Secure Ranked Keyword Search over Encrypted Cloud Data

Cong Wang[†], Ning Cao[‡], Jin Li[†], Kui Ren[†], and Wenjing Lou[‡]
[†]Department of ECE, Illinois Institute of Technology, Chicago, IL 60616
Email: {cong, jli, kren}@ece.iit.edu
[‡]Department of ECE, Worcester Polytechnic Institute, Worcester, MA 01609
Email: {ncao, wjlou}@ece.wpi.edu

***Abstract*—As Cloud Computing becomes prevalent, sensitive information are being increasingly centralized into the cloud. For the protection of data privacy, sensitive data has to be encrypted before outsourcing, which makes effective data utilization a very challenging task. Although traditional searchable encryption schemes allow users to securely search over encrypted data through keywords, these techniques support only *boolean* search, without capturing any relevance of data files. This approach suffers from two main drawbacks when directly applied in the context of Cloud Computing. On the one hand, users, who do not necessarily have pre-knowledge of the encrypted cloud data, have to postprocess every retrieved file in order to find ones most matching their interest; On the other hand, invariably retrieving all files containing the queried keyword further incurs unnecessary network traffic, which is absolutely undesirable in today's pay-as-you-use cloud paradigm.**

In this paper, for the first time we define and solve the problem of effective yet secure ranked keyword search over encrypted cloud data. Ranked search greatly enhances system usability by returning the matching files in a ranked order regarding to certain relevance criteria (e.g., keyword frequency), thus making one step closer towards practical deployment of privacy-preserving data hosting services in Cloud Computing. We first give a straightforward yet ideal construction of ranked keyword search under the state-of-the-art searchable symmetric encryption (SSE) security definition, and demonstrate its inefficiency. To achieve more practical performance, we then propose a definition for ranked searchable symmetric encryption, and give an efficient design by properly utilizing the existing cryptographic primitive, order-preserving symmetric encryption (OPSE). Thorough analysis shows that our proposed solution enjoys "as-strong-as-possible" security guarantee compared to previous SSE schemes, while correctly realizing the goal of ranked keyword search. Extensive experimental results demonstrate the efficiency of the proposed solution.

I. INTRODUCTION

Cloud Computing enables cloud customers to remotely store their data into the cloud so as to enjoy the on-demand high quality applications and services from a shared pool of configurable computing resources [1]. The benefits brought by this new computing model include but are not limited to: relief of the burden for storage management, universal data access with independent geographical locations, and avoidance of capital expenditure on hardware, software, and personnel maintenances, etc [2].

With the prevalence of cloud services, more and more sensitive information are being centralized into the cloud servers, such as emails, personal health records, private videos and photos, company finance data, government documents, etc [3]. To protect data privacy and combat unsolicited accesses, sensitive data has to be encrypted before outsourcing [4] so as to provide end-to-end data confidentiality assurance in the cloud and beyond. However, data encryption makes effective data utilization a very challenging task given that there could be a large amount of outsourced data files. Besides, in Cloud Computing, data owners may share their outsourced data with a large number of users, who might want to only retrieve certain specific data files they are interested in during a given session. One of the most popular ways to do so is through keyword-based search. Such keyword search technique allows users to selectively retrieve files of interest and has been widely applied in plaintext search scenarios [5]. Unfortunately, data encryption, which restricts user's ability to perform keyword search and further demands the protection of keyword privacy, makes the traditional plaintext search methods fail for encrypted cloud data.

Although traditional searchable encryption schemes (e.g. [6]–[10], to list a few) allow a user to securely search over encrypted data through keywords without first decrypting it, these techniques support only conventional *Boolean* keyword search[1], without capturing any relevance of the files in the search result. When directly applied in large collaborative data outsourcing cloud environment, they may suffer from the following two main drawbacks. On the one hand, for each search request, users without pre-knowledge of the encrypted cloud data have to go through every retrieved file in order to find ones most matching their interest, which demands possibly large amount of postprocessing overhead; On the other hand, invariably sending back all files solely based on presence/absence of the keyword further incurs large unnecessary network traffic, which is absolutely undesirable in today's pay-as-you-use cloud paradigm. In short, lacking of effective mechanisms to ensure the file retrieval accuracy is a significant drawback of existing searchable encryption schemes in the context of Cloud Computing. Nonetheless, the state-of-the-art in information retrieval (IR) community has already been utilizing various scoring mechanisms [11] to

[1]In the existing symmetric key based searchable encryption schemes, the support of disjunctive Boolean operation (OR) on multiple keywords searches still remains an open problem.

quantify and rank-order the relevance of files in response to any given search query. Although the importance of ranked search has received attention for a long history in the context of plaintext searching by IR community, surprisingly, it is still being overlooked and remains to be addressed in the context of encrypted data search.

Therefore, how to enable a searchable encryption system with support of secure ranked search, is the problem tackled in this paper. Our work is among the first few ones to explore ranked search over encrypted data in Cloud Computing. Ranked search greatly enhances system usability by returning the matching files in a ranked order regarding to certain relevance criteria (e.g., keyword frequency), thus making one step closer towards practical deployment of privacy-preserving data hosting services in the context of Cloud Computing. To achieve our design goals on both system security and usability, we propose to bring together the advance of both crypto and IR community to design the ranked searchable symmetric encryption scheme, in the spirit of "as-strong-as-possible" security guarantee. Specifically, we explore the statistical measure approach from IR and text-mining to embed weight information (i.e. relevance score) of each file during the establishment of searchable index before outsourcing the encrypted file collection. As directly outsourcing relevance scores will leak lots of sensitive frequency information against the keyword privacy, we then integrate a recent crypto primitive [12] order-preserving symmetric encryption (OPSE) and properly modify it for our purpose to protect those sensitive weight information, while providing efficient ranked search functionalities. Our contribution can be summarized as follows:

1) For the first time, we define the problem of secure ranked keyword search over encrypted cloud data, and provide such an effective protocol, which fulfills the secure ranked search functionality with little relevance score information leakage against keyword privacy.
2) Thorough security analysis shows that our ranked searchable symmetric encryption scheme indeed enjoys "as-strong-as-possible" security guarantee compared to previous SSE schemes.
3) Extensive experimental results demonstrate the effectiveness and efficiency of the proposed solution.

The rest of the paper is organized as follows. Section II gives the system and threat model, our design goals, notations and preliminaries. Then we provide the framework, definitions and basic scheme in Section III, followed by Section IV, which gives the detailed description of our ranked searchable symmetric encryption system. Section V and VI gives the security analysis and performance evaluation, respectively. Related work for both searchable encryption and secure result ranking is discussed in Section VII. Finally, Section VIII gives the concluding remark of the whole paper.

II. Problem Statement

A. The System and Threat Model

We consider a cloud data hosting service involving three different entities, as illustrated in Fig. 1: *data owner* (O),

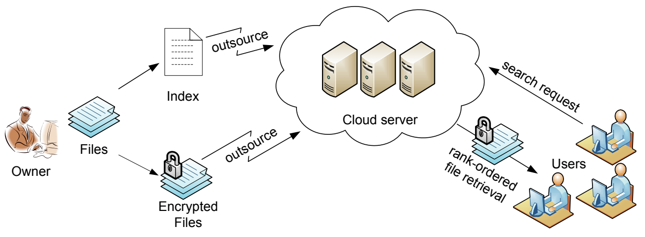

Fig. 1: Architecture of the search over encrypted cloud data

data user (U), and *cloud server* (CS). Data owner has a collection of n data files $\mathcal{C} = (F_1, F_2, \ldots, F_n)$ that he wants to outsource on the cloud server in encrypted form while still keeping the capability to search through them for effective data utilization reasons. To do so, before outsourcing, data owner will first build a secure searchable index $\mathcal{I}$ from a set of m distinct keywords $W = (w_1, w_2, ..., w_m)$ extracted[2] from the file collection $\mathcal{C}$, and store both the index $\mathcal{I}$ and the encrypted file collection $\mathcal{C}$ on the cloud server.

We assume the authorization between the data owner and users is appropriately done. To search the file collection for a given keyword w, an authorized user generates and submits a search request in a secret form—a trapdoor T_w of the keyword w—to the cloud server. Upon receiving the search request T_w, the cloud server is responsible to search the index $\mathcal{I}$ and return the corresponding set of files to the user. We consider the secure ranked keyword search problem as follows: the search result should be returned according to certain ranked relevance criteria (e.g., keyword frequency based scores, as will be introduced shortly), to improve file retrieval accuracy for users without prior knowledge on the file collection $\mathcal{C}$. However, cloud server should learn nothing or little about the relevance criteria themselves as they exhibit significant sensitive information against keyword privacy. To reduce bandwidth, the user may send an optional value k along with the trapdoor T_w and cloud server only sends back the top-k most relevant files to the user's interested keyword w.

We consider an "honest-but-curious" server in our model, which is consistent with most of the previous searchable encryption schemes. We assume the cloud server acts in an "honest" fashion and correctly follows the designated protocol specification, but is "curious" to infer and analyze the message flow received during the protocol so as to learn additional information. In other words, the cloud server has no intention to actively modify the message flow or disrupt any other kind of services.

B. Design Goals

To enable ranked searchable symmetric encryption for effective utilization of outsourced cloud data under the aforementioned model, our system design should achieve the following security and performance guarantee. Specifically, we have the

[2]To reduce the size of index, a list of standard IR techniques can be adopted, including case folding, stemming, and stop words etc. We omit this process of keyword extraction and refinement and refer readers to [5] for more details.

following goals: i) Ranked keyword search: to explore different mechanisms for designing effective ranked search schemes based on the existing searchable encryption framework; ii) Security guarantee: to prevent cloud server from learning the plaintext of either the data files or the searched keywords, and achieve the as-strong-as-possible security strength compared to the existing searchable encryption schemes; iii) Efficiency: above goals should be achieved with minimum communication and computation overhead.

C. Notation and Preliminaries

- $\mathcal{C}$ – the file collection to be outsourced, denoted as a set of n data files $\mathcal{C} = (F_1, F_2, \ldots, F_n)$.
- W – the distinct keywords extracted from file collection $\mathcal{C}$, denoted as a set of m words $W = (w_1, w_2, ..., w_m)$.
- $\mathrm{id}(F_j)$ – the identifier of file F_j that can help uniquely locate the actual file.
- $\mathcal{I}$ – the index built from the file collection, including a set of posting lists $\{\mathcal{I}(w_i)\}$, as introduced below.
- T_{w_i} – the trapdoor generated by a user as a search request of keyword w_i.
- $\mathcal{F}(w_i)$ – the set of identifiers of files in $\mathcal{C}$ that contain keyword w_i.
- N_i – the number of files containing the keyword w_i and $N_i = |\mathcal{F}(w_i)|$.

We now introduce some necessary information retrieval background for our proposed scheme:

Inverted Index In information retrieval, inverted index (also referred to as postings file) is a widely used indexing structure that stores a list of mappings from keywords to the corresponding set of files that contain this keyword, allowing full text search [11]. For ranked search purposes, the task of determining which files are most relevant is typically done by assigning a numerical score, which can be precomputed, to each file based on some ranking function introduced below. One example posting list of an index is shown in Fig. 2. We will use this inverted index structure to give our basic ranked searchable symmetric encryption construction.

Ranking Function In information retrieval, a ranking function is used to calculate relevance scores of matching files to a given search request. The most widely used statistical measurement for evaluating relevance score in the information retrieval community uses the TF × IDF rule, where TF (term frequency) is simply the number of times a given term or keyword (we will use them interchangeably hereafter) appears within a file (to measure the importance of the term within the particular file), and IDF (inverse document frequency) is obtained by dividing the number of files in the whole collection by the number of files containing the term (to measure the overall importance of the term within the whole collection). Among several hundred variations of the TF × IDF weighting scheme, no single combination of them outperforms any of the others universally [13]. Thus, without loss of generality, we choose an example formula that is commonly used and widely seen in the literature (see Chapter 4 in [5]) for the relevance score calculation in the following presentation.

Word	w_i				
File ID	F_{i_1}	F_{i_2}	F_{i_3}	$\cdots$	$F_{i_{N_i}}$
Relevance Score	6.52	2.29	13.42	4.76	13.80

Fig. 2: An example posting list of the inverted index.

Its definition is as follows:

$$Score(Q, F_d) = \sum_{t \in Q} \frac{1}{|F_d|} \cdot (1 + \ln f_{d,t}) \cdot \ln(1 + \frac{N}{f_t}). \quad (1)$$

Here Q denotes the searched keywords; $f_{d,t}$ denotes the TF of term t in file F_d; f_t denotes the number of files that contain term t; N denotes the total number of files in the collection; and $|F_d|$ is the length of file F_d, obtained by counting the number of indexed terms, functioning as the normalization factor.

III. The Definitions and Basic Scheme

In the introduction we motivated the ranked keyword search over encrypted data to achieve economies of scale for Cloud Computing. In this section, we start from the review of existing searchable symmetric encryption (SSE) schemes and provide the definitions and framework for our proposed ranked searchable symmetric encryption (RSSE). Note that by following the same security guarantee of existing SSE, it would be very inefficient to support ranked search functionality over encrypted data, as demonstrated in our basic scheme. The discussion of its demerits will lead to our proposed scheme.

A. Background on Searchable Symmetric Encryption

Searchable encryption allows data owner to outsource his data in an encrypted manner while maintaining the selectively-search capability over the encrypted data. Generally, searchable encryption can be achieved in its full functionality using an oblivious RAMs [14]. Although hiding everything during the search from a malicious server (including access pattern), utilizing oblivious RAM usually brings the cost of logarithmic number of interactions between the user and the server for each search request. Thus, in order to achieve more efficient solutions, almost all the existing work on searchable encryption literature resort to the weakened security guarantee, i.e., revealing the access pattern and search pattern but nothing else. Here access pattern refers to the outcome of the search result, i.e., which files have been retrieved. The search pattern includes the equality pattern among the two search requests (whether two searches were performed for the same keyword), and any information derived thereafter from this statement. We refer readers to [10] for the thorough discussion on SSE definitions.

Having a correct intuition on the security guarantee of existing SSE literature is very important for us to define our ranked searchable symmetric encryption problem. As later we will show that following the exactly same security guarantee of existing SSE scheme, it would be very inefficient to achieve ranked keyword search, which motivates us to further weaken

```
BuildIndex($K$,$\mathcal{C}$)
1. Initialization:
   i) scan $\mathcal{C}$ and extract the distinct words $W = (w_1, w_2, ..., w_m)$ from $\mathcal{C}$. For each $w_i \in W$, build $\mathcal{F}(w_i)$;
2. Build posting list:
   i) for each $w_i \in W$
      • for $1 \leq j \leq |\mathcal{F}(w_i)|$:
         a) calculate the score for file $F_{ij}$ according to equation 2, denoted as $S_{ij}$ ;
         b) compute $\mathcal{E}_z(S_{ij})$, and store it with $F_{ij}$'s identifier $\langle \mathrm{id}(F_{ij}) || \mathcal{E}_z(S_{ij}) \rangle$ in the posting list $\mathcal{I}(w_i)$;
3. Secure the index $\mathcal{I}$:
   i) for each $\mathcal{I}(w_i)$ where $1 \leq i \leq m$:
      • encrypt all $N_i$ entries with $\ell'$ padding $0'$s, $\langle 0^{\ell'} || \mathrm{id}(F_{ij}) || \mathcal{E}_z(S_{ij}) \rangle$, with key $f_y(w_i)$, where $1 \leq j \leq v$.
      • set remaining $\nu - N_i$ entries, if any, to random values of the same size as the existing $N_i$ entries of $\mathcal{I}(w_i)$.
      • replace $w_i$ with $\pi_x(w_i)$;
4. Output $\mathcal{I}$.
```

Fig. 3: The details of `BuildIndex(·)` for Basic Scheme

the security guarantee of existing SSE appropriately (leak the relative relevance order but not the relevance score) and realize an "as-strong-as-possible" ranked searchable symmetric encryption. Actually, this notion has been employed by cryptographers in many recent work [12], [15] where efficiency is preferred over security.

B. Definitions and Framework of RSSE System

We follow the similar framework of previously proposed searchable symmetric encryption scheme [10] and adapt the framework for our ranked searchable encryption system. A ranked searchable encryption scheme consists of four algorithms (`KeyGen`, `BuildIndex`, `TrapdoorGen`, `SearchIndex`). And our ranked searchable encryption system can be constructed from these four algorithms in two phases—`Setup` and `Retrieval`:

- `Setup`: The data owner initializes the public and secret parameters of the system by executing `KeyGen`, and pre-processes the data file collection $\mathcal{C}$ by using `BuildIndex` to generate the searchable index from the unique words extracted from $\mathcal{C}$. The owner then encrypts the data file collection $\mathcal{C}$, and publishes the index including the keyword frequency based relevance scores in some encrypted form, together with the encrypted collection $\mathcal{C}$ to the Cloud. As part of `Setup` phase, the data owner also needs to distribute the necessary secret parameters (in our case, the trapdoor generation key) to a group of authorized users by employing off-the-shelf public key cryptography or more efficient primitive such as broadcast encryption.
- `Retrieval`: The user uses `TrapdoorGen` to generate a secure trapdoor corresponding to his interested keyword, and submits it to the cloud server. Upon receiving the trapdoor, the cloud server will derive a list of matched file IDs and their corresponding encrypted relevance scores by searching the index via `SearchIndex`. The matched files should be sent back in a ranked sequence based on the relevance scores. However, the server should learn nothing or little beyond the order of the relevance scores.

Note that in our design, we focus on single keyword search. In this case, the IDF factor in equation 1 is always constant with regard to the given searched keyword. Thus, search results can be accurately ranked based only on the term frequency and file length information contained within the single file using equation 2:

$$Score(t, F_d) = \frac{1}{|F_d|} \cdot (1 + \ln f_{d,t}). \tag{2}$$

Data owner can keep a record of these two values and pre-calculate the relevance score, which introduces little overhead regarding to the index building. We will demonstrate this via experiments in the performance evaluation Section VI.

C. The Basic Scheme

Before giving our main result, we first start with a straightforward yet ideal scheme, where the security of our ranked searchable encryption is the same as previous SSE schemes, i.e., the user gets the ranked results without letting cloud server learn any additional information more than the access pattern and search pattern. However, this is achieved with the trade-off of efficiency, namely, either should the user wait for two round-trip time for each search request or he may even lose the capability to perform top-k retrieval, resulting the unnecessary communication overhead. The analysis of these demerits will lead to our main result. Note that the basic scheme we discuss here is tightly pertained to recent work [10], though our focus in on secure result ranking. Actually, it can be considered as the most simplified version of searchable symmetric encryption that satisfies the non-adaptive security definition of [10].

Basic Scheme: Let k, ℓ, ℓ', p be security parameters that will be used in `Keygen(·)`. Let $\mathcal{E}$ be a semantically secure symmetric encryption algorithm: $\mathcal{E} : \{0,1\}^{\ell} \times \{0,1\}^{r} \rightarrow \{0,1\}^{r}$.

Let ν be the maximum number of files containing word w_i, i.e., $\nu = \max_{i=1}^{m} N_i$. This value does not need to be known in advance for the instantiation of the scheme. Also, let f be a pseudo-random function and π be a collision resistant hash function with the following parameters:

- $f : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^\ell$
- $\pi : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^p$ where $p > \log m$

In practice, $\pi(\cdot)$ will be instantiated by off-the-self hash function like SHA-1, in which case p is 160 bits.

In the `Setup` phase:

1) The data owner initiates the scheme by calling `KeyGen(`$1^k, 1^\ell, 1^{\ell'}, 1^p$`)`, generates random keys $x, y \overset{R}{\longleftarrow} \{0,1\}^k$, $z \overset{R}{\longleftarrow} \{0,1\}^\ell$, and outputs $K = \{x, y, z, 1^\ell, 1^{\ell'}, 1^p\}$.
2) The data owner then builds a secure inverted index from the file collection $\mathcal{C}$ by calling `BuildIndex(`K,$\mathcal{C}$`)`. The details are given in Fig. 3. The ℓ' padding $0'$s indicate the valid posting entry.

In the `Retrieval` phase:

1) For an interested keyword w, the user generates a trapdoor $T = (\pi_x(w), f_y(w))$ by calling $\mathrm{TrapdoorGen}(w)$.
2) Upon receiving the trapdoor T_w, the server calls $\mathrm{SearchIndex}(\mathcal{I}, T_w)$: first locates the matching list of the index via $\pi_x(w)$, uses $f_y(w)$ to decrypt the entries, and then sends back the corresponding files according to $\mathcal{F}(w)$, together with their associated encrypted relevance scores.
3) User decrypts the relevance scores via key z and gets the ranked search results.

Discussion: The above scheme clearly satisfies the security guarantee of SSE, i.e., only the access pattern and search pattern is leaked. However, the ranking is done on the user side, which may bring in huge computation and post processing overhead. Moreover, sending back all the files consumes large undesirable bandwidth. One possible way to reduce the communication overhead is that server first sends back all the valid entries $\langle \mathrm{id}(F_{ij}) || \mathcal{E}_z(S_{ij}) \rangle$, where $1 \le j \le N_i$. User then decrypts the relevance score and sends cloud server another request to retrieve the most relevant files (top-k retrieval) by the rank-ordered decrypted scores. As the size of valid entries $\langle \mathrm{id}(F_{ij}) || \mathcal{E}_z(S_{ij}) \rangle$ is far less than the corresponding files, significant amount of bandwidth is expected to be saved, as long as user does not retrieve all the matching files. However, the most obvious disadvantage is the two round-trip time for each search request of every user. Also note that in this way, server still learns nothing about the value of relevance scores, but it knows the requested files are more relevant than the unrequested ones, which inevitably leaks more information than the access pattern and search pattern.

IV. Efficient Ranked Searchable Symmetric Encryption Scheme

The above straightforward approach demonstrates the core problem that causes the inefficiency of ranked searchable encryption. That is how to let server quickly perform the ranking

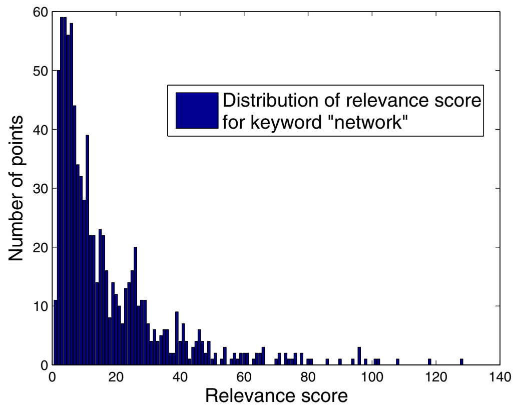

Fig. 4: An example of relevance score distribution.

without actually knowing the relevance scores. To effectively support ranked search over encrypted file collection, we now resort to the newly developed cryptographic primitive – order preserving symmetric encryption (OPSE) [12] to achieve more practical performance. Note that by resorting to OPSE, our security guarantee of RSSE is inherently weakened compared to SSE, as we now let server know the relevance order. However, this is the information we want to trade-off for efficient RSSE, as discussed in previous Section III. We will first briefly discuss the primitive of OPSE and its pros and cons. Then we show how we can adapt it to suit our purpose for ranked searchable encryption with an "as-strong-as-possible" security guarantee. Finally, we demonstrate how to choose different scheme parameters via concrete examples.

A. *Using Order Preserving Symmetric Encryption*

The OPSE is a deterministic encryption scheme where the numerical ordering of the plaintexts gets preserved by the encryption function. Boldyreva et al. [12] gives the first cryptographic study of OPSE primitive and provides a construction that is provably secure under the security framework of pseudorandom function or pseudorandom permutation. Namely, considering any order-preserving function $g(\cdot)$ from domain $\mathcal{D} = \{1, \ldots, M\}$ to range $\mathcal{R} = \{1, \ldots, N\}$ can be uniquely defined by a combination of M out of N ordered items, an OPSE is then said to be secure if and only if an adversary has to perform a brute force search over all the possible combinations of M out of N to break the encryption scheme. If the security level is chosen to be 80 bits, then it is suggested to choose $M = N/2 > 80$ so that the total number of combinations will be greater than 2^{80}. Their construction is based on an uncovered relationship between a random order-preserving function (which meets the above security notion) and the hypergeometric probability distribution, which will later be denoted as HGD. We refer readers to [12] for more details about OPSE and its security definition.

At the first glance, by changing the relevance score encryption from the standard indistinguishable symmetric encryption scheme to this OPSE, it seems to follow directly that efficient relevance score ranking can be achieved just like in the plaintext domain. However, as pointed out earlier, the OPSE is a deterministic encryption scheme. This inherent deterministic property, if not treated appropriately, will still leak a lot of information as any deterministic encryption scheme will do. Fig. 4 is an example of a skewed relevance score distribution of keyword "network", sampled from 1000 files of our test collection. For easy exposition, we encode the actual score into 128 levels in domain from 1 to 128. The score distribution can be seen as keyword specific from the slope, value range or other metrics [16]. Note that directly using OPSE will not randomize this keyword-specific score distribution nature. Therefore, with certain background information on the file collection, the adversary may reverse-engineer the keyword "network" directly from the encrypted score distribution without actually breaking the trapdoor construction, nor does the adversary need to break the OPSE.

B. Towards One-to-many Order-preserving Mapping

Therefore, we have to modify the OPSE to suit our purpose. In order to reduce the amount of information leakage from the deterministic property, an one-to-many OPSE scheme is thus desired, which can flatten or obfuscate the original relevance score distribution, increase its randomness, and still preserve the plaintext order. To do so, we first briefly review the encryption process of original deterministic OPSE, where a plaintext m in domain $\mathcal{D}$ is always mapped to the same random-sized non-overlapping interval bucket in range $\mathcal{R}$, determined by a keyed binary search over the range $\mathcal{R}$ and the result of a random HGD sampling function. A ciphertext c is then chosen within the bucket by using m as the seed for some random selection function.

Our one-to-many order-preserving mapping employs the random plaintext-to-bucket mapping of OPSE, but incorporates the unique file IDs together with the plaintext m as the random seed in the final ciphertext chosen process. Due to the use of unique file ID as part of random selection seed, the same plaintext m will no longer be deterministically assigned to the same ciphertext c, but instead a random value within the randomly assigned bucket in range $\mathcal{R}$. The whole process is shown in Algorithm 1, adapted from [12]. Here $\text{TapeGen}(\cdot)$ is a random coin generator and $\text{HYGEINV}(\cdot)$ is the efficient function implemented in MATLAB as our instance for the $\text{HGD}(\cdot)$ sampling function. The correctness of our one-to-many order-preserving mapping follows directly from the Algorithm 1. Note that our rational is to use the OPSE block cipher as a tool for different application scenarios and achieve better security, which is suggested by and consistent with [12]. Now, if we denote $\mathcal{OPM}$ as our one-to-many order-preserving mapping function with parameter: $\mathcal{OPM} : \{0,1\}^{\ell} \times \{0,1\}^{\log|\mathcal{D}|} \to \{0,1\}^{\log|\mathcal{R}|}$, our proposed RSSE scheme can be described as follows:

In the `Setup` phase:

1) The data owner calls `KeyGen(`$1^k, 1^\ell, 1^{\ell'}, 1^p, |\mathcal{D}|, |\mathcal{R}|$`)`, generates random keys $x, y, z \xleftarrow{R} \{0,1\}^k$, and output $K = \{x, y, z, 1^\ell, 1^{\ell'}, 1^p, |\mathcal{D}|, |\mathcal{R}|\}$.

Algorithm 1 One-to-many Order-preserving Mapping-$\mathcal{OPM}$

```
 1: procedure $\mathcal{OPM}_K(\mathcal{D}, \mathcal{R}, m, \text{id}(F))$
 2:     while $|\mathcal{D}|\ != 1$ do
 3:         $\{\mathcal{D}, \mathcal{R}\} \leftarrow \text{BinarySearch}(K, \mathcal{D}, \mathcal{R}, m)$;
 4:     end while
 5:     $coin \xleftarrow{R} \text{TapeGen}(K, (\mathcal{D}, \mathcal{R}, 1||m, \text{id}(F)))$
 6:     $c \xleftarrow{coin} \mathcal{R}$
 7:     Return $c$
 8: end procedure

 9: procedure $\text{BinarySearch}(K, \mathcal{D}, \mathcal{R}, m)$;
10:     $M \leftarrow |\mathcal{D}|$; $N \leftarrow |\mathcal{R}|$;
11:     $d \leftarrow \min(\mathcal{D}) - 1$; $r \leftarrow \min(\mathcal{R}) - 1$;
12:     $y \leftarrow r + \lceil N/2 \rceil$;
13:     $coin \xleftarrow{R} \text{TapeGen}(K, (\mathcal{D}, \mathcal{R}, 0||y))$
14:     $x \xleftarrow{R} d + \text{HYGEINV}(coin, M, N, y - r)$;
15:     if $m \le x$ then
16:         $\mathcal{D} \leftarrow \{d+1, \ldots, x\}$;
17:         $\mathcal{R} \leftarrow \{r+1, \ldots, y\}$;
18:     else
19:         $\mathcal{D} \leftarrow \{x+1, \ldots, d+M\}$;
20:         $\mathcal{R} \leftarrow \{y+1, \ldots, r+N\}$;
21:     end if
22:     Return $\{\mathcal{D}, \mathcal{R}\}$;
23: end procedure
```

2) The data owner calls `BuildIndex(`$K, \mathcal{C}$`)` to build the inverted index of collection $\mathcal{C}$, and uses $\mathcal{OPM}_{f_z(w_i)}(\cdot)$ instead of $\mathcal{E}(\cdot)$ to encrypt the scores.

In the `Retrieval` phase:

1) The user generates and sends a trapdoor $T_w = (\pi_x(w), f_y(w))$ for an interested keyword w. Upon receiving the trapdoor T_w, the cloud server first locates the matching entries of the index via $\pi_x(w)$, and then use $f_y(w)$ to decrypt the entry. These are the same with basic approach.
2) The cloud server now sees the file identifiers $\langle \text{id}(F_{ij}) \rangle$ (suppose $w = w_i$ and thus $j \in \{1, \ldots, N_i\}$) and their associated order-preserved encrypted scores: $\mathcal{OPM}_{f_z(w_i)}(S_{ij})$.
3) The server then fetches the files and sends back them in a ranked sequence according to the encrypted relevance scores $\{\mathcal{OPM}_{f_z(w_i)}(S_{ij})\}$, or sends top-$k$ most relevant files if the optional value k is provided.

Discussion: With the help of order-preserving mapping, now the server can accordingly rank the files as efficiently as for the unencrypted score. The reason that we use different keys ($f_z(w_i)$) to encrypt the relevance score for different posting lists is to make the one-to-many mapping more indistinguishable. Therefore, the same relevance score appearing in different lists of the index $\mathcal{I}$ will be mapped to different "bucket" in $\mathcal{R}$. Combining this with our one-to-many mapping will randomize the encrypted values from an overall point of view. Thus, we can further mitigate the useful information revealed

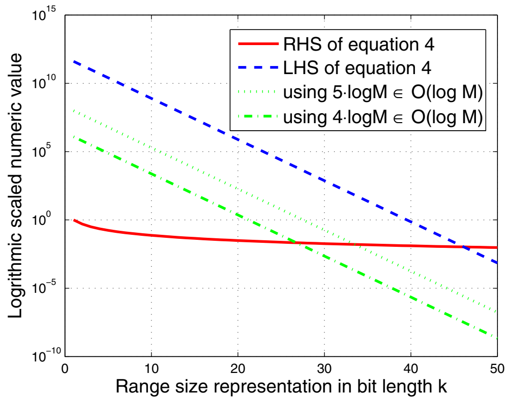

Fig. 5: Size selection of range $\mathcal{R}$, given $max/\lambda = 0.06$, $M = 128$, and $c = 1.1$. The LHS and RHS denote the corresponding side of the equation 4. Two example choices of $O(\log M)$ to replace $5\log M + 12$ in equation 4 are also included.

to the cloud server, who may be consistently interested at the statistical analysis on the encrypted numeric value to infer the underlying information.

C. Choosing Range Size of $\mathcal{R}$

We have highlighted our idea, but there still needs some care for implementation. Our purpose is to discard the peaky distribution of the plaintext domain as much as possible during the mapping, so as to eliminate the predictability of the keyword specific score distribution on the domain $\mathcal{D}$. Clearly, according to our random one-to-many order-preserving mapping (Algorithm 1 line 6), the larger size the range $\mathcal{R}$ is set, the less peaky feature will be preserved. However, the range size $|\mathcal{R}|$ cannot be arbitrarily large as it may slow down the efficiency of HGD function. Here, we use the min-entropy as our tool to find the size of range $\mathcal{R}$.

In information theory, the min-entropy of a discrete random variable $\mathcal{X}$ is defined as: $H_\infty(\mathcal{X}) = -\log(\max_a \Pr[\mathcal{X} = a])$. The higher $H_\infty(\mathcal{X})$ is, the more difficult the $\mathcal{X}$ can be predicted. We say $\mathcal{X}$ has high min-entropy if $H_\infty(\mathcal{X}) \in \omega(\log k)$ [15], where k is the bit length used to denote all the possible states of $\mathcal{X}$. Note that one possible choice of $H_\infty(\mathcal{X})$ is $(\log k)^c$ where $c > 1$.

Let max denote the maximum possible number of score duplicates within the index $\mathcal{I}$, and let λ denote the average number of scores to be mapped within each posting list $\mathcal{I}(w_i)$. Without loss of generality, we let $\mathcal{D} = \{1, \ldots, M\}$ and thus $|\mathcal{D}| = M$. Then based on above *high* min-entropy requirement, we can find the least possible $|\mathcal{R}|$ satisfying the following equation:

$$\frac{max/(|\mathcal{R}| \cdot \frac{1}{2}^{5\log M + 12})}{\lambda} \leq 2^{-(\log(\log|\mathcal{R}|))^c}. \qquad (3)$$

Here we use the result of [12] that the total recursive calls of HGD sampling during an OPSE operation is a function belonging to $O(\log M)$, and is at most $5\log M + 12$ on average, which is the expected number of times the range $\mathcal{R}$ will be cut into half during the function call of $\mathrm{BinarySearch}(\cdot)$. We also assume that the one-to-many mapping is truly random (Algorithm 1 line 5-6). Therefore, the numerator of left-hand-side of the above equation is indeed the expected largest number of duplicates after mapping. If we denote the range size $|\mathcal{R}|$ in bits, i.e., $k = \log|\mathcal{R}|$, we will have:

$$\frac{max \cdot 2^{5\log M + 12}}{2^k \cdot \lambda} = \frac{max \cdot M^5}{2^{k-12} \cdot \lambda} \leq 2^{-(\log k)^c}. \qquad (4)$$

With the established index $\mathcal{I}$, it is easy to determine the appropriate range size $|\mathcal{R}|$.

Following the same example of keyword "network" in Fig. 4, where $max/\lambda = 0.06$ (i.e., the max score duplicates is 60 and the average length of the posting list is 1000), one can determine the ciphertext range size $|\mathcal{R}| = 2^{46}$, when the relevance score domain is encoded as 128 different levels and c is set to be 1.1, as indicated in Fig. 5. Note that smaller size of range $|\mathcal{R}|$ is possible, when we replace the upper bound $5\log M + 12$ by other relatively "loose" function of M belonging to $O(\log M)$, e.g., $5\log M$ or $4\log M$. Fig. 5 shows that the range $|\mathcal{R}|$ size can be further reduced to 2^{34}, or 2^{27}, respectively. However, since the performance of our scheme for $|\mathcal{R}| = 2^{46}$ is already efficient enough (see Section VI), we have not tried to compare the overall performance and effectiveness of these different choices separately, and leave it as one of our future work.

V. Security Analysis

We evaluate the security of the proposed scheme by analyzing its fulfillment of the security guarantee described in Section II. Namely, the cloud server should not learn the plaintext of either the data files or the searched keywords. We start from the security analysis of our one-to-many order-preserving mapping. Then we analyze the security strength of the combination of one-to-many order-preserving mapping and SSE.

A. Security Analysis for One-to-many Mapping

Our one-to-many order-preserving mapping is adapted from the original OPSE, by introducing the file ID as the additional seed in the final ciphertext chosen process. Since such adaptation only functions at the final ciphertext selection process, it has nothing to do with the randomized plaintext-to-bucket mapping process in the original OPSE. In other words, the only effect of introducing file ID as the new seed is to make multiple plaintext duplicates m's no longer deterministically mapped to the same ciphertext c, but instead mapped to multiple random values within the assigned bucket in range $\mathcal{R}$. This helps flatten the ciphertext distribution to some extent after mapping. However, such a generic adaptation alone only works well when the number of plaintext duplicates are not large. In case there are many duplicates of plaintext m, its corresponding ciphertext distribution after mapping may still exhibit certain skewness or peaky feature of the plaintext distribution, due to the relative small size of assigned bucket from range $\mathcal{R}$.

This is why we propose to appropriately enlarge $\mathcal{R}$ in Section IV-C. Note that in the original OPSE, size $\mathcal{R}$ is determined

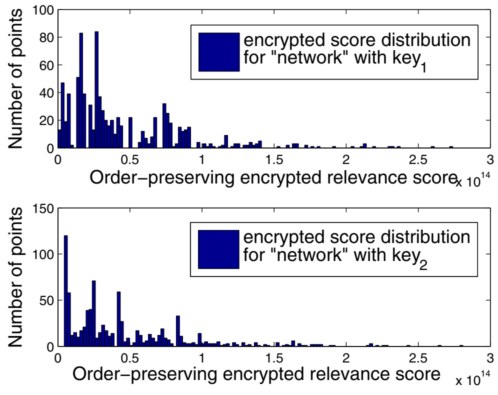

Fig. 6: Demonstration of effectiveness for one-to-many order-preserving mapping. The mapping is derived with the same relevance score set of keyword "network", but encrypted with two different random keys.

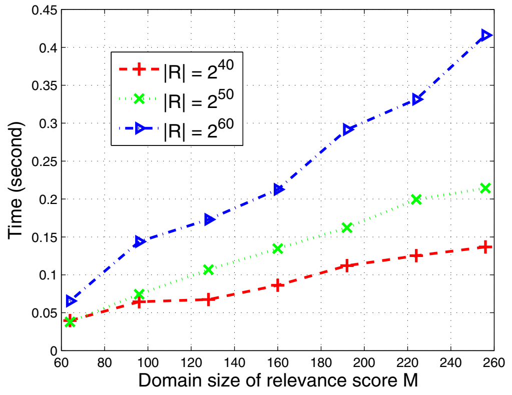

Fig. 7: The time cost of single one-to-many order-preserving mapping operation, with regarding to different choice of parameters: the domain size of relevance score M and the range size of encrypted score $|\mathcal{R}|$.

just to ensure the number of different combinations between $\mathcal{D}$ and $\mathcal{R}$ is larger than 2^{80}. But from a practical perspective, properly enlarging $\mathcal{R}$ in our one-to-many case further aims to ensure the low duplicates (with high probability) on the ciphertext range after mapping. This inherently increases the difficulty for adversary to tell precisely which points in the range R belong to the same score in the domain $\mathcal{D}$, making the order-preserving mapping as strong as possible. Note that one disadvantage of our scheme, compared to the original OPSE, is that fixing the range size $\mathcal{R}$ requires pre-knowledge on the percentage of maximum duplicates among all the plaintexts (i.e., max/λ in equation 3). However, such extra requirement can be easily met in our scenario when building the searchable index.

B. *Security Analysis for Ranked Keyword Search*

Compared to the original SSE, the new scheme embeds the encrypted relevance scores in the searchable index in addition to file ID. Thus the encrypted scores are the only additional information that the adversary can utilize against the security guarantee, i.e., keyword privacy and file confidentiality. Due to the security strength of the file encryption scheme, the file content is clearly well protected. Thus, we only need to focus on keyword privacy.

From previous discussion, we know that as long as data owner properly chooses the range size $\mathcal{R}$ sufficiently large, the encrypted scores in the searchable index will only be a sequence of order-preserved numeric values with very low duplicates. Though adversary may learn partial information from the duplicates (e.g., ciphertext duplicates may indicate very high corresponding plaintext duplicates), the fully randomized score-to-bucket assignment (inherited from OPSE) and the highly flattened one-to-many mapping still makes it difficult for the adversary to predict the original plaintext score distribution, let alone reverse engineer the keywords. Also note that we use different order-preserving encryption keys for different posting lists, which further reduces the information leakage from an overall point of view. Thus, the keyword privacy is also well preserved in our scheme.

VI. Performance Analysis

We conducted a thorough experimental evaluation of the proposed techniques on real data set: Request for comments database (RFC) [17]. At the time of writing, the RFC database contains 5563 plain text entries and totals about 277 MB. This file set contains a large number of technical keywords, many of which are unique to the files in which they are discussed. Our experiment is conducted using C programming language on a Linux machine with dual Intel Xeon CPU running at 3.0GHz. Algorithms use both openssl and MATLAB libraries. The performance of our scheme is evaluated regarding the effectiveness and efficiency of our proposed one-to-many order-preserving mapping, as well as the overall performance of our RSSE scheme, including the cost of index construction as well as the time necessary for searches.

A. *Effectiveness of One-to-many Order Preserving Mapping*

As indicated in Section IV-B, applying the proposed one-to-many mapping will further randomize the distribution of the encrypted values, which mitigates the chances of reverse-engineering the keywords by adversary. Fig. 6 demonstrates the effectiveness of our proposed scheme, where we choose $|\mathcal{R}| = 2^{46}$. The two figures shows the value distribution after one-to-many mapping with as input the same relevance score set of keyword "network", but encrypted with two different random keys. Note that due to our safe choice of $|\mathcal{R}|$ (see Section IV-C) and the relative small number of total scores per posting list (up to 1000), we do not have any duplicates after one-to-many order-preserving score mapping. However, for easy comparison purposes, the distribution in Fig. 6 is obtained with putting encrypted values into 128 equally spaced containers, as we do for the original score. Compared to previous Fig. 4, where the distribution of raw score is highly skewed, it can be seen that we indeed obtain

Number of files	Per keyword list size	Per keyword list build time
1000	12.414 KB	5.44s

TABLE I: Index construction overhead for 1000 RFC files.

two differently randomized value distribution. This is due to both the randomized score-to-bucket assignment inherited from the OPSE, and the one-to-many mapping. The former allows the same score mapped to different random-sized non-overlapping bucket, while the latter further obfuscates the score-to-ciphertext mapping accordingly. This confirms with our security analysis that the exposure of frequency information to the adversaries (the server in our case), utilized to reverse-engineer the keyword, can be further minimized.

B. *Efficiency of One-to-many Order-Preserving Mapping*

As shown in Section IV-C, the efficiency of our proposed one-to-many order-preserving mapping is determined by both the size of score domain M and the range $\mathcal{R}$. M affects how many rounds ($O(\log M)$) the procedure $\mathrm{BinarySearch}(\cdot)$ or $\mathrm{HGD}(\cdot)$ should be called. Meanwhile, M together with $\mathcal{R}$ both impact the time consumption for individual $\mathrm{HGD}(\cdot)$ cost. That's why the time cost of single one-to-many mapping order-preserving operation goes up faster than logarithmic, as M increases. Fig. 7 gives the efficiency measurement of our proposed scheme. The result represents the mean of 100 trials. Note that even for large range $\mathcal{R}$, the time cost of one successful mapping is still finished in 200 milliseconds, when M is set to be our choice 128. Specifically, for $|\mathcal{R}| = 2^{46}$, the time cost is less than 70 milliseconds. This is far more efficient than the order-preserving approach used in [16], [18], where [18] needs to keep lots of metadata to pre-build many different buckets on the data owner side, and [16] requires the pre-sampling and training of the relevance scores to be outsourced. However, our approach only needs the pre-generation of random keys.

C. *Performance of Overall RSSE System*

1) Index Construction: To allow for ranked keyword search, an ordinary inverted index attaches a relevance score to each posting entry. Our approach replaces the original scores with the ones after one-to-many order-preserving mapping. Specifically, it only introduces the mapping operation cost, additional bits to represent the encrypted scores, and overall entry encryption cost, compared to the original inverted index construction. Thus, we only list in Table I our index construction performance for a collection of 1000 RFC files. The index size and construction time listed were both per-keyword, meaning the posting list construction varies from one keyword to another. This was chosen as it removes the differences of various keyword set construction choices, allowing for a clean analysis of just the overall performance of the system. Note that the additional bits of encrypted scores is not a main issue due to the cheap storage cost on nowadays cloud servers. However, the one-to-many order-preserving mapping (about 70 ms per valid entries in the posting list) is a dominant factor for index construction time, as our raw-index only consumes 2.31s on average.

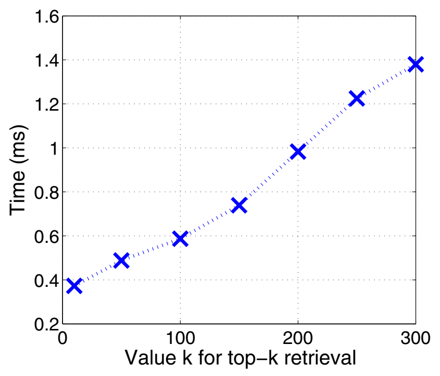

Fig. 8: The time cost for top-k retrieval.

2) Efficiency of Search: The search time includes fetching the posting list in the index, decrypting and rank-ordering each entries. Our focus is on top-k retrieval. As the encrypted scores are order-preserved, server can process the top-k retrieval almost as fast as in the plaintext domain. Note that the server does not have to traverse every posting list for each given trapdoor, but instead uses a tree-based data structure to fetch the corresponding list. Therefore, the overall search time cost is almost as efficient as on unencrypted data. Fig. 8 lists our search time cost against the increasing value of k, for the same index constructed above.

VII. Related Work

Searchable Encryption: Traditional searchable encryption [6]–[10] has been widely studied as a cryptographic primitive, with a focus on security definition formalizations and efficiency improvements. Song et al. [6] first introduced the notion of searchable encryption. They proposed a scheme in the symmetric key setting, where each word in the file is encrypted independently under a special two-layered encryption construction. Thus, a searching overhead is linear to the whole file collection length. Goh [7] developed a Bloom filter based per-file index, reducing the work load for each search request proportional to the number of files in the collection. Chang et al. [9] also developed a similar per-file index scheme. To further enhance search efficiency, Curtmola et al. [10] proposed a per-keyword based approach, where a single encrypted hash table index is built for the entire file collection, with each entry consisting of the trapdoor of a keyword and an encrypted set of related file identifiers. Searchable encryption has also been considered in the public-key setting. Boneh et al. [8] presented the first public-key based searchable encryption scheme, with an analogous scenario to that of [6]. In their construction, anyone with the public key can write to the data stored on the server but only authorized users with the private key can search. As an attempt to enrich query predicates, conjunctive keyword search over encrypted data have also been proposed in [19]–[21]. Very recently, aiming at tolerance of both minor typos and format inconsistencies in the user search input, fuzzy keyword search over encrypted cloud data has been proposed by Li. et al in [22]. Note that all these schemes support only boolean keyword search, and none of them support the ranked search problem which we are focusing in this paper.

Secure top-k retrieval from Database Community [16], [18] from database community are the most related work to our proposed RSSE. The idea of uniformly distributing posting elements using an order-preserving cryptographic function was first discussed in [18]. However, the order-preserving mapping function proposed in [18] does not support score dynamics, i.e., any insertion and updates of the scores in the index will result in the posting list completely rebuilt. [16] uses a different order-preserving mapping based on pre-sampling and training of the relevance scores to be outsourced, which is not as efficient as our proposed schemes. Besides, when scores following different distributions need to be inserted, their score transformation function still needs to be rebuilt. On the contrary, in our scheme the score dynamics can be gracefully handled, which is an important benefit inherited from the original OPSE. This can be observed from the $\mathrm{BinarySearch}(\cdot)$ procedure in Algorithm 1, where the same score will always be mapped to the same random-sized non-overlapping bucket, given the same encryption key. In other words, the newly changed scores will not affect previous mapped values. We note that supporting score dynamics, which can save quite a lot of computation overhead when file collection changes, is a significant advantage in our scheme. Moreover, both works above do not exhibit thorough security analysis which we do in the paper.

VIII. Concluding Remarks

In this paper, as an initial attempt, we motivate and solve the problem of supporting efficient ranked keyword search for achieving effective utilization of remotely stored encrypted data in Cloud Computing. We first give a basic scheme and show that by following the same existing searchable encryption framework, it is very inefficient to achieve ranked search. We then appropriately weaken the security guarantee, resort to the newly developed crypto primitive OPSE, and derive an efficient one-to-many order-preserving mapping function, which allows the effective RSSE to be designed. Through thorough security analysis, we show that our proposed solution is secure and privacy-preserving, while correctly realizing the goal of ranked keyword search. Extensive experimental results demonstrate the efficiency of our solution.

Following the current research, we propose several possible directions for future work on ranked keyword search over encrypted data. The most promising one is the support for multiple keywords. In this case, for the security requirement of searchable encryption, constructions for conjunctive keyword search in the existing literature [19]–[21] might be good candidates for our proposed ranked search. However, as the IDF factor now has to be included for score calculation, new approaches still need to be designed to completely preserve the order when summing up scores for all the provided keywords. Another interesting direction is to integrate advanced crypto techniques, such as attribute-based encryption to enable fine-grained access control in our multi-user settings.

Acknowledgement

This work was supported in part by the US National Science Foundation under grant CNS-0831963, CNS-0626601, CNS-0716306, and CNS-0831628. The authors would like to thank Nathan Chenette for helpful discussions. Thanks also to the anonymous reviewers and our shepherd Reza Curtmola for their suggestions in preparing the camera-ready version.

References

[1] P. Mell and T. Grance, "Draft nist working definition of cloud computing," Referenced on Jan. 23rd, 2010 Online at http://csrc.nist.gov/groups/SNS/cloud-computing/index.html, 2010.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," University of California, Berkeley, Tech. Rep. UCB-EECS-2009-28, Feb 2009.

[3] S. Kamara and K. Lauter, "Cryptographic cloud storage," in *Proceedings of Financial Cryptography: Workshop on Real-Life Cryptographic Protocols and Standardization 2010*, January 2010.

[4] Cloud Security Alliance, "Security guidance for critical areas of focus in cloud computing," 2009, http://www.cloudsecurityalliance.org.

[5] I. H. Witten, A. Moffat, and T. C. Bell, "Managing gigabytes: Compressing and indexing documents and images," Morgan Kaufmann Publishing, San Francisco, May 1999.

[6] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. of IEEE Symposium on Security and Privacy'00*, 2000.

[7] E.-J. Goh, "Secure indexes," Cryptology ePrint Archive, Report 2003/216, 2003, http://eprint.iacr.org/.

[8] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. of EUROCRYP'04, volume 3027 of LNCS*. Springer, 2004.

[9] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Proc. of ACNS'05*, 2005.

[10] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proc. of ACM CCS'06*, 2006.

[11] A. Singhal, "Modern information retrieval: A brief overview," *IEEE Data Engineering Bulletin*, vol. 24, no. 4, pp. 35–43, 2001.

[12] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-preserving symmetric encryption," in *Proceedings of Eurocrypt'09, volume 5479 of LNCS*. Springer, 2009.

[13] J. Zobel and A. Moffat, "Exploring the similarity space," *SIGIR Forum*, vol. 32, no. 1, pp. 18–34, 1998.

[14] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *Journal of the ACM*, vol. 43, no. 3, pp. 431–473, 1996.

[15] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in *Proceedings of Crypto'07, volume 4622 of LNCS*. Springer, 2007.

[16] S. Zerr, D. Olmedilla, W. Nejdl, and W. Siberski, "Zerber+r: Top-k retrieval from a confidential index," in *Proceedings of EDBT'09*, 2009.

[17] RFC, "Request For Comments Database," http://www.ietf.org/rfc.html.

[18] A. Swaminathan, Y. Mao, G.-M. Su, H. Gou, A. L. Varna, S. He, M. Wu, and D. W. Oard, "Confidentiality-preserving rank-ordered search," in *Proc. of the Workshop on Storage Security and Survivability*, 2007.

[19] P. Golle, J. Staddon, and B. R. Waters, "Secure Conjunctive Keyword Search over Encrypted Data," in *Proc. of ACNS'04*, 2004, pp. 31–45.

[20] L. Ballard, S. Kamara, and F. Monrose, "Achieving efficient conjunctive keyword searches over encrypted data," in *Proc. of ICICS'05*, 2005.

[21] Y. H. Hwang and P. J. Lee, "Public Key Encryption with Conjunctive Keyword Search and Its Extension to a Multi-User System," in *Proc. of Pairing'07*, 2007, pp. 31–45.

[22] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *Proc. of IEEE INFOCOM'10 Mini-Conference*, San Diego, CA, USA, March 2010.