Towards Differential Query Services in Cost-Efficient Clouds

Qin Liu, Chiu C. Tan, *Member, IEEE*, Jie Wu, *Fellow, IEEE*, and Guojun Wang, *Member, IEEE*

Abstract—Cloud computing as an emerging technology trend is expected to reshape the advances in information technology. In a cost-efficient cloud environment, a user can tolerate a certain degree of delay while retrieving information from the cloud to reduce costs. In this paper, we address two fundamental issues in such an environment: privacy and efficiency. We first review a private keyword-based file retrieval scheme that was originally proposed by Ostrovsky. Their scheme allows a user to retrieve files of interest from an untrusted server without leaking any information. The main drawback is that it will cause a heavy querying overhead incurred on the cloud and thus goes against the original intention of cost efficiency. In this paper, we present three efficient information retrieval for ranked query (EIRQ) schemes to reduce querying overhead incurred on the cloud. In EIRQ, queries are classified into multiple ranks, where a higher ranked query can retrieve a higher percentage of matched files. A user can retrieve files on demand by choosing queries of different ranks. This feature is useful when there are a large number of matched files, but the user only needs a small subset of them. Under different parameter settings, extensive evaluations have been conducted on both analytical models and on a real cloud environment, in order to examine the effectiveness of our schemes.

Index Terms—Cloud computing, cost efficiency, differential query services, privacy

1 Introduction

Cloud computing as an emerging technology is expected to reshape information technology processes in the near future [1]. Due to the overwhelming merits of cloud computing, e.g., cost-effectiveness, flexibility and scalability, more and more organizations choose to outsource their data for sharing in the cloud. As a typical cloud application, an organization subscribes the cloud services and authorizes its staff to share files in the cloud. Each file is described by a set of keywords, and the staff, as authorized users, can retrieve files of their interests by querying the cloud with certain keywords. In such an environment, how to protect *user privacy* from the cloud, which is a third party outside the security boundary of the organization, becomes a key problem.

User privacy can be classified into *search privacy* and *access privacy* [2]. Search privacy means that the cloud knows nothing about what the user is searching for, and access privacy means that the cloud knows nothing about which files are returned to the user. When the files are stored in the clear forms, a naïve solution to protect user privacy is for the user to request *all* of the files from the cloud; this way, the cloud cannot know which files the user is really interested in. While this does provide the necessary privacy, the communication cost is high.

Private searching was proposed by Ostrovsky *et al.* [3], [4] (referred to as the Ostrovsky scheme in this paper), which allows a user to retrieve files of interest from an untrusted server without leaking any information. However, the Ostrovsky scheme has a high computational cost, since it requires the cloud to process the query (perform homomorphic encryption) on *every* file in a collection. Otherwise, the cloud will learn that certain files, without processing, are of no interest to the user. It will quickly become a performance bottleneck when the cloud needs to process thousands of queries over a collection of hundreds of thousands of files. We argue that subsequently proposed improvements, like [5], [6], also have the same drawback. Commercial clouds follow a *pay-as-you-go* model, where the customer is billed for different operations such as bandwidth, CPU time, and so on. Solutions that incur excessive computation and communication costs are unacceptable to customers.

To make private searching applicable in a cloud environment, our previous work [7] designed a cooperate private searching protocol (COPS), where a proxy server, called the aggregation and distribution layer (ADL), is introduced between the users and the cloud. The ADL deployed inside an organization has two main functionalities: aggregating user queries and distributing search results. Under the ADL, the computation cost incurred on the cloud can be largely reduced, since the cloud only needs to execute a combined query *once*, no matter how many users are executing queries. Furthermore, the communication cost incurred on the cloud will also be reduced, since files shared by the users need to be returned only once. Most importantly, by using a series of secure functions, COPS can protect user privacy from the ADL, the cloud, and other users.

- *Q. Liu is with the School of Information Science and Engineering, Central South University, Changsha, Hunan Province, 410083, China, and also with the College of Information Science and Engineering, Hunan University, Changsha, Hunan Province, 410082, China. E-mail: gracelq628@hnu.edu.cn.*
- *C.C. Tan and J. Wu are with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122 USA. E-mail: {cctan, jiewu}@temple.edu.*
- *G. Wang is with the School of Information Science and Engineering, Central South University, Changsha, Hunan Province, 410083, China. E-mail: csgjwang@mail.csu.edu.cn.*

Manuscript received 18 Jan. 2013; revised 22 Mar. 2013; accepted 3 May 2013. Date of publication 12 May 2013; date of current version 16 May 2014. Recommended for acceptance by A. Nayak.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2013.132

In this paper, we introduce a novel concept, *differential query services*, to COPS, where the users are allowed to personally decide how many matched files will be returned. This is motivated by the fact that under certain cases, there are a lot of files matching a user's query, but the user is interested in only a certain percentage of matched files. To illustrate, let us assume that Alice wants to retrieve 2 percent of the files that contain keywords ''A, B'', and Bob wants to retrieve 20 percent of the files that contain keywords ''A, C''. The cloud holds 1,000 files, where $\{F_1, \ldots, F_{500}\}$ and $\{F_{501}, \ldots, F_{1000}\}$ are described by keywords ''A, B'' and ''A, C'', respectively. In the Ostrovsky scheme, the cloud will have to return 2,000 files. In the COPS scheme, the cloud will have to return 1,000 files. In our scheme, the cloud only needs to return 200 files. Therefore, by allowing the users to retrieve matched files on demand, the bandwidth consumed in the cloud can be largely reduced.

Motivated by this goal, we propose a scheme, termed Efficient Information retrieval for Ranked Query (EIRQ), in which each user can choose the rank of his query to determine the percentage of matched files to be returned. The basic idea of EIRQ is to construct a privacy-preserving *mask matrix* that allows the cloud to filter out a certain percentage of matched files before returning to the ADL. This is not a trivial work, since the cloud needs to correctly filter out files according to the rank of queries without knowing anything about user privacy. Focusing on different design goals, we provide two extensions: the first extension emphasizes *simplicity* by requiring the least amount of modifications from the Ostrovsky scheme, and the second extension emphasizes *privacy* by leaking the least amount of information to the cloud.

Our key contributions are as follows:

1. We propose three EIRQ schemes based on the ADL to provide a cost-efficient solution for private searching in cloud computing.
2. The EIRQ schemes can protect user privacy while providing a differential query service that allows each user to retrieve matched files on demand.
3. We provide two solutions to adjust related parameters; one is based on the Ostrovsky scheme, and the other is based on Bloom filters.
4. Extensive experiments were performed using a combination of simulations and real cloud deployments to validate our schemes.

The remainder of this paper is organized as follows. We introduce related work in Section 2 before presenting preliminaries in Section 3. We describe EIRQ schemes in Section 4 and adjust the parameters in Section 5. After analyzing the performance and security of the proposed schemes in Section 6, we conduct evaluations in Section 7. Finally, we conclude this paper in Section 8.

2 Related Work

Our work aims to provide differential query services while protecting user privacy from the cloud. Existing research that is similar to ours can be found in the areas of private searching [3], [4], [5], [6], [7], [8], [9], [10], [11].

Unlike searchable encryption [2], [12], where the user conducts searches on encrypted data, private searching performs keyword-based searches on unencrypted data. Private searching was first proposed in [3], [4], which allows a server to filter streaming data without compromising user privacy. Their solution requires the server to return a buffer of size $O(f \log(f))$ when f files match a user's query. Each file is associated with a *survival rate*, which denotes the probability of this file being successfully recovered by the user. Based on the Paillier cryptosystem [13], the files that mismatch a query will not survive in the buffer, but the matched files enjoy a high survival rate.

Among various extensions, [5], [6] further reduced the communication cost from $O(f \log(f))$ to $O(f)$ by solving a set of linear equations to recover f matched files. However, their scheme requires the decryption of one more buffer, thus the computation cost is higher than the Ostrovsky scheme. Reference [8] presented an efficient decoding mechanism which allows the recovery of files that collide in a buffer position. Reference [9] proposed a recursive extraction mechanism, which requires a buffer of size $O(f)$ when f files match a user's query. Reference [10] proposed two new communication-optimal constructions; one uses Reed-Solomon codes and allows for a zero-error, and the other is based on irregular LDPC codes and allows for lower computation cost at the server. The above private searching schemes only support searching for OR of keywords or AND of two sets of keywords. Reference [11] extended the types of queries to support disjunctive normal forms (DNF) of keywords. The main drawback of existing private searching schemes is that both the computation and communication costs grow linearly with the number of users executing queries. Thus, when applying these schemes to a large-scale cloud environment, querying costs will be extensive.

Our previous work [7] was the first to make private searching techniques applicable to a cloud environment. However, [7] requires the cloud to return all of the matched files, which may cause a waste of bandwidth when only a small percentage of files are of interest. To alleviate the problem, we introduced the concept of differential query services in [14]. The main difference between this work and [14] is that we provide two extensions to address different aspects of the problem, and we conduct extensive experiments on a real cloud to verify the effectiveness of the proposed schemes.

3 Background

3.1 System Model

The system mainly consists of three entities:[1] the aggregation and distribution layer (ADL), many users, and the cloud, as shown in Fig. 1. For ease of explanation, we only use a single ADL in this paper, but multiple ADLs can be deployed as necessary. An ADL is deployed in an organization that authorizes its staff to share data in the cloud. The staff members, as the authorized users, send their queries to the ADL, which will aggregate user queries and send a combined

[1] The users inside an organization share data in the cloud. Thus, we assume that a management server maintained by the organization is in charge of managing the authorized users and related keys. Limited by the space, we do not detail this entity here.

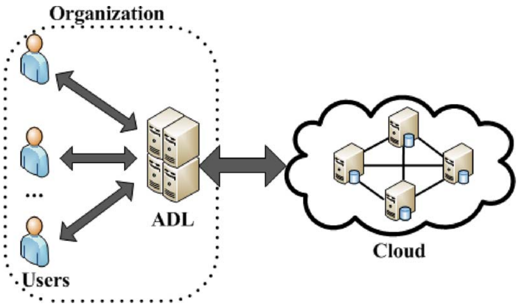

Fig. 1. System model.

query to the cloud. Then, the cloud processes the combined query on the file collection and returns a buffer that contains all of matched files to the ADL, which will distribute the search results to each user. To aggregate sufficient queries, the organization may require the ADL to wait for a period of time before running our schemes, which may incur a certain querying delay. In the supplementary file which is available in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/132, we will discuss the computation and communication costs as well as the querying delay incurred on the ADL.

To further reduce the communication cost, a differential query service is provided by allowing each user to retrieve matched files on demand. Specifically, a user selects a particular *rank* for his query to determine the percentage of matched files to be returned. This feature is useful when there are a lot of files that match a user's query, but the user only needs a small subset of them.

3.2 Security Model and Design Goals

The ADL is deployed inside the security boundary of an organization, and thus it is assumed to be *trusted* by all of the users. In the supplementary file available online, we will discuss how the EIRQ schemes work without such an assumption. The communication channels are assumed to be secured under existing security protocols, such as SSL, during information transfer. With these assumptions, as long as the ADL obeys our schemes, a user cannot know anything about other users' interests, and thus the cloud is the only attacker in our security model. As in existing work [15], [16], the cloud is assumed to be *honest but curious*. That is, it will obey our schemes, but still wants to know some additional information about user privacy.

Reference [2] classified user privacy into *search privacy* and *access privacy*. In our work, user queries are classified into multiple ranks, and thus a new kind of user privacy, *rank privacy*, also needs to be protected against the cloud. Rank privacy entails hiding the rank of each user query from the cloud, i.e., the cloud provides differential query services without knowing which level of service is chosen by the user. Rank privacy can be classified into *basic level* and *high level*, where basic level will hide the rank of each query from the cloud, and the high level will further hide the number of ranks from the cloud. Our design goal can be subdivided as follows:

- **Cost efficiency**. The users can retrieve matched files on demand to further reduce the communication costs incurred on the cloud.
- **User privacy**. The cloud cannot know anything about the user's search privacy, access privacy, and at least the basic level of rank privacy.

3.3 Overview of the Ostrovsky Scheme

We briefly introduce the Ostrovsky scheme [3], [4], which relies on a public key cryptosystem, the *Paillier cryptosystem* [13]. Let $E_{pk}(m)$ denote the encryption of plaintext m under public key pk. The Paillier cryptosystem has the following homomorphic properties:

- $E_{pk}(a) \cdot E_{pk}(b) = E_{pk}(a+b)$
- $E_{pk}(a)^b = E_{pk}(a \cdot b)$.

The Paillier cryptosystem allows the performance of certain operations, such as *multiplication* and *exponentiation*, on ciphertext directly. Given the resultant ciphertext, the user can obtain the corresponding plaintext that processes *addition* and *multiplication* operations.

The Ostrovsky scheme consists of three algorithms, the working process of which is shown in Fig. 2a. Two assumptions are used in their scheme: first, a dictionary that consists of the universal keywords is assumed to be publicly available; second, the users are assumed to have the ability to estimate the number of files that match their queries. To better illustrate its working process, we provide an example in the supplementary file available online.

Step 1. The user runs the *GenerateQuery* algorithm to send an encrypted query to the cloud. The query is a bit string encrypted under the user's public key, where each bit is an encryption of 1, if the keyword in the dictionary is chosen; otherwise, it is an encryption of 0.

Step 2. The cloud runs the *PrivateSearch* algorithm to return an encrypted buffer to the user. Generally speaking, the cloud processes the encrypted query on every file in the collection to generate an encrypted c-e pair, and maps it to multiple entries of an encrypted buffer. For file F_j, the corresponding c-e pair, denoted as (c_j, e_j), is generated as follows: the bits in query Q corresponding to keywords in F_j are multiplied

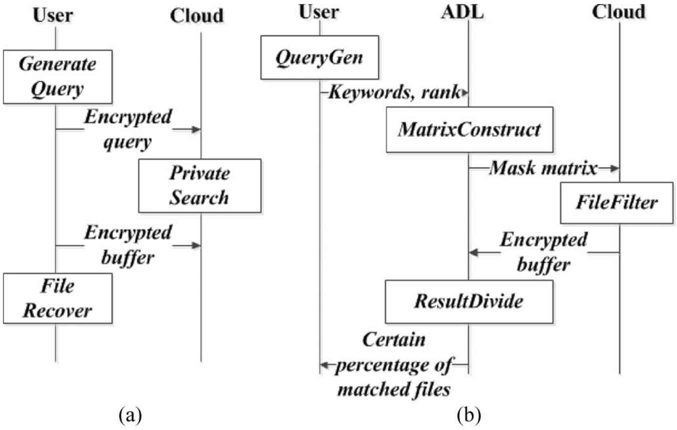

Fig. 2. Working process. (a) Ostrovsky scheme. (b) EIRQ-Efficient scheme.

together to form $c_j = \prod_{Dic[i] \in F_j} Q[i]$, where $Dic[i]$ denotes the i-th keyword in the dictionary, and file content $|F_j|$ is powered to c_j to form $e_j = c_j^{|F_j|}$.

Then, the cloud constructs a buffer of size β. Let $\mathbb{B}$ denote the buffer, where the i-th entry, denoted as $\mathbb{B}[i]$, consists of two parts, denoted as $\mathbb{B}[i,1]$ and $\mathbb{B}[i,2]$, both of which are initialized with an encryption of 0 under the user's public key. To map (c_j, e_j) to the buffer, the cloud randomly chooses an entry, say p_*, and multiplies (c_j, e_j) to this entry by performing $\mathbb{B}[p_*,1] = \mathbb{B}[p_*,1] \cdot c_j$ and $\mathbb{B}[p_*,2] = \mathbb{B}[p_*,2] \cdot e_j$. The mapping operation will be performed γ times. After mapping all pairs to the buffer, each buffer entry has one of the three statuses: *survival*, *collision*, and *mismatch*. If only one matched file is mapped, the entry state is survival; if more than one matched file is mapped, the entry state is collision; if no matched files are mapped, the entry state is mismatch.

Step 3. The user runs the *FileRecover* algorithm to recover files. The user decrypts the buffer, entry by entry, to obtain the plaintext c-e pairs. For the entries in the survival state, file content can be recovered by dividing the plaintext e value by the plaintext c value.

The security of the Ostrovsky scheme derives from the *semantic security* of the Paillier cryptosystem. The key technique of their scheme is that the files mismatching a user's query are processed to encrypted 0s, which have no impact on the matched files, even if they are mapped in the same entry. Thus, the buffer size only depends on the number of matched files, which is much smaller than the number of files stored in the cloud.

4 Scheme Description

In this section, we will describe the original EIQR scheme and its two extensions. To distinguish the three EIRQ schemes, we name the original EIRQ scheme as EIRQ-Efficient, the first extension as EIRQ-Simple, and the second extension as EIRQ-Privacy, in this paper.

The basic idea of EIQR-Efficient is to construct a privacy-preserving *mask matrix* with which the cloud can filter out a certain percentage of matched files before mapping them to a buffer. As proven in the Ostrovsky scheme, the file survival rate is determined by the buffer size β and mapping times γ. Therefore, the basic idea of two extensions is that, for each rank $i \in \{0, \ldots, r\}$, the ADL adjusts the buffer size β_i and the mapping times γ_i to make the file survival rate q_i approach $1 - i/r$. To better illustrate the working process of the EIRQ schemes, we provide examples in the supplementary file available online.

4.1 The EIRQ-Efficient Scheme

Before illustrating EIQR-Efficient, two fundamental problems should be resolved:

Firstly, we should determine the relationship between query rank and the percentage of matched files to be returned. Suppose that queries are classified into $0 \sim r$ ranks. Rank-0 queries have the highest rank and Rank-r queries have the lowest rank. In this paper, we simply determine this relationship by allowing Rank-i queries to retrieve $(1 - i/r)$ percent of matched files. Therefore, Rank-0 queries can retrieve 100 percent of matched files, and Rank-r queries cannot retrieve any files.

Secondly, we should determine which matched files will be returned and which will not. In this paper, we simply determine the probability of a file being returned by the highest rank of queries matching this file. Specifically, we first rank each keyword by the highest rank of queries choosing it, and then rank each file by the highest rank of its keywords. If the file rank is i, then the probability of being filtered out is i/r. Therefore, Rank-0 files will be mapped into a buffer with probability 1, and Rank-r files will not be mapped at all. Since unneeded files have been filtered out before mapping, the mapped files should survive in the buffer with probability 1. In Section 5, we will illustrate how to adjust the buffer size and mapping times to achieve this goal.

EIRQ-Efficient mainly consists of four algorithms, with its working process being shown in Fig. 2b. Since algorithms *QueryGen* and *ResultDivide* are easily understood, we only provide the details of algorithms *MatrixConstruct* and *FileFilter* in Alg. 1.

Algorithm 1 The EIRQ-Efficient scheme.

```
MatrixConstruct (run by the ADL with public key pk)
for $i = 1$ to $d$ do
  set $l$ to be the highest rank of queries choosing $Dic[i]$
  for $j = 1$ to $r$ do
    if $j \leq r - l$ then
      $M[i,j] = E_{pk}(1)$
    else
      $M[i,j] = E_{pk}(0)$
adjust $\gamma$ and $\beta$ so that file survival rate is 1
FileFilter (run by the cloud)
for each file $F_j$ stored in the cloud do
  for $i = 1$ to $d$ do
    $k = j \bmod r$; $c_j = \prod_{Dic[i] \in F_j} M[i,k]$; $e_j = c_j^{|F_j|}$
    map $(c_j, e_j)$ $\gamma$ times to a buffer of size $\beta$
```

Step 1. The user runs the *QueryGen* algorithm to send keywords and the rank of the query to the ADL. Since the ADL is assumed to be a trusted third party, this query will be sent without encryption.

Step 2. After aggregating enough user queries, the ADL runs the *MatrixConstruct* algorithm to send a *mask matrix* to the cloud. The mask matrix M is a d-row and r-column matrix, where d is the number of keywords in the dictionary, and r is the lowest query rank. Let $M[i,j]$ denote the element in the i-th row and the j-th column, and let l be the highest rank of queries that choose the i-th keyword $Dic[i]$ in the dictionary. M is constructed as follows: for the i-th row of M that corresponds to $Dic[i]$,

$M[i,1],\ldots,M[i,r-l]$ are set to 1, and $M[i,r-l+1],\ldots,M[i,r]$ are set to 0, then each element is encrypted under the ADL's public key pk. For the rows that correspond to Rank-l keywords, the ADL sets the *first* $r-l$ elements, rather than random $r-l$ elements, to 1. The reason is to ensure that, given any Rank-l file F_j, when we choose a random number k, the probability of all of the k-th elements of the rows that correspond F_j's keywords being 0 is l/r, which is determined by the highest rank of F_j's keywords.

Step 3. The cloud runs the *FileFilter* algorithm to return a buffer that contains a certain percentage of matched files to the ADL. Specifically, the cloud multiplies the k-th elements of the rows that correspond to F_j's keywords together to form c_j, where $k = j \bmod r$. Then, it powers $|F_j|$ to c_j to obtain e_j, and maps the c-e pair into multiple entries of a buffer, as in the Ostrovsky scheme. Note that, with Step 2, we can make sure that, for a Rank-l file F_j, the probability of c_j being 0 is l/r, and thus the probability of F_j being filtered out is l/r.

Step 4. The ADL runs the *ResultDivide* algorithm to distribute search results to each user. File contents are recovered as the *FileRecover* algorithm in the Ostrovsky scheme. To allow the ADL to distribute files correctly, we require the cloud to attach keywords to the file content. Thus, the ADL can find out all of the files that match users' queries by executing keyword searches.

4.2 The EIRQ-Simple Scheme

The working process of EIRQ-Simple is similar to Fig. 2b. The main differences lie in the *MatrixConstruct* and *FileFilter* algorithms (see Alg. 2). Intuitively, given queries that are classified into $0 \sim r$ ranks, ADL sends r combined queries, denoted as $Q_0,\ldots,Q_{r-1}$, to the cloud, each with a different rank. Specifically, for Q_i, the ADL sets the j-th bit to an encryption of 1 if the j-th keyword $Dic[j]$ in the dictionary is chosen by at least one Rank-i query. The cloud then will generate r buffers, denoted as $\mathbb{B}_0,\ldots,\mathbb{B}_{r-1}$, each with a different file survival rate. Specifically, for $\mathbb{B}_i$, the ADL adjusts the mapping time γ_i and the buffer size β_i so that the survival rate of files in $\mathbb{B}_i$ is $q_i = 1 - i/r$, where $0 \le i \le r-1$.

Algorithm 2 The EIRQ-Simple scheme.

```
MatrixConstruct (run by the ADL with public key pk)
  for $i = 0$ to $r - 1$ do
    for $j = 1$ to $d$ do
      if $Dic[j]$ is in Rank-$i$ queries then
        $Q_i[j] = E_{pk}(1)$
      else
        $Q_i[j] = E_{pk}(0)$
    adjust $\gamma_i$ and $\beta_i$ so that survival rate of Rank-$i$ files is $q_i = 1 - i/r$
FileFilter (run by the cloud)
  for $i = 0$ to $r - 1$ do
    for each file $F$ in the cloud do
      for $j = 1$ to $d$ do
        $c = \prod_{Dic[j] \in F} Q_i[j]$; $e = c^{|F|}$
        map $(c, e)$ $\gamma_i$ times to $\mathbb{B}_i$ of size $\beta_i$
```

The main drawback of EIRQ-simple is that it returns redundant files when there are files satisfying more than one ranked query. For example, if F_i is of interest by Rank-0 and Rank-1 queries, it will be returned *twice* (in Rank-0 buffer and Rank-1 buffer, respectively), which wastes the network bandwidth. Therefore, the best case scenario is when there are no files of interest to different ranked queries, and the worst case scenario is when queries of different ranks query the same files.

4.3 The EIRQ-Privacy Scheme

The working process of EIRQ-Privacy is similar to Fig. 2b. The main differences lie in the *MatrixConstruct* and *FileFilter* algorithms (see Alg. 3). Intuitively, EIRQ-Privacy adopts one buffer, with different mapping times for files of different ranks. Let γ_i denote the mapping times for a Rank-i query, and let l be the highest rank of queries that choose the i-th keyword $Dic[i]$ in the dictionary. The mask matrix M is a d-row and m-column matrix, where d is the number of keywords in the dictionary, and $m = \max \gamma_i$. The *MatrixConstruct* algorithm constructs M in the following way: for the i-th row of M that corresponds to $Dic[i]$, the ADL sets $M[i,1],\ldots,M[i,\gamma_l]$ to 1, and $M[i,\gamma_l+1],\ldots,M[i,m]$ to 0, and then encrypts each element under its public key. Note that for a row that corresponds to a Rank-l keyword, the ADL sets the *first*γ_l elements, rather than random γ_l elements, to 1. The reason is to ensure that, given any Rank-l file, when we multiply the rows that correspond to file keywords together in a element-by-element way, the resulting row contains γ_l elements whose values are larger than 0.

Algorithm 3 The EIRQ-Privacy scheme.

```
MatrixConstruct (run by the ADL with public key pk)
  for $i = 0$ to $r - 1$ do
    adjust $\gamma_i$ and $\beta$ so that survival rate of Rank-$i$ files is $q_i = 1 - i/r$
  for $i = 1$ to $d$ do
    set $l$ to be the highest rank of queries choosing $Dic[i]$
    for $j = 1$ to $\max \gamma_i$ do
      if $j \le \gamma_l$ then
        $M[i,j] = E_{pk}(1)$
      else
        $M[i,j] = E_{pk}(0)$
FileFilter(run by the cloud)
  for each file $F_j$ in the cloud do
    for $k = 1$ to $\max \gamma_i$ do
      for $i = 1$ to $d$ do
        $c_{j,k} = \prod_{Dic[i] \in F_j} M[i,k]$; $e_{j,k} = c_{j,k}^{|F_j|}$
        map $(c_{j,k}, e_{j,k})$ once to a buffer of size $\beta$
```

In the *FileFilter* algorithm, for each file F_j, the cloud multiplies the rows that correspond to file keywords, element by element, to form a resulting row. Each element in the resulting row corresponds to a c value. Let $c_{j,1}, \ldots, c_{j,m}$ denote F_j's c values, where $m = \max \gamma_i$. The cloud powers the file content $|F_j|$ to $c_{j,k}$ to form $e_{j,k}$, and maps $(c_{j,k}, e_{j,k})$ to the buffer once, where $1 \leq k \leq m$. Note that with the *MatrixConstruct* algorithm, we can make sure that, for a Rank-l file, the number of c values larger than 0 is γ_l. Therefore, although m c-e pairs will be mapped, only γ_l of them will take effect, which is equal to mapping c-e pairs γ_l times to a buffer.

5 Parameter Setting

5.1 Ostrovsky Parameter Setting

The Ostrovsky scheme has proven that, given f files that match a query, when each file is randomly mapped γ times into a buffer of $2 \cdot f \cdot \gamma$ entries, the file *failure rate* will be lower than $f/2^{\gamma}$, i.e., the file *survival rate* will be higher than $1 - f/2^{\gamma}$. Therefore, given a threshold failure rate $p'{>}0$, if we map each file $\log(f/p')$ times[2] into a buffer of size $2 \cdot f \cdot \log(f/p')$, then the real failure rate p is smaller than p', and the real file survival rate q will be higher than $1 - p'$. Furthermore, we know that, given the estimated number of the matched files, two factors have an impact on file survival rate: the buffer size and the mapping times.

Suppose that queries are classified into $0 \sim r$ ranks, where f'_i files match Rank-i query but mismatch higher ranked queries, and f_i files match Rank-i query. The Ostrovsky parameter setting is as follows: the ADL determines a threshold value $\alpha > 0$, and then adjusts parameters with Eq. (1), (2), and (3). EIRQ-Efficiency filters out a certain percentage of matched files before mapping them into the buffer, and thus all remaining files should be returned. EIRQ-Efficiency adopts one buffer, where the file survival rate is 100 percent. EIRQ-Simple returns multiple buffers with different file survival rates, one for each rank. EIRQ-Privacy still adopts one buffer, but with different mapping times for files of different ranks. Therefore, EIRQ-Efficient will use Eq. (1), EIRQ-Simple will use Eq. (2), and EIRQ-Privacy will use Eq. (3) to adjust the parameters under the Ostrovsky parameter setting

$$\begin{aligned} \gamma &= \log\left(\frac{\sum_{i=0}^{r} f'_i \cdot \left(1 - \frac{i}{r}\right)}{\alpha}\right) \\ \beta &= 2 \cdot \gamma \cdot \sum_{i=0}^{r} f'_i \cdot \left(1 - \frac{i}{r}\right) \end{aligned} \tag{1}$$

$$\gamma_i = \log\left(\frac{f_i}{\left(\frac{i}{r} + \alpha\right)}\right), \; \beta_i = 2 \cdot \gamma_i \cdot f_i \tag{2}$$

$$\gamma_i = \log\left(\frac{f'_i}{\left(\frac{i}{r} + \alpha\right)}\right), \; \beta = \sum_{i=0}^{r} 2 \cdot \gamma_i \cdot f'_i. \tag{3}$$

5.2 Bloom Filter Parameter Setting

An alternative solution is to use Bloom filters [17], [18] to adjust the parameters in our schemes. Bloom filter is a technique that is used to represent a subset S of n members from a universe U. A Bloom filter consists of an array of m bits, all of which are initially set to 0. It uses k independent random hash functions $h_1, \ldots, h_k$, with range $0, \ldots, m-1$, to map each member in U to random k bits. For each member $s \in U$, the bits $h_i(s)$ are set to 1 if $s \in S$; otherwise, they are set to 0, for $1 \leq i \leq k$. A location can be set to 1 multiple times, but only the first change has an effect. To check if a member s is in S, we check whether all $h_i(s)$ are set to 1. If not, then clearly s is not a member of S. Otherwise, there is still a certain probability (false positive) that s is not in S, since k bits may be set to 1 by other members.

The false positive is calculated as follows [17]: after all members of S are hashed, the probability for a specific bit to be 0 is $(1 - 1/m)^{kn} \approx e^{(-kn/m)}$. A false positive occurs when each of k locations of one non-member are set to 1, which is $(1-(1-1/m)^{kn})^{k} \approx (1-e^{(-kn/m)})^{k}$. Let $g = k \cdot \ln(1 - e^{(-kn/m)})$. Minimizing the false positive is equivalent to minimizing k, which in turn is equivalent to minimizing g. When $k = \ln 2 \cdot (m/n)$, we have $dg/dk = 0$. In this case, the false positive is minimized to $(0.6185)^{m/n}$. That is to say, given the number of members n and the threshold false positive p'', we can make the real false positive *approximate*p'' when each member is hashed $\log_{0.6185}(p'') \cdot \ln 2$ times to a Bloom filter of $\log_{0.6185}(p'') \cdot n$ bits. Recall that the file failure rate in EIRQ schemes denotes the probability of a missing file, i.e., the probability that all mappings of each file collide. In a sense, the failure rate is equivalent to the false probability in Bloom filters. Therefore, we let the number of members n, the threshold false positive p'', and the number of bits m in Bloom filters represent the number of files that match the query f, threshold failure rate p', and buffer size β in EIRQ schemes, respectively.

The Bloom filter parameter setting is as follows: the ADL determines a threshold value $\alpha > 0$, and then adjusts parameters with Eqs. (4), (5), and (6). Here, EIRQ-Efficient will use Eq. (4), EIRQ-Simple will use Eq. (5), and EIRQ-Privacy will use Eq. (6) to adjust the parameters under the Bloom filter parameter setting

$$\gamma = \log_{0.6185}(\alpha) \cdot \ln 2, \; \beta = \sum_{i=0}^{r} \left(\frac{\gamma}{\ln 2} \cdot f'_i \cdot \left(1 - \frac{i}{r}\right)\right) \tag{4}$$

$$\gamma_i = \log_{0.6185}\left(\frac{i}{r} + \alpha\right) \cdot \ln 2, \; \beta_i = \frac{\gamma_i}{\ln 2} \cdot f_i \tag{5}$$

$$\gamma_i = \log_{0.6185}\left(\frac{i}{r} + \alpha\right) \cdot \ln 2, \; \beta = \sum_{i=0}^{r} \left(\frac{\gamma_i}{\ln 2} \cdot f'_i\right). \tag{6}$$

Note that, in Eqs. (1) and (4), $p' = \alpha$, and thus, all of the files that are mapped into the buffer will survive with a rate higher than $1 - \alpha$; in Eqs. (2), (3), (5), (6) $p'_i = i/r + \alpha$, and thus the files that match Rank-i queries will survive in the buffer with a rate higher than $1 - i/r - \alpha$.

6 Analysis

6.1 Security Analysis

We will show that EIRQ schemes can provide search privacy, access privacy, and rank privacy as follows.

6.1.1 Search Privacy

In the three schemes, the combined query sent to the cloud is encrypted under the ADL's public key with the Paillier

[2] log is the abbreviation of $\log_2$.

TABLE 1
No Rank vs. Three EIRQ Schemes

Scheme	Computation	Communication under Ostrovsky setting	Communication under Bloom filter setting [2]
No Rank	$O(t)$	$O(d + f \cdot \log(f/\alpha))$	$O(d + f \cdot \log_{0.6185}(\alpha))$
EIRQ-Simple	$O(r \cdot t)$	$O(r \cdot d + \sum_{i=0}^{r}(f_i \cdot \log(f_i/(\alpha + i/r))))$	$O(\sum_{i=0}^{r}(r \cdot d + f_i \cdot \log_{0.6185}(i/r + \alpha)))$
EIRQ-Privacy	$O(\max(\gamma_i) \cdot t)$	$O(\max(\gamma_i) \cdot d + \sum_{i=0}^{r}(f'_i \cdot \log(f'_i/(\alpha + i/r))))$	$O(\max(\gamma_i) \cdot d + \sum_{i=0}^{r}(f'_i \cdot \log_{0.6185}(i/r + \alpha)))$
EIRQ-Efficient	$O(t)$	$O(r \cdot d + \sum_{i=0}^{r}(f'_i \cdot (1 - i/r)) \cdot \log(\frac{\sum_{i=0}^{r} f'_i \cdot (1-i/r)}{\alpha}))$	$O(r \cdot d + \sum_{i=0}^{r}(f'_i \cdot (1 - i/r)) \cdot \log_{0.6185}(\alpha))$

[2] $O(\log_{0.6185}(x)) = O(\log(x))$, as $\log_{0.6185}(x) = \frac{\log(x)}{\log(0.6185)}$, where $1/\log(0.6185)$ is a constant. We keep $\log_{0.6185}(x)$ in the complexity analysis for the ease of comparison.

cryptosystem. The query is a matrix of encrypted 0s and 1s. The Paillier cryptosystem is semantically secure, and the ciphertext of every 1 or 0 is different from other 1s or 0s. Therefore, the cloud cannot deduce what each user is searching for from the encrypted query.

6.1.2 Access Privacy

In the three schemes, the cloud processes the encrypted query on each file in a collection, and maps the processing result into a buffer, which is encrypted with the ADL's public key. The cloud conducts this process for all files in the same way. Therefore, the cloud cannot know which files are actually returned from the encrypted buffer.

6.1.3 Rank Privacy

In EIRQ-Simple, the messages from the ADL to the cloud are r encrypted queries, the buffer size, and the mapping times, where r is the information, which we leak more than [3]. Given r, the cloud only knows the number of query ranks without knowing how many users are in each rank, nor which users are in which ranks. Therefore, EIRQ-Simple can protect the basic level of rank privacy for a user. In EIRQ-Privacy, the message from the ADL to the cloud is a d-row and m-column mask matrix, where d is the number of keywords in the dictionary, and $m = \max \gamma_i$ is the maximal value of mapping times. Here, no extra information is leaked more than [3]. Therefore, EIRQ-Privacy provides a *high level* of user rank privacy. In EIRQ-Efficient, the message from the ADL to the cloud is a d-row and r-column mask matrix, where d is the number of keywords in the dictionary, and r is the lowest rank of user queries. Here, r is the information that we leak more than [3]. Therefore, EIRQ-Efficient can protect the basic level of rank privacy for a user.

6.2 Performance Analysis

We compare the performance between No Rank and the three EIRQ schemes under different parameter settings (see Table 1). In No Rank, the ADL only combines user queries, but does not provide differential query services. In the supplementary file available online, we also provide a comparison of performance between No Rank and the work in [3], [6]. Suppose that queries are classified into $0 \sim r$ ranks, t files stored in the cloud whose keywords constitute a dictionary of size d, f_i files matching Rank-i queries, and f'_i files matching Rank-i queries but mismatching higher ranked queries. Furthermore, in No Rank and EIRQ-Efficient, the threshold file survival rate p' is set to α; in EIRQ-Simple and EIRQ-Privacy, p'_i is set to $i/r + \alpha$.

6.2.1 Computational Cost

We only consider the cost of the exponential operation, which is the most expensive. In both parameter settings, the results are the same. In EIRQ-Simple, the computational cost is r times more than No Rank since, for each ranked query, the cloud needs to process it on the file collection once. In EIRQ-Privacy, the computational cost is $\max(\gamma_i)$ times more than No Rank since, for each file, the cloud needs to execute $\max(\gamma_i)$ exponentiations with the matrix elements. In EIRQ-Efficient, the computational cost is much the same as in No Rank, since the cloud needs to execute exponentiation once for each file.

6.2.2 Communication Cost

Under the Ostrovsky parameter setting, for EIRQ-Simple, the ADL sends r queries, each of which is of size $O(d)$, to the cloud, which will return r buffers to the ADL, each of which is of size $O(f_i \cdot \log(f_i/(i/r + \alpha)))$; for EIRQ-Privacy, the ADL sends a mask matrix of size $O(\max \gamma_i \cdot d)$ to the cloud, which will return a buffer of size $O(\sum_{i=0}^{r}(f'_i \cdot \log(f'_i/(i/r + \alpha))))$ to the ADL; for EIRQ-Efficient, the ADL sends a mask matrix of size $O(r \cdot d)$ to the cloud, which will return a buffer of size $O(\sum_{i=0}^{r}(f'_i \cdot (1 - i/r)) \cdot \log(\frac{\sum_{i=0}^{r} f'_i \cdot (1-i/r)}{\alpha}))$ to the ADL. Under the Bloom filter parameter setting, the query sizes are the same as those under the Ostrovsky parameter setting. The buffer sizes returned from the cloud in EIRQ-Simple, EIRQ-Privacy, and EIRQ-Efficient are $O(\sum_{i=0}^{r}(f_i \cdot \log_{0.6185}(i/r + \alpha)))$, $O(\sum_{i=0}^{r}(f'_i \cdot \log_{0.6185}(i/r + \alpha)))$, and $O(\sum_{i=0}^{r}(f'_i \cdot (1 - i/r)) \cdot \log_{0._{0.6185}}(\alpha))$, respectively.

7 EVALUATION

In this section, we will compare three EIRQ schemes from the following aspects: file survival rate and computation/communication cost incurred on the cloud. Then, based on the simulation results, we deploy our program in Amazon Elastic Compute Cloud (EC2) to test the transfer-in and transfer-out time incurred on the cloud when executing private searches. Note that the energy-performance tradeoff is crucial to the success of cloud computing, and existing energy-saving techniques are hard to directly extend to a cloud environment [19], [20]. As part of our future extensions, we will evaluate the consumed energy

TABLE 2
Parameters

Notation	Description	Value
$\|F\|$	File content	$1KB$
$\|w\|$	Keyword content	$1KB$
n	The number of users	1-100
d	The number of keywords in Dic	100
k	The number of keywords in each query	1-5
w	The number of keywords in each file	1-5
t	The number of files stored in the cloud	$1,000$
r	The lowest user rank	4
α	Threshold value	0.1

overhead in the cloud to verify the effectiveness of our schemes. We use No Rank to denote unranked queries under the ADL. The summary of the experiment parameters are shown in Table 2.

7.1 File Survival Rate

Since queries are classified into $0 \sim 4$ ranks, queries in Rank-0, Rank-1, Rank-2, Rank-3, and Rank-4 should retrieve 100 percent, 75 percent, 50 percent, 25 percent, 0 percent of matched files, respectively. However, in Fig. 3, the real failure rate in EIRQ-Simple and EIRQ-Privacy under the Ostrovsky parameter setting is much lower than i/r, and thus, the real file survival rate is higher than the desired value of $1 - i/r$ (about 25 percent and 50 percent of files are redundantly returned to users); Only EIRQ-Efficient, which filters a certain percentage of matched files before mapping them to a buffer, provides differential query services.

Under the Bloom filter parameter setting, we first obtain corresponding mapping times. Specifically, for file survival rate 100 percent, 75 percent, 50 percent, 25 percent, we have the optimal mapping times 7, 2, 1, 0.4, respectively. Based on these values, the buffer size can be calculated with Eqs. (4), (5), and (6) for different schemes. In practice, γ and β must be integers. Thus, we use $\lfloor\gamma\rfloor$ and $\lfloor\beta\rfloor$ to replace the corresponding values. Using these parameters, the file survival rates for different ranks are shown in Fig. 4, where three EIRQ schemes can provide differential query services, and no bandwidth is wasted in each EIRQ scheme. Therefore, in terms of file survival rate, the Bloom filter parameter setting can achieve better performance than the Ostrovsky parameter setting.

7.2 Computational Cost

As described in Section 6.2, the computational cost is mainly determined by the number of exponentiations performed by the cloud, which is almost the same under the Bloom filter and the Ostrovsky parameter settings. In order to justify the analyses, we will compare the computational cost between No Rank and three EIRQ schemes.

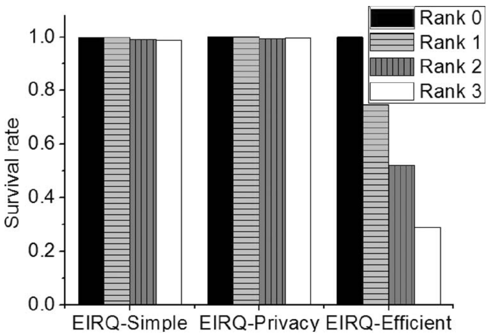

Fig. 3. File survival rate under Ostrovsky setting.

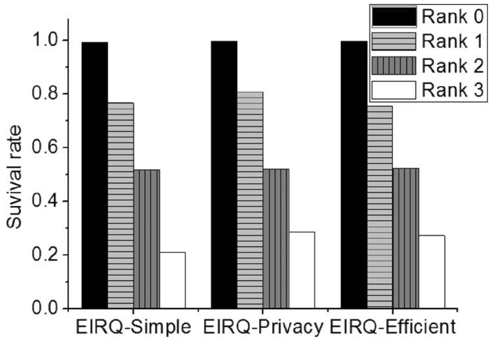

Fig. 4. File survival rate under Bloom filter setting.

The comparisons of computational cost on the cloud are shown in Fig. 5, where the number of queries in each rank ranges from 1 to 25. In Fig. 5a, under the Bloom filter parameter setting, the computational cost is approximately 14.807s in No Rank, 59. 274s in EIRQ-Simple, 101.075s in EIRQ-Privacy, and 14.861s in EIRQ-Efficient. In Fig. 5b, under the Ostrovsky parameter setting, the computational cost approximately ranges from 14.8270s to 14.8788s in No Rank, from 59.1671s to 59.3838s in EIRQ-Simple, from 114. 0475s to 176.5107s in EIRQ-Privacy, and from 14.8664s to 14.9269s in EIRQ-Efficient. In both settings, EIRQ-Privacy consumes the most computation cost, and EIRQ-Efficient, like No Rank, consumes the least computation cost.

7.3 Communication Cost

As described in Section 6.2, the communication cost mainly depends on the buffer size generated by the cloud, which is calculated in different ways under different parameter settings. Furthermore, the buffer size depends on the number of files that match the queries, which is different when users have different *common interests*, i.e., the average number of common keywords among user queries. Therefore, in different parameter settings, we will analyze the buffer size under different common interests. In the following experiments, 1 common keyword, 2 common keywords, and 4 common keywords denote that the average common keywords among user queries are 1, 2,

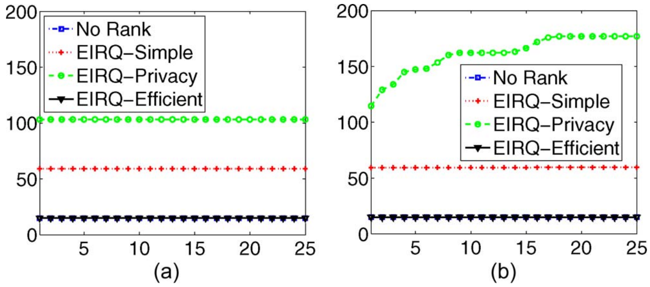

Fig. 5. Comparison of computational cost at the cloud. The x-axis denotes the number of queries in each rank, and the y-axis denotes the computation time (s). (a) Bloom filter parameter setting. (b) Ostrovsky parameter setting.

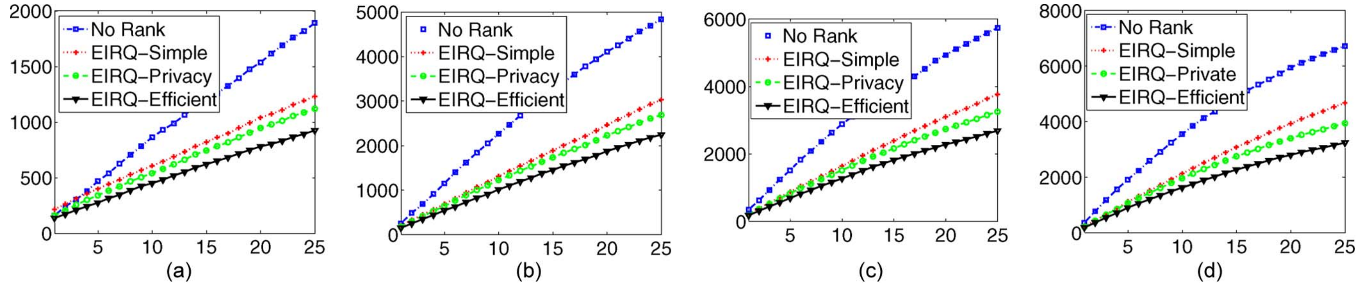

Fig. 6. Comparison of communication cost under the Bloom filter setting. The x-axis denotes the number of queries in each rank, and the y-axis denotes the buffer size (KB). (a) 4 common keywords; (b) 2 common keywords; (c) 1 common keyword; (d) random keywords.

and 4, respectively; random keywords denote that each user randomly chooses keywords for its query.

From Figs. 6 and 7, we know that the EIRQ schemes perform better under the Bloom filter setting compared to under the Ostrovsky setting. Under the Bloom filter setting, all of the EIRQ schemes consume less communication costs than No Rank, e.g., EIRQ-Efficient, EIRQ-Privacy, and EIRQ-Simple can further reduce communication costs by about 50 percent, 35 percent, and 30 percent compared to No Rank, respectively, when the queries share 4 common keywords. Under the Ostrovsky setting, EIRQ-Simple always consumes more bandwidth than No Rank, and EIRQ-Privacy only performs better than No Rank under certain conditions. In both settings, the EIRQ schemes consume less bandwidth as the common interests among users increase. For example, when there are 25 users in each rank under the Bloom filter setting, EIRQ-Efficient only generates a 1 MB buffer under 4 common keywords, but 3 MB under 1 common keyword.

Notice that in both settings, EIRQ-Efficient always has the best performance, the next is EIRQ-Privacy, and the last is EIRQ-Simple. Furthermore, EIRQ-Efficient works better than No Rank when only a few users are conducting searches. For example, when there are 5 queries with 4 common keywords, EIRQ-Efficient generates a buffer of size 274 KB, but No Rank generates a buffer of size 467 KB, under the Bloom filter setting; EIRQ-Efficient generates a buffer of size 439 KB, but No Rank generates a buffer of size 834 KB under the Ostrovsky setting. When there are 5 queries in each rank with 1 common keyword, EIRQ-Efficient generates a buffer of size 687 KB, but No Rank generates a buffer of size 1513 KB, under the Bloom filter setting; EIRQ-Efficient generates a buffer of size 1309 KB, but No Rank generates a buffer of size 3194 KB, under Ostrovsky setting.

7.4 Transfer Time in a Real Cloud

To verify the feasibility of our schemes, we deploy our program in Amazon EC2, to test the transfer-in (receiving query) and transfer-out (sending buffer) time at the cloud. The local machine has an Intel Core 2 Duo E8400 3.0 GHz CPU and 8 GB Linux RAM. We subscribe EC2 amzn-ami-2011.02.1.i386-ebs (ami-8c1fece5) AMI and a small type instance with the following specifications: 32-bit platform, a single virtual core equivalent to 1 compute unit CPU, and 1.7 GB RAM. The average bandwidth from EC2 to the local machine is 33.43 MB/s, and from the local machine to EC2 is 42.98 MB/s.

First, we test the transfer-in time in the real cloud, which is mainly incurred by receiving queries from the ADL. Under both parameter settings, the query size for No Rank, EIRQ-Simple, EIRQ-Privacy, and EIRQ-Efficient can be calculated with $O(d)$, $O(r \cdot d)$, $O(\max \gamma_i \cdot d)$, and $O(r \cdot d)$, respectively. Given $d = 100$, $r = 4$, and $|w| = 1$ KB, the query size for No Rank, EIRQ-Simple, and EIRQ-Efficient is about 100 KB, 400 KB, and 400 KB, respectively. For EIRQ-Privacy, the mapping times are calculated in different ways under different parameter settings. Under the Bloom filter parameter setting, the mapping times are 7, 4, 1, 1, respectively, and thus the query size is about 700 KB. However, under the Ostrovsky parameter setting, the mapping times depend on the number of matched files, which in turn depends on the common interests among queries. The comparisons of transfer-in time are shown in Fig. 8.

Then, we test the transfer-out time at the cloud, which is mainly incurred by returning files to the ADL. The results are shown in Figs. 9 and 10. In all cases, EIRQ-Efficient consumes the least amount of transfer time, and EIRQ-Simple works better than No-Rank under the Bloom filter setting. For example, under the Ostrovsky scheme, No-Rank consumes

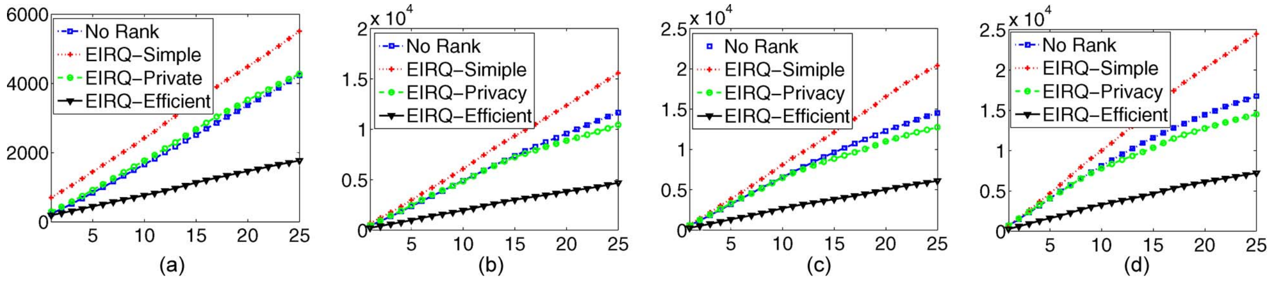

Fig. 7. Comparison of communication cost under the Ostrovsky setting. The x-axis denotes the number of queries in each rank, and the y-axis denotes the buffer size (KB). (a) 4 common keywords; (b) 2 common keywords; (c) 1 common keyword; (d) random keywords.

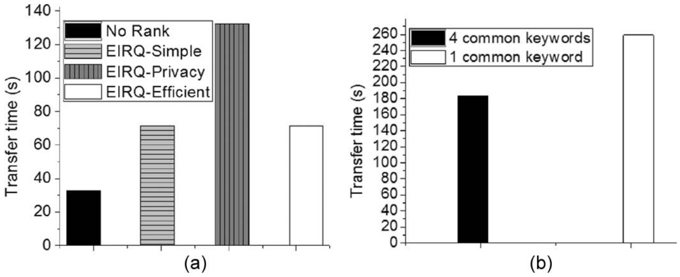

Fig. 8. Comparison of transfer-in time. (a) Comparision under Bloom filter setting; (b) EIRQ-privacy under Ostrovsky setting.

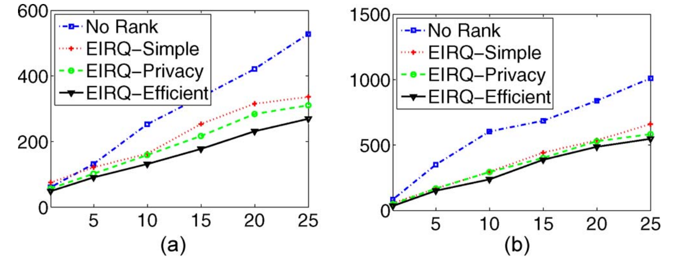

Fig. 9. Transfer-out time in real cloud under the Bloom filter setting. The x-axis denotes the number of users, and the y-axis denotes transferring time (s). (a) 4 common keywords; (b) 1 common keyword.

from 83.6s to 1191.8s, EIRQ-simple consumes from 189.8s to 1597.6s, EIRQ-Privacy consumes from 83.3s to 1099.9s, and EIRQ-Efficient consumes from 57.4s to 475.1s when there are 4 common keywords; No-Rank consumes from 191.1s to 3857.5s, EIRQ-simple consumes from 181.5s to 5369.7s, EIRQ-Privacy consumes from 161.8s to 3323.4s, and EIRQ-Efficient consumes from 81.3s to 1502.7s when there is 1 common keyword.

Therefore, EIRQ-Efficient is most suitable to be deployed to a cloud environment. For example, the time to transfer a query from the ADL to the cloud consumes less than 100 seconds, and the time to transfer the buffer from the cloud to the ADL consumes less than 500 seconds, under 4 common keywords.

8 Conclusion

In this paper, we proposed three EIRQ schemes based on an ADL to provide differential query services while protecting user privacy. By using our schemes, a user can retrieve different percentages of matched files by specifying queries of different ranks. By further reducing the communication cost incurred on the cloud, the EIRQ schemes make the private searching technique more applicable to a cost-efficient cloud environment. However, in the EIRQ schemes, we simply determine the rank of each file by the highest rank of queries it matches. For our future work, we will try to design a flexible ranking mechanism for the EIRQ schemes.

Acknowledgment

This research was supported in part by NSF grants ECCS 1231461, ECCS 1128209, CNS 1138963, CNS 1065444, and CCF 1028167; NSFC grants 61272151 and 61073037, ISTCP grant 2013DFB10070, the China Hunan Provincial Science & Technology Program under Grant Number 2012GK4106, and the ''Mobile Health'' Ministry of Education-China Mobile Joint Laboratory (MOE-DST No. [2012]311); and Fundamental Research Funds for the Central Universities.

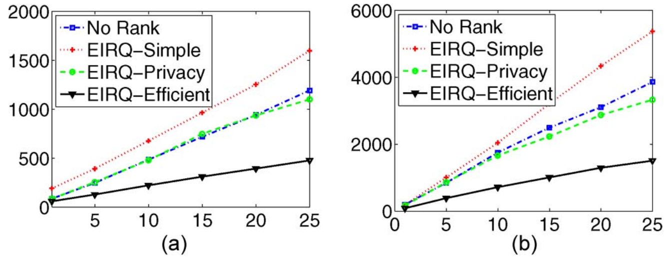

Fig. 10. Transfer-out time in real cloud under the Ostrovsky setting. The x-axis denotes the number of users, and the y-axis denotes the transferring time (s). (a) 4 common keywords; (b) 1 common keyword.

References

[1] P. Mell and T. Grance, ''The NIST Definition of Cloud Computing (Draft),'' in *NIST Special Publication*. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2011.

[2] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, ''Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions,'' in *Proc. ACM CCS*, 2006, pp. 79-88.

[3] R. Ostrovsky and W. Skeith, ''Private Searching on Streaming Data,'' in *Proc. CRYPTO*, 2005, pp. 233-240.

[4] R. Ostrovsky and W. Skeith, ''Private Searching on Streaming Data,'' *J. Cryptol.*, vol. 20, no. 4, pp. 397-430, Oct. 2007.

[5] J. Bethencourt, D. Song, and B. Waters, ''New Constructions and Practical Applications for Private Stream Searching,'' in *Proc. IEEE SP*, 2006, pp. 1-6.

[6] J. Bethencourt, D. Song, and B. Waters, ''New Techniques for Private Stream Searching,'' *ACM Trans. Inf. Syst. Security*, vol. 12, no. 3, p. 16, Jan. 2009.

[7] Q. Liu, C. Tan, J. Wu, and G. Wang, ''Cooperative Private Searching in Clouds,'' *J. Parallel Distrib. Comput.*, vol. 72, no. 8, pp. 1019-1031, Aug. 2012.

[8] G. Danezis and C. Diaz, ''Improving the Decoding Efficiency of Private Search,'' Int'l Assoc. Cryptol. Res., IACR Eprint Archive No. 024, Schloss Dagstuhl, Germany, 2006.

[9] G. Danezis and C. Diaz, ''Space-Efficient Private Search with Applications to Rateless Codes,'' in *Proc. Financial Cryptogr. Data Security*, 2007, pp. 148-162.

[10] M. Finiasz and K. Ramchandran, ''Private Stream Search at the Same Communication Cost as a Regular Search: Role of LDPC Codes,'' in *Proc. IEEE ISIT*, 2012, pp. 2556-2560.

[11] X. Yi and E. Bertino, ''Private Searching for Single and Conjunctive Keywords on Streaming Data,'' in *Proc. ACM Workshop Privacy Electron. Soc.*, 2011, pp. 153-158.

[12] B. Hore, E.-C. Chang, M.H. Diallo, and S. Mehrotra, ''Indexing Encrypted Documents for Supporting Efficient Keyword Search,'' in *Proc. Secure Data Manage.*, 2012, pp. 93-110.

[13] P. Paillier, ''Public-Key Cryptosystems Based on Composite Degree Residuosity Classes,'' in *Proc. EUROCRYPT*, 1999, pp. 223-238.

[14] Q. Liu, C.C. Tan, J. Wu, and G. Wang, ''Efficient Information Retrieval for Ranked Queries in Cost-Effective Cloud Environments,'' in *Proc. IEEE INFOCOM*, 2012, pp. 2581-2585.

[15] S. Yu, C. Wang, K. Ren, and W. Lou, ''Achieving Secure, Scalable, Fine-Grained Data Access Control in Cloud Computing,'' in *Proc. IEEE INFOCOM*, 2010, pp. 1-9.

[16] G. Wang, Q. Liu, J. Wu, and M. Guo, ''Hierarchical Attribute-Based Encryption and Scalable User Revocation for Sharing Data in Cloud Servers,'' *Comput. Security*, vol. 30, no. 5, pp. 320-331, July 2011.

[17] M. Mitzenmacher, ''Compressed Bloom Filters,'' *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, pp. 604-612, Oct. 2002.

[18] D. Guo, J. Wu, H. Chen, and X. Luo, ''Theory and Network Applications of Dynamic Bloom Filters,'' in *Proc. IEEE INFOCOM*, 2006, pp. 1-12.

[19] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M.Q. Dang, and K. Pentikousis, ''Energy-Efficient Cloud Computing,'' *Comput. J.*, vol. 53, no. 7, pp. 1045-1051, Sept. 2010.

[20] E. Gelenbe, R. Lent, and M. Douratsos, ''Choosing a Local or Remote Cloud,'' in *Proc. NCCA*, 2012, pp. 25-30.

Qin Liu received the BSc degree in computer science from Hunan Normal University, Changsha, China, in 2004, and the MSc and PhD degrees in computer science both from Central South University, Changsha, China, in 2007 and 2012, respectively. She has been a Visiting Student at Temple University, Philadelphia, PA, USA. Her research interests include security and privacy issues in cloud computing. She is a Assistant Professor in College of Information Science and Engineering at Hunan University, Changsha, Hunan, China.

Chiu C. Tan received the PhD in computer science from the College of William and Mary, Williamsburg, VA, USA, in 2010. He is a Research Assistant Professor in the Computer and Information Sciences Department at Temple University, Philadelphia, PA, USA. His research interests include wireless security (802.11, vehicular, RFID), cloud computing security, and security for mobile health (mHealth) systems. He is a member of IEEE.

Jie Wu is the Chair and a Laura H. Carnell Professor in the Department of Computer and Information Sciences at Temple University, Philadelphia, PA, USA. Prior to joining Temple University, he was a Program Director at the National Science Foundation and a Distinguished Professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Dr. Wu regularly published in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including *IEEE Transactions on Computers*, *IEEE Transactions on Service Computing*, and Journal of Parallel and Distributed Computing. Dr. Wu was a general chair for IEEE IPDPS 2008 and IEEE ICDCS 2013 and program co-chair/chair for IEEE INFOCOM 2011 and CCF CNCC 2013. Currently, he is serving as general chair for ACM MobiHoc 2014. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a CCF Distinguished Speaker and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.

Guojun Wang received the BSc degree in geophysics, in 1992, the MSc degree in computer science, in 1996, and the PhD degree in computer science, in 2002, all from Central South University, Changsha, China. He is now Chair and Professor of the Department of Computer Science at Central South University. He is also the Director of Trusted Computing Institute of the University. He has been an Adjunct Professor at Temple University, USA, a Visiting Scholar at Florida Atlantic University, USA, a Visiting Researcher at the University of Aizu, Japan, and a Research Fellow at the Hong Kong Polytechnic University. His research interests include network and information security, Internet of things, and cloud computing. He is a Senior Member of CCF and a member of IEEE, ACM, and IEICE.