sensitive Data, Top-k Query

## 1 Introduction

The cloud allows users and companies to efficiently store and process their data in third-party data centers. However, users typically loose physical access control to their data. Thus, potentially sensitive data gets at risk of security attacks, *e.g.*, from employees of the cloud provider. According to a recent report published by the Cloud Security Alliance [4], security attacks are one of the main concerns for cloud users.

One solution for protecting user sensitive data is to encrypt it before sending to the cloud. 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 queries over *plaintext (non encrypted)* data. This solution is inefficient, because it does not take advantage of the cloud computing power for evaluating queries.

In this paper, we are interested in processing top-k queries over encrypted data in the cloud. 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*.

Top-k query processing over encrypted data is critical for many applications that outsource sensitive data. For example, consider a university that outsources the students database in a public cloud, with non-trusted nodes. The database is encrypted for privacy reasons. Then, an interesting top-k query over the outsourced encrypted data is the following: return the $k$ students that have the worst averages in some given courses.

There are many different approaches for processing top-k queries over plaintext data. One of the best known approaches is TA (threshold algorithm) [8] 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 its extensions assume that the attribute values are available as plaintext, and not encrypted.

In this paper, we address the problem of privacy preserving top-k query processing in clouds. We first propose a basic approach, called OPE-based, that uses a combination of the order preserving encryption (OPE) and the FA algorithm for privacy preserving top-k query processing.

Then, we propose a complete system, called *BuckTop*, that is able to efficiently evaluate top-k queries over encrypted data, without decrypting them in the cloud. BuckTop includes a top-k query processing algorithm that works on the encrypted data, and returns a set that is proved to contain the encrypted data corresponding to the top-k results. It also comes with an efficient filtering algorithm that is executed in the cloud and removes most of the false positives included in the set returned by the top-k query processing algorithm. This filtering is done without needing to decrypt the data in the cloud.

We implemented BuckTop, and compared its response time over encrypted data with a baseline algorithm and 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 cloud by BuckTop's filtering algorithm.

The rest of this paper is organized as follows. Section 2 gives the problem definition. Section 3 presents our basic approach for privacy preserving top-k query processing. Section 4 describes our BuckTop system and its algorithms. Section 5 reports performance evaluation results. Section 6 discusses related work, and Section 7 concludes.

## 2 Problem Definition

In this paper, we address the problem of processing top-k queries over encrypted data in the cloud.

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*) [8]. 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. Like many previous works on top-k query processing (*e.g.*, [8]), we assume that the scoring function is monotonic.

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 such 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 [8]. The first is *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 *random access* by which we look up a given data item in the list.

In this paper, we consider the honest-but-curious adversary model for the cloud. 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 solutions proposed for secure processing of the different queries [13].

Let us now formally state the problem which we address. 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. *We assume that the database $E(D)$ is stored in one node of the cloud.*

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.

## 3 OPE-based Top-k Query Processing Approach

In this section, we propose an approach, called OPE-based, that uses a combination of the order preserving encryption (OPE) [1] and the FA algorithm [7] for privacy preserving top-k query processing. Our main contribution, called BuckTop, is presented in the next section.

Let us first explain how the local scores are encrypted. With the OPE-based approach, the local scores (attribute values) in the sorted lists are encrypted using the order preserving encryption technique. We also use a *deterministic* encryption method for encrypting the ID of data items. The *deterministic* encryption generates the same ciphertexts for two equal inputs. This allows us to do random access to the encrypted sorted lists by using the ID of data items.

After encrypting the data IDs and local scores in each sorted list, the lists are sent to the cloud.

Let us now describe how top-k queries can be answered in the cloud over the encrypted data. Given a top-k query $Q$ with a scoring function $f$, the query is sent to the cloud. Then, the cloud uses the FA algorithm for processing $Q$ as follows. It continuously performs sorted access in parallel to each sorted list, and maintains the encrypted data IDs and their encrypted local scores in a set $Y$. When there are at least $k$ encrypted data IDs in $Y$ such that each of them has been seen in each of the lists, then the cloud stops doing sorted access to the lists. Then, for each data item $d$ involved in $Y$, and each list $L_i$, the cloud performs random access to $L_i$ to find the encrypted local scores of $d$ in $L_i$ (if it has not been seen yet). The cloud sends $Y$ to the user machine which decrypts the local scores of each item $d \in Y$, computes their overall scores, and find the final $k$ items with the highest overall scores.

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

**Proof.** Let $Y$ be the set of data items, which have been seen by top-k query processing algorithm 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 the algorithm 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 top-k query processing 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 $d$. □

## 4 BuckTop System

In this section, we present our BuckTop system. We first describe the architecture of BuckTop, and introduce our method for encrypting the data items and storing them in the cloud. Afterwards, we propose an algorithm for processing top-k queries over encrypted data, and an algorithm for filtering the false positives in the cloud.

### 4.1 System Architecture and Data Encryption

The architecture of BuckTop 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.
  For example, suppose we have a relation $R$ with attributes $att_1$, $att_2$,, $att_m$, and the user issues the following query:
  *SELECT * FROM R ORDERED BY* $f(att_1, \ldots, att_m)$ *LIMIT k*;
  This query is transformed to:
  *SELECT * FROM E(R) ORDERED BY* $F(E(att_1),\ldots, E(att_m))$ *LIMIT k*;
  where $E(R)$ and $E(att_i)$ are the encrypted name of the relation $R$ and the attribute $att_i$ respectively.
  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.
- **Service provider.** It is installed in the cloud, 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 cloud.

Let us now present our approach for encrypting and outsourcing the data to the cloud. As mentioned before, the trusted client component of BuckTop is responsible for encrypting the user databases. Before encrypting a database, the trusted client 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 [9]. 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 itme 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 the cloud. The buckets are stored in the cloud 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 cloud to know the order of data items inside the buckets.

### 4.2 Top-k Query Processing Algorithm of BuckTop

The main idea behind top-k query processing in BuckTop system is to use the bucket boundaries to decide when to stop reading the encrypted data from the lists.

Given a top-k query $Q$ including a number $k$ and a scoring function $f$. To answer $Q$, the following top-k processing algorithm is executed by the service provider component of BuckTop:

1. Let $Y$ be an empty set;
2. Perform sorted access to the lists:
    - 2.1 Read the next bucket, say $b_i$, from each list $L_i$ (starting from the head of the list);
    - 2.2 For each encrypted data $d$ contained in the bucket $b_i$:
        - 2.2.1. Perform random access in parallel to the other lists to find the encrypted score and the bucket of $d$ in all lists;
        - 2.2.2. Compute a minimum overall score for $d$, denoted by $min\_ovl(d)$, by applying the scoring function on the lower bound of the buckets that contain $d$ in different lists. Formally, $min\_ovl(d) = f(min(b_1)$, $min(b_2)$, ..., $min(b_m))$, where $b_i$ is the bucket involving $d$ in the list $L_i$.
        - 2.2.3. Store the encrypted ID of $d$, its encrypted local scores, and its $min\_ovl$ score in the set $Y$.
    - 2.3 Compute a threshold $TH$ as follows: $TH = f(min(b'_1)$, $min(b'_2)$, ... , $min(b'_m))$, where $b'_i$ is the last bucket seen under sorted access in the $L_i$, for $1 < i < m$. In other words, $TH$ is computed by applying the scoring function on the lower bounds of the last seen buckets in the lists.

2.4 If the set $Y$ contains at least $k$ encrypted data items having minimum overall scores higher than *TH*, then stop. Otherwise, go to Step 2.1.

When the top-k query processing algorithm stops, the set $Y$ includes the encrypted top-k data items (see the proof below). This set is sent to the trusted client that decrypts its contained data items, 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.

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

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

*Proof.* Let $Y$ be the 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 $Y$. 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 *TH.* When the algorithm stops, there are at least $k$ data items in $Y$ whose minimum overall scores are greater than or equal to *TH.* Thus, their overall scores are at least *TH.* Therefore, their overall scores are greater than or equal to that of the data item $d$.

In the set $Y$ returned by the top-k query processing algorithm of BuckTop, 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 cloud, without decrypting the data items. As shown by our experimental results, our filtering algorithm eliminates most of the false positives (more than 99% in the different tested datasets). This improves significantly the response time of top-k queries, because the eliminated false positives do not need to be communicated to the trusted client and should not be decrypted by it.

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 on the upper bound of the buckets involving the data item in the lists. The algorithm proceeds as follows:

1. Let $Y' \subseteq Y$ be the $k$ data items in $Y$ that have the highest minimum overall scores (*min_ovl*) among the items contained in $Y$.

2. Let $d_{min}$ be the data item that has the lowest *min_ovl* score in $Y'$.
3. For each item $d \in Y$
   - 3.1 Compute the maximum overall score of $d$, *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, $max\_ovl(d) = f(max(b_1), max(b_2), ..., max(b_m))$.
   - 3.2 If the maximum overall score of $d$ is less than or equal to the minimum overall score of $d_{min}$, then remove $d$ from $Y$. In other words, if $max\_ovl(d) \leq min\_ovl(d_{min}) \Rightarrow Y = Y - \{d\}$

Let us prove that the filtering algorithm 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 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 $d$. □

A security analysis of the BuckTop system is provided in [15].

## 5 Performance Evaluation

In this section, we evaluate the performance of BuckTop using synthetic and real datasets. We first describe the experimental setup, and then report the results of our experiments.

### 5.1 Experimental Setup

We implemented our top-k query processing system and performed our tests on real and synthetic datasets. As in some previous work on encrypted data (*e.g.*, [13]), 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 [13], 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.

We compare our solution with the two following approaches:

- *OPE*: this is the OPE-based solution (presented in Section 3) that uses the order preserving encryption for encrypting the data scores.
- *TA over plaintext data*: the objective is to show the overhead of top-k query processing by BuckTop over encrypted data compared to an efficient top-k algorithm over plaintext data.

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.

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.

In the experiments, we measure the following metrics:

- **Cloud top-k time:** the time required by the service provider of BuckTop in the cloud to find the set that includes the top-k results, *i.e.*, the set $Y$.
- **Response time:** the total time elapsed between the time when the query is sent to the cloud and the time when the $k$ decrypted results are returned to the user. This time includes the cloud 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 cloud.

We performed our experiments using a node with 16 GB of main memory and Intel Core i7-5500 @ 2.40Ghz as processor.

### 5.2 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.*, $n$.

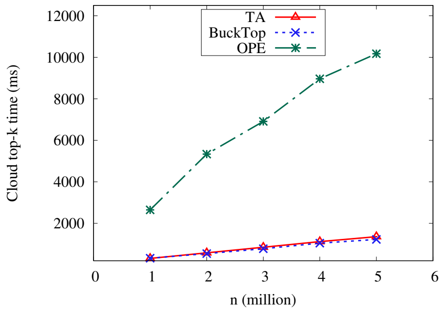

Fig. 1: Cloud top-k time vs. number of database tuples

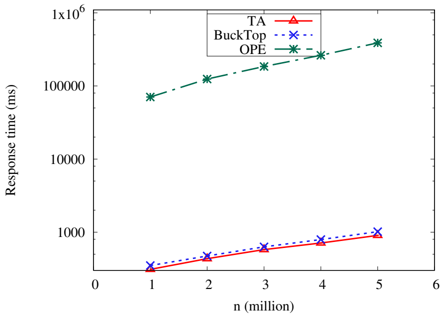

Fig. 2: Response time vs. number of database tuples

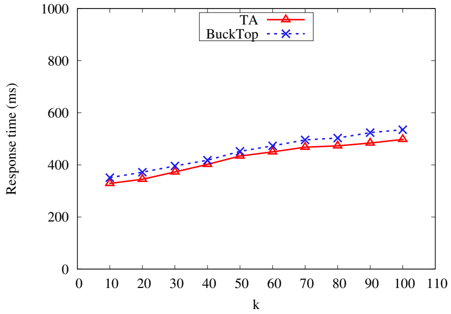

Fig. 3: Response time vs. k

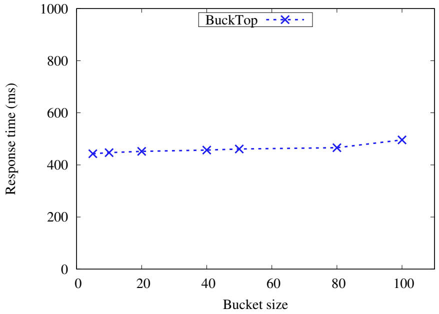

Fig. 4: Response time vs. bucket size

Figure 1 shows how cloud top-k time evolves, with increasing $n$, and the other parameters set as default values described in Section 5.1. The cloud 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 2 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 cloud time. The response time of BuckTop is slightly higher than its cloud top-k time, as in addition to top-k query processing it performs the filtering in the cloud and also needs to decrypt at least $k$ data items. We see that the response time of OPE is much higher than its cloud 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 cloud (see the results in Section 5.5), thus there is no need to decrypt them.

| Database size (M) | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 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 |
|---|---|---|---|---|---|---|
| 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 |
|---|---|---|---|---|---|---|
| Rate of eliminated false positives | 99.94% | 99.96% | 99.97% | 99.98% | 99.98% | 99.98% |

C: over Gaussian dataset

Table 1: False positive elimination by our filtering algorithm over different datasets

### 5.3 Effect of *k*

Figure 3 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).

### 5.4 Effect of Bucket Size

Figure 4 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.

### 5.5 Effect of the Filtering Algorithm

BuckTop's filtering algorithm is used to eliminate/reduce the false positives in the cloud. We study the filtering rate by increasing the size of the dataset. For the uniform synthetic dataset, the results are shown in Table 1-*A*. For datasets with up to three million data items, the filtering method eliminates 100% of the false positives, and the cloud returns to the trusted client only the $k$ data items that are the result of the query. For larger datasets, BuckTop filters up to 99,99% of the false positives. By using the Gaussian dataset, we obtain the results shown in Table 1-*C*. We see that around 99,94% of false positives are eliminated.

Over the real dataset, Table 1-*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.

## 6 Related Work

In the literature, there has been some research work to process keyword queries over encrypted data , *e.g.*, [2, 17]. For example [2] and [17] 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.*, [3,5,6,14,19]. 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.*, [9, 10, 16]. For example, in [10], 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.

There have been access pattern attacks against range query processing methods that use the bucketization technique, e.g. [11]. The main idea is to utilize the intersection between the results of the queries and also some background knowledge to guess the bucket boundaries. However, these attacks are not valid for our approach, because there is no range in our queries. In our system, the main plaintext information in the queries is $k$ (*i.e.*, the number of asked top tuples), and this information is not usually useful to violate the privacy of users.

In [12], Kim et al. propose an approach for preserving the privacy of data access patterns during top-k query processing. In [18], 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 encrypted), but in our system we trust no node of the cloud.

Meng et al [20] propose a solution for processing top-k queries over encrypted data. They assume the existence of two non-colluding nodes in the cloud, one of which can decrypt the data (using the decryption key) and execute a TA-based algorithm. Our assumptions about the cloud are different, as we do not trust any node of the cloud.

## 7 Conclusion

In this paper, we proposed a novel system, called BuckTop, designed to encrypt sensitive data items, outsource them to a non-trusted cloud, and answer top-k queries. BuckTop has a top-k query processing algorithm that is executed over encrypted data, and returns a set containing the top-k results, without decrypting the data in the cloud. It also comes with a powerful filtering algorithm that eliminates significantly the false positives from the result set.

We validated our system through experimentation over synthetic and real datasets. We compared its response time with OPE over encrypted data, and with the popular TA algorithm 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 false positive filtering.

## 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.

## References

1. Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order-preserving encryption for numeric data. In *SIGMOD Conf.*, pages 563–574, 2004.
2. Y. 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*, pages 2818–2831, 2014.
4. C. Coles and J. Yeoh. Cloud adoption practices and priorities survey report. Technical report, Cloud Security Alliance report, January 2015.
5. X. Ding, P. Liu, and H. Jin. Privacy-preserving multi-keyword top-k similarity search over encrypted data. *IEEE TDSC*, (99):1–14, 2017.
6. Y. Elmehdwi, BK. Samanthula, and W. Jiang. Secure k-nearest neighbor query over encrypted data in outsourced environments. In *ICDE Conf.*, 2014.
7. Ronald Fagin. Combining fuzzy information from multiple systems. *J. Comput. Syst. Sci.*, 58(1):83–99, 1999.
8. Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.
9. B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu. Secure multidimensional range queries over outsourced data. *VLDB J.*, 21(3):333–358, 2012.
10. B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In *VLDB Conf.*, pages 720–731, 2004.
11. MS. Islam, M. Kuzu, and M. Kantarcioglu. Inference attack against encrypted range queries on outsourced databases. In *ACM CODASPY*, pages 235–246, 2014.
12. H-I. Kim, H-J. Kim, and J-W. Chang. A privacy-preserving top-k query processing algorithm in the cloud computing. In *GECON Conf.*, pages 277–292, 2017.

13. 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), 2014.
14. 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.
15. S. Mahboubi, R. Akbarinia, and P. Valduriez. Top-k Query Processing Over Outsourced Encrypted Data. Research Report RR-9053, INRIA, 2017.
16. C. Sahin, T. Allard, R. Akbarinia, A. El Abbadi, and E. Pacitti. A differentially private index for range query processing in clouds. In *ICDE Conf.*, 2018.
17. DX. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE S&P*, pages 44–55, 2000.
18. Jaideep Vaidya and Chris Clifton. Privacy-preserving top-k queries. In *ICDE Conf.*, pages 545–546, 2005.
19. WK. Wong, DW. Cheung, B. Kao, and N. Mamoulis. Secure knn computation on encrypted databases. In *SIGMOD Conf*, pages 139–152, 2009.
20. Haohan Zhu Xianrui Meng and George Kollios. Top-k query processing on encrypted databases with strong security guarantees. In *ICDE Conf.*, 2018.