Efficient Information Retrieval for Ranked Queries in Cost-Effective Cloud Environments

Qin Liu[†‡], Chiu C. Tan[‡], Jie Wu[‡], and Guojun Wang[*†]
[†]School of Information Science and Engineering, Central South University, Changsha, Hunan Province, P. R. China, 410083
[‡]Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA
*Correspondence to: csgjwang@mail.csu.edu.cn

***Abstract*—Cloud computing as an emerging technology trend is expected to reshape the advances in information technology. In this paper, we address two fundamental issues in a cloud environment: privacy and efficiency. We first review a private keyword-based file retrieval scheme proposed by Ostrovsky et. al. Then, based on an aggregation and distribution layer (ADL), we present a scheme, termed efficient information retrieval for ranked query (EIRQ), to further reduce querying costs incurred in the cloud. Queries are classified into multiple ranks, where a higher ranked query can retrieve a higher percentage of matched files. Extensive evaluations have been conducted on an analytical model to examine the effectiveness of our scheme.**

***Index Terms*—Cloud computing, efficiency, privacy.**

I. Introduction

Cloud computing as an emerging technology is expected to reshape the information technology processes in the near future. A typical cloud application would have a data owner outsourcing data services to a cloud, where the data is stored in a keyword-value form, and users could retrieve the data with several keywords. Since a cloud is operated by a third party, there have been some concerns over the possible privacy leaks that may occur. Such concerns have led researchers to propose various techniques to protect user privacy.

A key privacy search solution was proposed by Ostrovsky et al. [1], which can provide the same privacy level as downloading the entire database from the cloud with significantly less communication costs. By asking the cloud to return the entire database, the cloud cannot know which files are really interested by a user. However, the Ostrovsky scheme has a high computation cost, since it must require the cloud to process the encrypted query on *every* file in a collection; Otherwise, the cloud will learn that certain files are not related to that user's query. Therefore, 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 [2], also have the same drawback.

To make private search applicable in a cloud environment, in our previous work [3], we introduce an *aggregation and distribution layer* (ADL)-a middleware layer between the users and the cloud. We envision that an ADL will be deployed in an organization that has outsourced the data operations to a cloud. The ADL will aggregate queries from multiple users and send a *combined* query to the cloud. Given the combined query, the cloud needs to execute the query only once and return all matched files to the ADL. Furthermore, since the files of most interest to the users need to be returned only once, the communication costs will also be reduced. To illustrate, let us assume that files F_1, F_2, and F_3, which are stored in the cloud, are described with keywords "A, B", "B", and "C", and Alice and Bob query data with "A, B" and "A, C", respectively. Under the ADL, the cloud needs to execute the query only once to return F_1, F_2, F_3 to the ADL. Compared to the Ostrovsky scheme, the computation and communication costs are saved by 50% and 25%, respectively. Note that introducing the ADL will incur some processing delay for aggregating queries. However, the degree of aggregation can be controlled through a time-out mechanism to meet a given processing delay requirement. When the time-out is set to zero, this is degraded to the normal sequential search.

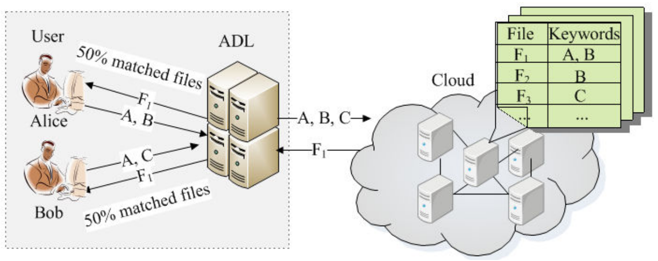

Fig. 1. Alice and Bob want to retrieve 50% of files that match their queries.

In this paper, we propose an improvement based on our previous work, where the cloud can return a certain percentage of matched files to the user. This is motivated by the fact that under certain cases, a user may only be interested in a certain percentage of matched files. By returning a smaller percentage of files, the communication cost can be reduced. To illustrate, let us assume that both Alice and Bob only wants 50% of the files that match their queries as shown in Fig. 1. To satisfy their requirements, the cloud only needs to return F_1 to the ADL, where the communication cost is further saved by 67%.

Motivated by this goal, we design a scheme, termed Efficient Information retrieval for Ranked Query (EIRQ), where each user can choose the rank of his query to determine the percentage of returned matched files. In general, the higher the rank of the query is, the higher the percentage of returned matched files. The basic idea of EIQR is to construct a privacy-preserving *mask matrix* that allows the cloud to filter out a certain percentage of matched files. This is not a trivial work, since the cloud needs to correctly filter out files according to the rank of queries, without knowing the rank of each query nor which files are returned/ filtered out.

Our key contributions are as follows: (1) EIRQ can provide *differential query services*, where the queries with higher rank can retrieve higher percentage of matched files. (2) EIRQ can perfectly protect user privacy while providing differential query services. (3) Extensive experiments were performed on an analytical model to validate our scheme.

II. Related work

Our work aims to provide differential query services while protecting user privacy. To the best of our knowledge, no previous works have addressed this problem. Existing research that is similar to ours can be found in the areas of private searching and ranked searchable encryption.

Private searching is proposed by [1], where the data is stored in the clear form, and the query is encrypted with the Paillier cryptosystem. The cloud stores all files into a compact buffer, with which the user can successfully recover all wanted files with high probability. In the following work, [2] reduced the communication cost in [1] by solving a set of linear programs; [4] presented an efficient decoding mechanism for private searching. The main drawback of the current private searching techniques is that both the computation and communication costs grow linearly with the number of users that are executing searches. Thus, when applying these schemes to a large-scale cloud environment, querying costs will be extensive.

Ranked searchable encryption enables users to retrieve the most matched files from the cloud in the case that both the query and data are in the encrypted form. The work by [5], which only supports single-keyword searches, encrypts files and queries with Order Preserving Symmetric Encryption (OPSE) [6] and utilizes keyword frequency to rank results. Their following work [7], which supports multiple-keyword searches, uses the secure KNN technique [8] to rank results based on inner products. The main limitation of these approaches is that user access privacy [9] will not be preserved.

III. Background

A. System Model

The system consists of three types of entities: cloud, aggregation and distribution layer (ADL), and users. For ease of explanation, in this paper, we only use a single ADL, but multiple ADLs can be deployed as necessary. Multiple files are stored in a potentially untrusted cloud, where each file is described by several distinct keywords. The union of all keywords form a *dictionary*. Users will query the ADL using keywords from the dictionary. The ADL will aggregate user queries and query the cloud with a combined query. The cloud will return the files matching the combined query to the ADL.

To reduce the communication cost, a different query service is provided to allow each user to select a particular *rank* for his query. In general, the higher the rank of the user query is, the more matched files will be returned to the user. This feature is useful when there are a large number of files that match a user's query, but the user only needs a small subset of them. We can illustrate this using the following example. Let us assume that Alice wants to retrieve 2% of the files that contain keywords "A, B", and Bob wants to retrieve 20% of the files that contain keywords "A, C". Suppose that the cloud holds 500 files described by keywords "A, B" and 500 files described by keywords "A, C". Without combination, the cloud will have to return 2000 files, and without ranking, the cloud will have to return 1000 files, but only 110 of which are actually needed. Fig. 2 illustrates the differences. The notations used in our schemes are shown in Table I.

TABLE I
SUMMARY OF NOTATIONS

Notation	Description
Dic	Public dictionary
d	Number of keywords in Dic
r	Highest user rank
Q	Query
M	Mask matrix
F_i, $\|F_i\|$	File name, file content

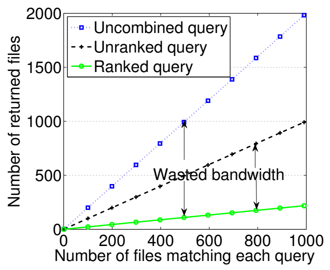

Fig. 2. Comparison of communication costs among different queries.

B. Security Requirements

The ADL is assumed to be *trusted* by all of the users, and the communication channels are assumed to be secured under SSH. Each user individually sends the query to the ADL, which will distribute appropriate files to each user. As long as the ADL is trusted and correctly executes our schemes, the user cannot know anything about other users' interests. Thus, the cloud is the only adversary for each user. The cloud is assumed to be *honest but curious*. That is, it will obey our schemes, but still want to know some additional information.

User privacy can be divided into *search privacy* and *access privacy*, where the cloud neither learns what the user is searching for nor which files are returned to a user. Since user queries are classified into multiple ranks, *rank privacy*, a new kind of user privacy, also needs to be protected against the cloud. Rank privacy entails hiding the rank of each query from the cloud, i.e., the cloud provides differential query services without knowing which level of service is chosen by the user. Our security goal is to thoroughly protect user search privacy, access privacy, and rank privacy against the cloud.

C. Overview of the Ostrovsky Scheme

We briefly introduce the Ostrovsky scheme [1], which relies on a public key cryptosystem, the *Paillier cryptosystem*. Let

Algorithm 1 Ostrovsky scheme

```
***GenerateQuery (run by user)***
**for** $i$=1 **to** $d$ **do**
    **if** the *i-th* keyword in the dictionary $Dic[i]$ is chosen **then**
        $Q[i] = 1$ **else** $Q[i] = 0$
    Encrypt $Q[i]$ with the user's public key
***PrivateSearch (run by cloud)***
**for** each file $F_j$ in the cloud **do**
    **for** $i$=1 **to** $d$ **do**
        $c_j = \prod_{Dic[i] \in F_j} Q[i]$; $e_j = c_j^{|F_j|}$
        Multiply $(c_j, e_j)$ many times in a compact buffer
***FileRecover (run by user)***
Decrypt the buffer to obtain plaintext pair $(c'_j, e'_j)$
**if** $c'_j \neq 0$ **then** recover file content with $e'_j/c'_j$
```

TABLE II
SAMPLE FILES IN THE CLOUD

File name	File keywords	File content
F_1	A, B	$\|F_1\|$
F_2	B, C	$\|F_2\|$
F_3	C, D	$\|F_3\|$
F_4	C	$\|F_4\|$
F_5	D	$\|F_5\|$

$E(m)$ denote the encryption of plaintext m. The Paillier cryptosystem has the following homomorphic properties:

- $E(a) \cdot E(b) = E(a + b)$
- $E(a)^b = E(a \cdot b)$

The Ostrovsky scheme consists of three algorithms, as shown in Alg. 1. We use a simple example to illustrate its working process as follows: public dictionary $Dic = \langle A, B, C, D \rangle$ and files stored in the cloud are as in Table II; A user, Alice, wishes to retrieve files with keywords "A, B".

In the first step, Alice runs the *GenerateQuery* algorithm to generate a query $Q = \langle E(1), E(1), E(0), E(0) \rangle$, where each entry is an encryption of 1 if the corresponding keyword is chosen; otherwise it is 0.

In the second step, the cloud runs the *PrivateSearch* algorithm to generate *occurrence-content pairs*. For example, user keywords "A, B" appear in F_1, both of which correspond to $E(1)$ in user query Q. Thus, the occurrence of the user keywords is the product of corresponding entries in Q, i.e., $c_1 = E(1) \cdot E(1) = E(1 + 1) = E(2)$. File content is then powered by the occurrence, i.e., $e_1 = c_1^{|F_1|} = E(2 \cdot |F_1|)$.

Then, the cloud maps each pair many times to a compact buffer. For each buffer entry, there are three statuses: *survival*, *collision*, and *mismatch*, where a collision will appear only when more than one matched file is mapped, a survival will appear when only one matched file is mapped, and a mismatch will appear when unmatched files are mapped. For example, in Fig. 3, the second entry is a collision. When a collision happens, no files in the entry can be recovered.

In the third step, Alice runs the *FileRecover* algorithm to recover files. Note that if a file is mismatching Q, then the occurrence is an encryption of 0, and the file content is processed to be an encryption of 0. Otherwise, the occurrence is an encryption of some value v larger than 0, and the file content is processed to be an encryption of $v \cdot |F_j|$. Therefore, the user can obtain file content by dividing the content by the occurrence. This scheme also provides a collision-detection mechanism to let the user get rid of the conflicting copies. We refer readers to [1] for more details.

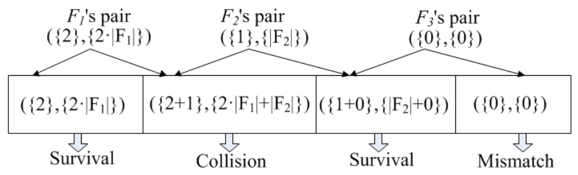

Fig. 3. Mapping files to a buffer. $\{a\}$ is used to denote $E(a)$; thus, $E(a) \cdot E(b)$ and $E(a)^b$ are replaced with $\{a + b\}$ and $\{a \cdot b\}$, respectively.

IV. Protocol Description

A. Intuition

The basic idea of EIQR is that a privacy-preserving *mask matrix* is used to filter out a certain percentage of files before mapping them to a buffer. Before illustrating EIQR, two fundamental problems should be resolved:

First, we should determine the relationship between query rank and the percentage of returned matched files. Suppose that queries are classified into r ranks, where 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 statuses matched files. Therefore, Rank-0 queries can retrieve 100% of the matched files, and Rank-r queries cannot retrieve any files.

Second, 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.

B. EIRQ

EIRQ consists of four algorithms, as shown in Fig. 4. We will use the following example to describe its working process. The dictionary and files are the same as in Section III-(C); users are classified into four ranks, where Alice, a Rank-0 user, queries with keywords "A, B", and Bob, a Rank-1 user, uses keywords "A, C". According to our rules, "A, B" are Rank-0 keywords, "C" is a Rank-1 keyword, and "D" is a Rank-4 keyword. Correspondingly, F_1 and F_2 are Rank-0 files which will be returned with a probability of 1, F_3 and F_4 are Rank-1 files which will be returned with a probability of 75%, and F_5 is a Rank-4 file which will not be returned.

Step 1: Each user runs the *QueryGen* algorithm to send a query to the ADL, where the user query consists of the chosen keywords and the query rank.

Step 2: Given users' queries, the ADL runs the *Matrix-Construct* algorithm (Alg. 2) to send a *mask matrix* to the

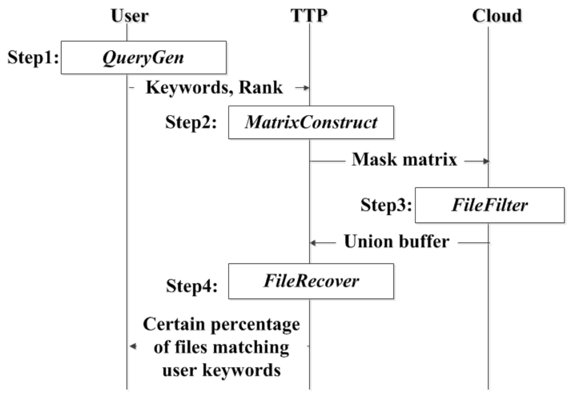

Fig. 4. Working process of EIRQ.

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 highest rank of queries. The mask matrix M can be constructed as follows: For each keyword w, the ADL first sets w's rank with l, the highest query rank choosing this keyword. Then, for the row corresponding to keyword w, the ADL sets the first $r-l$ columns to 1 and the last l columns to 0. The example mask matrix is shown in Fig. 5-(a). Note that, the reason for setting the first $r-l$ columns, rather than random $r-l$ columns, to 1 is to ensure that, given any two files with rank l, the probability of the product of the columns corresponding to file keywords being 0 is l/r.

Step 3: Based on the mask matrix, the cloud runs the *FileFilter* algorithm (Alg. 3) to filter out a certain percentage of matched files and returns a union buffer to the ADL. The process is as follows: For each file F_j, the cloud first multiplies the *k-th* columns that correspond to F_j's keywords in the mask matrix to obtain c_j, where $k = j \mod r$. The example columns chosen for each file are as shown in Fig. 5-(b). Then, the cloud powers the file content to c_j to obtain e_j and maps (c_i, e_i) to many entries of a union buffer as the Ostrovsky scheme. Here, c_j denotes the occurrence of ranked keywords in file F_j. Thus, c_j will be larger than 0, and file F_j will be returned only when $l + k \leq r$, where $k = j \mod r$.

Step 4: The ADL runs the *ResultDivide* algorithm to distribute files to each user. The ADL first recovers all files that match user queries as the *FileRecover* algorithm in the Ostrovsky scheme. Then, the ADL distributes appropriate files to each user based on the user queries. To make sure that the ADL distributes files correctly, we can require the cloud to attach file keywords with the file content. Thus, the ADL can find out all of the files that match each user's query by executing keyword searches.

V. Security analysis

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

Search privacy. In EIRQ, the combined query (the mask matrix) from the ADL to the cloud is encrypted with the ADL's public key. Therefore, the cloud cannot deduce what each user is searching for from the encrypted query.

Algorithm 2 MatrixConstruct

```
for $i = 1$ to $d$ do
    Set $l$ to be the highest query rank choosing the i-th keyword in $Dic$
    for $j = 1$ to $r$ do
        if $l + j \leq r$ then $M[i,j] = 1$ else $M[i,j] = 0$
        Encrypt $M[i,j]$ with the ADL's public key
```

Algorithm 3 FileFilter

```
for each file $F_j$ stored in the cloud do
    for $i = 1$ to $d$ do
        $k = j \mod r$; $c_j = \prod_{Dic[i] \in F_j} M[i,k]$; $e_j = c_j^{|F_j|}$
        Multiply pair $(c_j, e_j)$ many times to a compact buffer
```

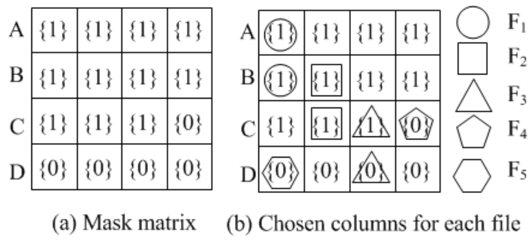

Fig. 5. Sample mask matrix and chosen columns.

Access privacy. In EIRQ, the cloud processes each file similarly to generate a compact buffer where unmatched files are encrypted to 0, while conducting searches. The buffer returned to the ADL is encrypted with the ADL's public key. Therefore, the cloud cannot know which files are actually returned from the encrypted buffer.

Rank privacy. In EIRQ, the mask matrix from the ADL to the cloud is a d-row and r-column matrix, where r is the information that is the information that we leak more than [1]. Given r, the cloud only knows that all users are classified into r ranks without knowing how many users are in each rank, nor which users are in which ranks. Therefore, user rank privacy is protected.

VI. Evaluation

In this section, we will first show the percentage of returned matched files for each ranked query to justify that EIQR can provide differential query services. Then, we will compare computation and communication costs in the cloud between EIRQ and private search under ADL without ranking (ADL). We will set buffer size and mapping times based on the Ostrovsky scheme. The parameters used in experiments are shown in Table III.

A. Percentage of Returned Files

Since users are classified into 0 to 4 ranks, queries in Rank-0, Rank-1, Rank-2, Rank-3, and Rank-4 should retrieve $100\%, 75\%, 50\%, 25\%, 0\%$ of the files that match their queries, respectively. From Fig. 6, we know that Rank-0, Rank-1, Rank-2, Rank-3, and Rank-4 can retrieve

TABLE III
PARAMETERS

Notation	Description	Value
$\|F\|$	File content	$1KB$
n	The number of users	1-100
d	The number of keywords in the dictionary	100
q	The number of keywords in each query	1-5
f	The number of keywords in each file	1-5
t	The number of files stored in the cloud	10^3
r	The highest user rank	4

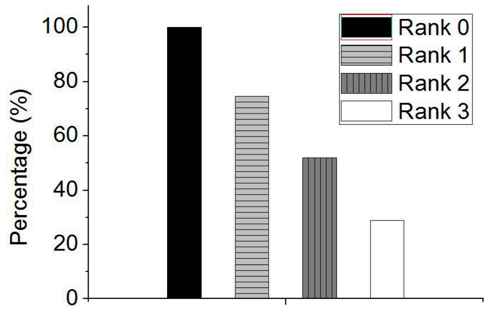

Fig. 6. Percentage of returned files.

$100\%, 75\%, 52\%, 29\%, 0\%$ of matched files, which justify that EIRQ can provide differential query services.

B. Computation Cost

Since the computational cost is mainly determined by the number of exponentiations [1], we will compare the computation cost between ADL and EIRQ under different parameter settings. In each setting, the number of users in each rank is from 1 to 25, where each user randomly chooses 1-5 keywords from the dictionary of 100 keywords. The comparisons of computation cost are shown in Fig. 7, where the computation cost approximately ranges from $14.8270s$ to $14.8788s$ in ADL, and from $14.8664s$ to $14.9269s$ in EIRQ.

C. Communication Cost

We will compare the buffer size between EIRQ and ADL. The buffer size depends on the number of files matching queries, which is different when users have different *common interests*, which can be calculated with $1 - q/\sum_{i=1}^{n} q_i$, where q is the number of keywords in the combined query, and q_i is the number of keywords in the *i-th* user's query. Therefore, we will analyze the buffer size under different common interests (4 common keywords and 1 common keyword).

From Fig. 8, we know that EIRQ consumes less bandwidth as the common interests increase. Note that EIRQ still works better than ADL when only a few users are conducting searches. For example, when there are 5 users in each rank querying with 4 common keywords, EIRQ generates only a $439KB$ buffer, but ADL generates a $834KB$ buffer.

VII. Conclusion

In this paper, we proposed a scheme based on an ADL to allow secure differential query services for a cloud environment. By using our scheme, users of different ranks can retrieve different percentages of files that match their queries so as to make the cloud services more scalable and flexible. The main drawback is that the assumption of having a trusted third party may not be realistic. For our future work, we will explore an extension of our solution that would apply to the case where we don't need to trust the ADL.

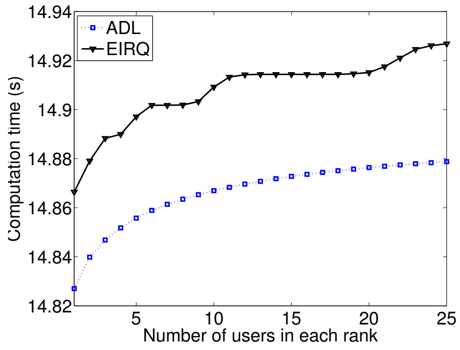

Fig. 7. Comparison of computation cost in the cloud.

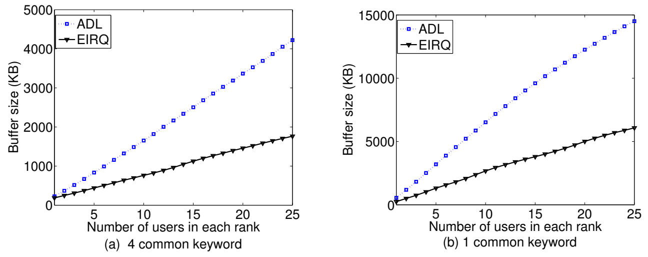

Fig. 8. Comparison of communication cost.

Acknowledgments

This research was supported in part by the NSF grants ECCS 1128209, CNS 1065444, CCF 1028167, CNS 0948184, CCF 0830289; and the National Natural Science Foundation of China under Grant No. 61073037, the Hunan Provincial Science and Technology Program under Grant No. 2010GK2003, and the National 973 Basic Research Program of China under Grant No. 2011CB302800.

References

[1] R. Ostrovsky and W. Skeith III, "Private searching on streaming data," in *Proc. of ACM CRYPTO*, 2005.
[2] J. Bethencourt, D. Song, and B. Waters, "New techniques for private stream searching," *ACM Transactions on Information and System Security*, 2009.
[3] Q. Liu, C. C. Tan, J. Wu, and G. Wang, "Cooperative private searching in clouds," $http://www.cis.temple.edu/\ cctan/TR_1.pdf$, Tech. Rep., 2011.9.
[4] G. Danezis and C. Diaz, "Improving the decoding efficiency of private search," in *IACR Eprint archive number 024*, 2006.
[5] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," in *Proc. of IEEE ICDCS*, 2010.
[6] A. Boldyreva, N. Chenette, Y. Lee, and A. Oneill, "Order-preserving symmetric encryption," *Advances in Cryptology-EUROCRYPT*, 2009.
[7] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," in *Proc. of IEEE INFOCOM*, 2011.
[8] W. Wong, D. Cheung, B. Kao, and N. Mamoulis, "Secure knn computation on encrypted databases," in *Proc. of ACM SIGMOD*, 2009.
[9] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proc. of ACM CCS*, 2006.