Enabling Secure and Efficient Ranked Keyword Search over Outsourced Cloud Data

Cong Wang, *Student Member*, *IEEE*, Ning Cao, *Student Member*, *IEEE*, Kui Ren, *Senior Member*, *IEEE*, and Wenjing Lou, *Senior Member*, *IEEE*

Abstract—Cloud computing economically enables the paradigm of data service outsourcing. However, to protect data privacy, sensitive cloud data have to be encrypted before outsourced to the commercial public cloud, which makes effective data utilization service a very challenging task. Although traditional searchable encryption techniques allow users to securely search over encrypted data through keywords, they support only Boolean search and are not yet sufficient to meet the effective data utilization need that is inherently demanded by large number of users and huge amount of data files in cloud. In this paper, we define and solve the problem of secure ranked keyword search over encrypted cloud data. Ranked search greatly enhances system usability by enabling search result relevance ranking instead of sending undifferentiated results, and further ensures the file retrieval accuracy. Specifically, we explore the statistical measure approach, i.e., relevance score, from information retrieval to build a secure searchable index, and develop a one-to-many order-preserving mapping technique to properly protect those sensitive score information. The resulting design is able to facilitate efficient server-side ranking without losing keyword privacy. Thorough analysis shows that our proposed solution enjoys "as-strong-as-possible" security guarantee compared to previous searchable encryption schemes, while correctly realizing the goal of ranked keyword search. Extensive experimental results demonstrate the efficiency of the proposed solution.

Index Terms—Ranked search, searchable encryption, order-preserving mapping, confidential data, cloud computing.

1 Introduction

Cloud Computing is the long dreamed vision of computing as a utility, where cloud customers can 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 [2]. 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., [3].

As Cloud Computing becomes prevalent, more and more sensitive information are being centralized into the cloud, such as e-mails, personal health records, company finance data, and government documents, etc. The fact that data owners and cloud server are no longer in the same trusted domain may put the outsourced unencrypted data at risk [4], [33]: the cloud server may leak data information to unauthorized entities [5] or even be hacked [6]. It follows that sensitive data have to be encrypted prior to outsourcing for data privacy and combating unsolicited accesses. 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 [7]. 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., [8], [9], [10], [11], [12], 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 preknowledge 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

- *C. Wang is with the Department of Electrical and Computer Engineering, Illinois Institute of Technology, 3301 S Dearborn St., Siegel Hall 221A, Chicago, IL 60616. E-mail: cong@ece.iit.edu.*
- *N. Cao is with the Department of Electrical and Computer Engineering, Worcester Polytechnic Institute, 100 Institute Road, Worcester, MA 01609. E-mail: ncao@wpi.edu.*
- *K. Ren is with the Department of Electrical and Computer Engineering, Illinois Institute of Technology, 3301 S Dearborn St., Siegel Hall 319, Chicago, IL 60616. E-mail: kren@ece.iit.edu.*
- *W. Lou is with the Department of Computer Science, Virginia Polytechnic Institute and State University, 7054 Haycock Rd, Falls Church, VA 22043. E-mail: wjlou@vt.edu.*

Manuscript received 22 Mar. 2011; revised 4 Oct. 2011; accepted 17 Oct. 2011; published online 29 Nov. 2011.
Recommended for acceptance by I. Ahmad.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2011-03-0166.
Digital Object Identifier no. 10.1109/TPDS.2011.282.

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.

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 [13] to 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 toward 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 (RSSE) 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 [14] order-preserving symmetric encryption (OPSE) and properly modify it to develop a one-to-many order-preserving mapping technique 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 searchable symmetric encryption (SSE) schemes.
3. We investigate the practical considerations and enhancements of our ranked search mechanism, including the efficient support of relevance score dynamics, the authentication of ranked search results, and the reversibility of our proposed one-to-many order-preserving mapping technique.
4. Extensive experimental results demonstrate the effectiveness and efficiency of the proposed solution.

The rest of the paper is organized as follows: Section 2 gives the system and threat model, our design goals, notations, and preliminaries. Then, we provide the framework, definitions, and basic scheme in Section 3, followed by Section 4, which gives the detailed description of our ranked searchable symmetric encryption system. Section 5 gives the security analysis. Section 6 studies further enhancements and practical considerations, followed by Section 7 on performance evaluations. Related work for both searchable encryption and secure result ranking is discussed in Section 8. Finally, Section 9 gives the concluding remark of the whole paper.

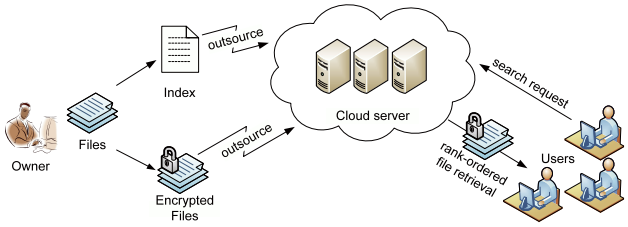

Fig. 1. Architecture for search over encrypted cloud data.

2 Problem Statement

2.1 The System and Threat Model

We consider an encrypted cloud data hosting service involving three different entities, as illustrated in Fig. 1: *data owner*, *data user*, and *cloud server*. 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 $W = (w_1, w_2, \ldots, w_m)$ 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 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 primarily 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

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 [7] for more details.

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. However, in some unexpected events, the cloud server may behave beyond the "honest-but-curious" model. We specifically deal with this scenario in Section 6.2.

2.2 Design Goals

To enable ranked searchable symmetric encryption for effective utilization of outsourced and encrypted cloud data under the aforementioned model, our system design should achieve the following security and performance guarantee. Specifically, we have the following goals: 1) Ranked keyword search: to explore different mechanisms for designing effective ranked search schemes based on the existing searchable encryption framework; 2) 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 existing searchable encryption schemes; 3) Efficiency: above goals should be achieved with minimum communication and computation overhead.

2.3 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 $W = (w_1, w_2, \ldots, w_m)$.
- $\mathrm{id}(F_j)$—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 [13]. 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 Table 1. 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 $\times$ IDF rule, where term frequency (TF) 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 inverse document frequency (IDF) 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 $\times$ IDF weighting scheme, no single combination of them outperforms any of the others universally [15]. Thus, without loss of generality, we choose an example formula that is commonly used and widely seen in the literature (see [7, Ch. 4]) for the relevance score calculation in the following presentation. Its definition is as follows:

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

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.

TABLE 1
An Example Posting List of the Inverted Index

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

3 The Definitions and Basic Scheme

In the introduction, we have 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 schemes and provide the definitions and framework for our proposed ranked searchable symmetric encryption. 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.

3.1 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 [16]. 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 works 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 [12] 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 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 [14], [17] where efficiency is preferred over security.

3.2 Definitions and Framework of RSSE System

We follow the similar framework of previously proposed searchable symmetric encryption schemes [12] and adapt the framework for our ranked searchable encryption system. A ranked searchable encryption scheme consists of four algorithms (`KeyGen`, `BuildIndex`, `TrapdoorGen`, `SearchIndex`). 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 as an initial attempt to investigate the secure ranked searchable encryption system, in this paper we focus on single keyword search. In this case, the IDF factor in (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

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

Data owner can keep a record of these two values and precalculate the relevance score, which introduces little overhead regarding to the index building. We will demonstrate this via experiments in the performance evaluation Section 7.

3.3 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 tradeoff 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. We believe 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 [12], though our focus is 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 [12].

Basic scheme. Let k, ℓ, ℓ', p be security parameters that will be used in `Keygen`$(\cdot)$. 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 some keyword $w_i \in W$ for $i = 1, \ldots, m$, 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 pseudorandom function and π be a collision resistant hash function with the following parameters:

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

In practice, $\pi(\cdot)$ will be instantiated by off-the-shelf hash functions 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, $y \xleftarrow{R} \{0,1\}^{k}$, $z \xleftarrow{R} \{0,1\}^{\ell}$, 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 Table 2. 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,

TABLE 2
The Details of BuildIndex(·) for Basic Scheme

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 ℓ' padding 0's, $\langle 0^{\ell'} || \mathrm{id}(F_{ij}) || \mathcal{E}_z(S_{ij}) \rangle$, with key $f_y(w_i)$, where $1 \leq j \leq \nu$.
   • 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 $\mathcal{I}$.

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 are leaked. However, the ranking is done on the user side, which may bring in huge 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 $1 \leq j \leq N_i$. 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.

4 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 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 [14] 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 3. 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.

4.1 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. [14] 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 that 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 [14] 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. One such information leakage is the plaintext distribution. Take Fig. 2, for example, which shows a skewed relevance score distribution of keyword "network," sampled from 1,000 files of our test collection. For easy exposition, we encode the actual score into 128 levels in domain from 1 to 128. Due to the deterministic

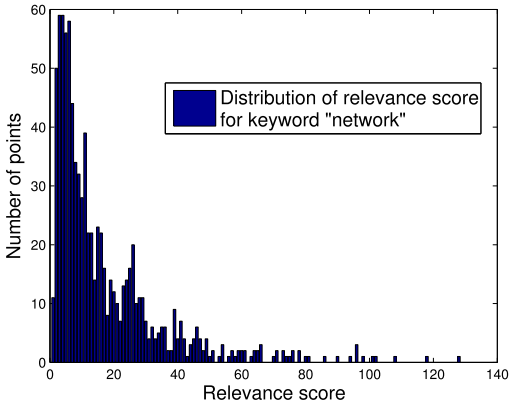

Fig. 2. An example of relevance score distribution.

property, if we use OPSE directly over these sampled relevance scores, the resulting ciphertext shall share exactly the same distribution as the relevance score in Fig. 2. On the other hand, previous research works [18], [22] have shown that the score distribution can be seen as keyword specific. Specifically, in [22], the authors have shown that the TF distribution of certain keywords from the Enron e-mail corpus[3] can be very peaky, and thus result in significant information leak for the corresponding keyword. In [18], the authors further point out that the TF distribution of the keyword in a given file collection usually follows a power law distribution, regardless of the popularity of the keyword. Their results on a few test file collections show that not only different keywords can be differentiated by the slope and value range of their TF distribution, but even the normalized TF distributions, i.e., the original score distributions (see (2)), can be keyword specific. Thus, with certain background information on the file collection, such as knowing it contains only technical research papers, the adversary may be able to 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.

4.2 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 nonoverlapping 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 [14]. Here, $\mathrm{TapeGen}(\cdot)$ is a random coin generator and $\mathrm{HYGEINV}(\cdot)$ is the efficient function implemented in Matlab as our instance for the $\mathrm{HGD}(\cdot)$ 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 [14]. Now, if we denote $\mathcal{OPM}$ as our one-to-many order-preserving mapping function with parameter: $\mathcal{OPM} : \{0,1\}^{\ell} \times \{0,1\}^{\log|\mathcal{D}|} \rightarrow \{0,1\}^{\log|\mathcal{R}|}$, our proposed RSSE scheme can be described as follows:

In the `Setup` phase

1. The data owner calls $\texttt{KeyGen}(1^k, 1^{\ell}, 1^{\ell'}, 1^p, |\mathcal{D}|, |\mathcal{R}|)$, generates random keys $x, y, z \xleftarrow{R} \{0,1\}^k$, and outputs $K = \{x, y, z, 1^{\ell}, 1^{\ell'}, 1^p, |\mathcal{D}|, |\mathcal{R}|\}$.
2. The data owner calls $\texttt{BuildIndex}(K, \mathcal{C})$ 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 uses $f_y(w)$ to decrypt the entry. These are the same with basic approach.
2. The cloud server now sees the file identifiers $\langle \mathrm{id}(F_{ij}) \rangle$ (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.

Algorithm 1. One-to-Many Order-Preserving Mapping-$\mathcal{OPM}$

procedure $\mathcal{OPM}_K(\mathcal{D}, \mathcal{R}, m, \mathrm{id}(F))$
  while $|\mathcal{D}|! = 1$ **do**
    $\{\mathcal{D}, \mathcal{R}\} \leftarrow \mathrm{BinarySearch}(K, \mathcal{D}, \mathcal{R}, m)$;
  end while
  $coin \xleftarrow{R} \mathrm{TapeGen}(K, (\mathcal{D}, \mathcal{R}, 1 \| m, \mathrm{id}(F)))$;
  $c \xleftarrow{coin} \mathcal{R}$;
  return c;
end procedure

procedure $\mathrm{BinarySearch}(K, \mathcal{D}, \mathcal{R}, m)$;
  $M \leftarrow |\mathcal{D}|$; $N \leftarrow |\mathcal{R}|$;
  $d \leftarrow \min(\mathcal{D}) - 1$; $r \leftarrow \min(\mathcal{R}) - 1$;
  $y \leftarrow r + \lceil N/2 \rceil$;
  $coin \xleftarrow{R} \mathrm{TapeGen}(K, (\mathcal{D}, \mathcal{R}, 0 \| y))$;
  $x \xleftarrow{R} d + \mathrm{HYGEINV}(coin, M, N, y - r)$;
  if $m \leq x$ **then**
    $\mathcal{D} \leftarrow \{d + 1, \ldots, x\}$;
    $\mathcal{R} \leftarrow \{r + 1, \ldots, y\}$;
  else

3. http://www.cs.cmu.edu/~enron/.

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**

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

4.3 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 $H_\infty(\mathcal{X}) \in \omega(\log k)$ [17], 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 $c > 1$. Based on this high min-entropy requirement as a guideline, we aim to find appropriate size of the range $\mathcal{R}$, which not only helps discard the peaky feature of the plaintext score distribution after score encryption, but also maintains within relatively small size so as to ensure the order-preserving mapping efficiency.

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:

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

Here, we use the result of [14] 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,

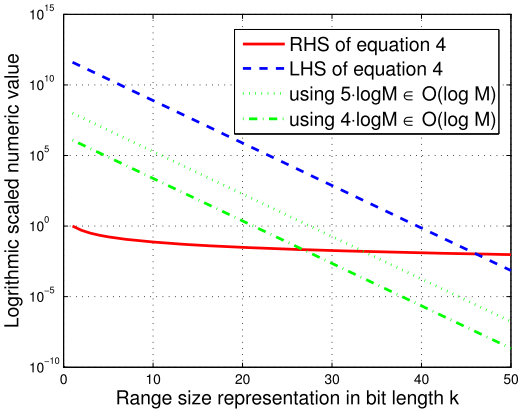

Fig. 3. Size selection of range $\mathcal{R}$, given $max/\lambda = 0.06$, $M = 128$, and $c = 1.1$. The LHS and RHS denote the corresponding side of the (4). Two example choices of $O(\log M)$ to replace $5 \log M + 12$ in (4) are also included.

which is the expected number of times the range $\mathcal{R}$ will be cut into half during the function call of $\text{BinarySearch}(\cdot)$. 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. Dividing the numerator by λ, we have on the left-hand side the expected largest probability of a plaintext score mapped to a given encrypted value in range $\mathcal{R}$. If we denote the range size $|\mathcal{R}|$ in bits, i.e., $k = \log |\mathcal{R}|$, we can rewrite the above inequation as

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

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. 2, where $max/\lambda = 0.06$ (i.e., the max score duplicates is 60 and the average length of the posting list is 1,000), 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. 3. 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. 3 shows that the range $|\mathcal{R}|$ size can be further reduced to 2^{34}, or 2^{27}, respectively. In Section 7, we provide detailed experimental results and analysis on the performance and effectiveness of these different parameter selections.

5 Security Analysis

We evaluate the security of the proposed scheme by analyzing its fulfillment of the security guarantee described in Section 2. 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.

5.1 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 selected from range $\mathcal{R}$.

This is why we propose to appropriately enlarge $\mathcal{R}$ in Section 4.3. Note that in the original OPSE, size $\mathcal{R}$ is determined 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 preknowledge on the percentage of maximum duplicates among all the plaintexts (i.e., max/λ in (3)). However, such extra requirement can be easily met in our scenario when building the searchable index.

5.2 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.

6 Further Enhancements and Investigations

Above discussions have shown how to achieve an efficient RSSE system. In this section, we give further study on how to make the RSSE system more readily deployable in practice. We start with some practical considerations on the index update and show how our mechanism can gracefully handle the case of score dynamics without introducing recomputation overhead on data owners. For enhanced quality of service assurance, we next study how the RSSE system can support ranked search result authentication. Finally, we uncover the reversible property of our one-to-many order-preserving mapping, which may find independent use in other interesting application scenarios.

6.1 Supporting Score Dynamics

In Cloud Computing, outsourced file collection might not only be accessed but also updated frequently for various application purposes (see [19], [20], [21], for example). Hence, supporting the score dynamics in the searchable index for an RSSE system, which is reflected from the corresponding file collection updates, is thus of practical importance. Here, we consider score dynamics as adding newly encrypted scores for newly created files, or modifying old encrypted scores for modification of existing files in the file collection. Ideally, given a posting list in the inverted index, the encryption of all these newly changed scores should be incorporated directly without affecting the order of all other previously encrypted scores, and we show that our proposed one-to-many order-preserving mapping does exactly that. Note that we do not consider file deletion scenarios because it is not hard to infer that deleting any file and its score does not affect the ranking orders of the remaining files in the searchable index.

This graceful property of supporting score dynamics is inherited from the original OPSE scheme, even though we made some adaptations in the mapping process. 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 nonoverlapping bucket, given the same encryption key and the same parameters of the plaintext domain $\mathcal{D}$ and ciphertext range $\mathcal{R}$. Because the buckets themselves are nonoverlapping, the newly changed scores indeed do not affect previously mapped values. Thus, with this property, the data owner can avoid the recomputation of the whole score encryption for all the file collection, but instead just handle those changed scores whenever necessary. Note that the scores chosen from the same bucket are treated as ties and their order can be set arbitrarily.

Supporting score dynamics is also the reason why we do not use the naive approach for RSSE, where data owner arranges file IDs in the posting list according to relevance score before outsourcing. As whenever the file collection changes, the whole process, including the score calculation, would need to be repeated, rendering it impractical in case of frequent file collection updates. In fact, supporting score dynamics will save quite a lot of computation overhead during the index update, and can be considered as a significant advantage compared to the related work [18], [22], as will be discussed in Section 8.

6.2 Authenticating Ranked Search Result

In practice, cloud servers may sometimes behave beyond the semihonest model. This can happen either because cloud server intentionally wants to do so for saving cost when handling large number of search requests, or there may be software bugs, or internal/external attacks. Thus, enabling a search result authentication mechanism that can detect such unexpected behaviors of cloud server is also of practical interest and worth further investigation.

To authenticate a ranked search result (or Top-k retrieval), one need to ensure: 1) the retrieved results are the most relevant ones; 2) the relevance sequence among the results are not disrupted. To achieve this two authentication requirements, we propose to utilize the one way hash chain technique, which can be added directly on top of the previous RSSE design. Let $H(\cdot)$ denote some cryptographic one-way hash function, such as SHA-1. Our mechanism requires one more secret value u in the `Setup` phase to be generated and shared between data owner and users. The details go as follows:

In the `Setup` phase

1. When data owner calls `BuildIndex`$(K, \mathcal{C})$, he picks an initial seed $s_0^i = f_u(w_i)$ for each posting list of keyword $w_i \in W$. Then, he sorts the posting list based on the encrypted scores.
2. Suppose $\mathrm{id}(F_{i1}), \mathrm{id}(F_{i2}), \ldots, \mathrm{id}(F_{iv})$ denotes the ordered sequence of file identifiers based on the encrypted relevance scores. The data owner generates a hash chain

$$H_1^i = H(\mathrm{id}(F_{i1}) \| s_0^i), H_2^i = H(\mathrm{id}(F_{i2}) \| H_1^i), \ldots,$$
$$H_v^i = H(\mathrm{id}(F_{iv}) \| H_{v-1}^i).$$

3. For each corresponding entry $\langle 0^{\ell'} \| \mathrm{id}(F_{ij}) \| \mathcal{E}_z(S_{ij}) \rangle$, $0 \le j \le v$, in the posting list of keyword w_i, the data owner inserts the corresponding hash value of the hash chain and gets $\langle 0^{\ell'} \| \mathrm{id}(F_{ij}) \| H_j^i \| \mathcal{E}_z(S_{ij}) \rangle$. All other operations, like entry encryption and entry permutation remain the same as previous RSSE scheme.

In the `Retrieval` phase

1. Whenever the cloud server is transmitting back top-k most relevant files, the corresponding k hash values embedded in the posting list entries should also be sent back as a correctness proof.
2. The user simply generates the initial seed $s_0^i = f_u(w_i)$ and verifies the received portion of the hash chain accordingly.

Discussion. It is easy to see that the secret seed shared between data owner and user ensures the authentication requirement 1, while the one-way property of hash chain guarantees the authentication requirement 2. The hash chain itself is a lightweight technique, which can be easily incorporated in our RSSE system with negligible computation/performance overhead on both data owner and users. The only tradeoff for achieving this high quality of data search assurance is the storage overhead on cloud, due to the augmented size of posting list. But we believe this should be easily acceptable because of the cheap storage cost today.

6.3 Reversing One-to-Many Order-Preserving Mapping

For any order-preserving mapping process, being reversible is very useful in many practical situations, especially when the underlying plaintext values need to be modified or utilized for further computation purposes. While OPSE, designed as a block cipher, by default has this property, it is not yet clear whether our one-to-many order-preserving mapping can be reversible too. In the following, we give a positive answer to this question.

Again, the reversibility of the proposed one-to-many order-preserving mapping can be observed from the $\mathrm{BinarySearch}(\cdot)$ procedure in Algorithm 1. The intuition is that the plaintext-to-bucket mapping process of OPSE is reversible. Namely, as long as the ciphertext is chosen from the certain bucket, one can always find through the $\mathrm{BinarySearch}(\cdot)$ procedure to uniquely identify the plaintext value, thus making the mapping reversible. For completeness, we give the details in Algorithm 2, which again we acknowledge that is adapted from [14]. The corresponding reversed mapping performance is reported in Section 7.2.

Algorithm 2. Reversing One-to-many Order-preserving Mapping-$\mathcal{ROPM}$

procedure $\mathcal{OPM}_K(\mathcal{D}, \mathcal{R}, c, \mathrm{id}(F))$
  while $|\mathcal{D}| ! = 1$ **do**
    $\{\mathcal{D}, \mathcal{R}\} \leftarrow \mathrm{BinarySearch}(K, \mathcal{D}, \mathcal{R}, c)$;
  end while
  $m \leftarrow \min(\mathcal{D})$;
  $coin \xleftarrow{R} \mathrm{TapeGen}(K, (\mathcal{D}, \mathcal{R}, 1 \| m, \mathrm{id}(F)))$;
  $w \xleftarrow{coin} \mathcal{R}$;
  if $w = c$ **then** return m;
  end if
  return $\perp$;
end procedure

procedure $\mathrm{BinarySearch}(K, \mathcal{D}, \mathcal{R}, c)$;
  $M \leftarrow |\mathcal{D}|$; $N \leftarrow |\mathcal{R}|$;
  $d \leftarrow \min(\mathcal{D}) - 1$; $r \leftarrow \min(\mathcal{R}) - 1$;
  $y \leftarrow r + \lceil N/2 \rceil$;
  $coin \xleftarrow{R} \mathrm{TapeGen}(K, (\mathcal{D}, \mathcal{R}, 0 \| y))$;
  $x \xleftarrow{R} d + \mathrm{HYGEINV}(coin, M, N, y - r)$;
  if $c \le y$ **then**
    $\mathcal{D} \leftarrow \{d + 1, \ldots, x\}$;
    $\mathcal{R} \leftarrow \{r + 1, \ldots, y\}$;
  else
    $\mathcal{D} \leftarrow \{x + 1, \ldots, d + M\}$;
    $\mathcal{R} \leftarrow \{y + 1, \ldots, r + N\}$;
  end if
  return $\{\mathcal{D}, \mathcal{R}\}$;
end procedure

7 Performance Analysis

We conducted a thorough experimental evaluation of the proposed techniques on real data set: Request for comments (RFC) database [23]. At the time of writing, the RFC database contains 5,563 plain text entries and totals about 277 MB. This file set contains a large number of technical

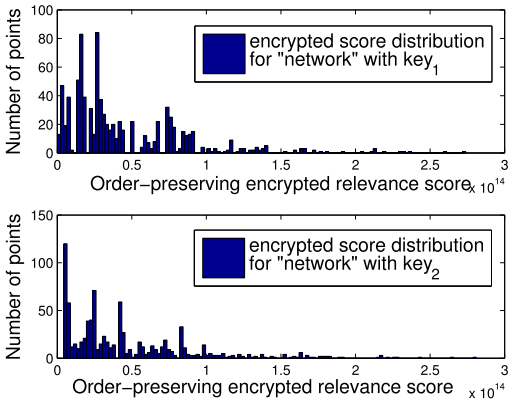

Fig. 4. Demonstration of effectiveness for one-to-many order-preserving mapping. The mapping is derived with the same relevance score set of keyword "network," but encrypted with two different random keys.

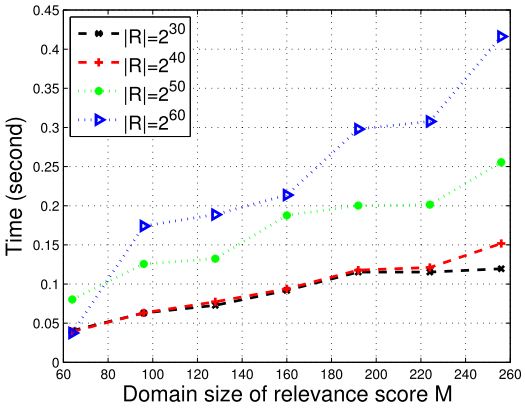

Fig. 5. The time cost of single one-to-many order-preserving mapping operation, with regarding to different choice of parameters: the domain size of relevance score M and the range size of encrypted score $|\mathcal{R}|$.

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 3.0 GHz. 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. Note that though we use a single server in the experiment, in practice we can separately store the searchable index and the file collections on different virtualized service nodes in the commercial public cloud, such as the Amazon EC2 and Amazon S3, respectively. In that way, even if data owners choose to store their file collection in different geographic locations for increased availability, the underlying search mechanism, which always takes place based on the searchable index, will not be affected at all.

7.1 Effectiveness of One-to-Many Order Preserving Mapping

As indicated in Section 4.2, 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. 4 demonstrates the effectiveness of our proposed scheme, where we choose $|\mathcal{R}| = 2^{46}$. The two figures show 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 4.3) and the relative small number of total scores per posting list (up to 1,000), we do not have any duplicates after one-to-many order-preserving score mapping. However, for easy comparison purposes, the distribution in Fig. 4 is obtained with putting encrypted values into 128 equally spaced containers, as we do for the original score. Compared to previous Fig. 2, where the distribution of raw score is highly skewed, it can be seen that we indeed obtain 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 nonoverlapping 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.

7.2 Efficiency of One-to-Many Order-Preserving Mapping

As shown in Section 4.3, 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. 5 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 $|\mathcal{R}| = 2^{40}$, the time cost is less than 70 milliseconds. This is far more efficient than the order-preserving approach used in [18], [22], where [22] needs to keep lots of metadata to prebuild many different buckets on the data owner side, and [18] requires the presampling and training of the relevance scores to be outsourced. However, our approach only needs the pregeneration of random keys.

As shown in Section 6.3, our one-to-many order-preserving mapping is in fact a reversible process. For completeness, the corresponding reverse mapping performance results are reported in Fig. 6. Compared to Fig. 5, the reverse mapping process is almost as fast as the original mapping process. This can be explained from the Algorithms 1 and 2 of the two approaches. Namely, for the same prefixed system parameters, both processes share the same number of recursive calls for the $\mathrm{BinarySearch}(\cdot)$ procedure, thus resulting the similar performance.

7.3 Performance of Overall RSSE System

7.3.1 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

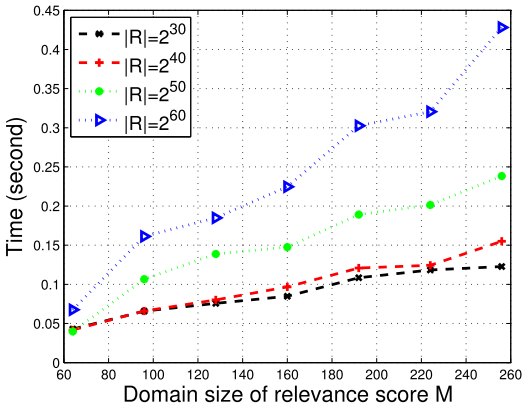

Fig. 6. The time cost of single reverse one-to-many order-preserving mapping operation, with regarding to different choice of parameters: the domain size of relevance score M and the range size of encrypted score $|\mathcal{R}|$.

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 3 our index construction performance for a collection of 1,000 RFC files. The index size and construction time listed were both perkeyword, 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.

Our experiment shows the total per list building time is 5.44 s, while the raw-index only consumes 2.31 s on average. Here, the raw-index construction corresponds to the steps 1 and 2 of the `BuildIndex` Algorithm in Table 2, which includes the plaintext score calculations and the inverted index construction but without considering security. To have a better understanding of the extra overhead introduced by RSSE, we also conducted an experiment for the basic searchable encryption scheme that supports only single keyword-based Boolean search. The implementation is based on the algorithm in Table 2, excluding the score calculations and the corresponding score encryptions. Building such a searchable index for secure Boolean search costs 1.88 s per posting list. In both comparisons, we conclude that the score encryption via proposed one-to-many order-preserving mapping is the dominant factor for index construction time, which costs about 70 ms per valid entries in the posting list. However, given that the index construction is the one-time cost before outsourcing and the enabled secure server side ranking functionality significantly improves subsequent file retrieval accuracy and efficiency, we consider the overhead introduced is reasonably acceptable.

TABLE 3
Per Keyword Index Construction Overhead for 1,000 RFC Files

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

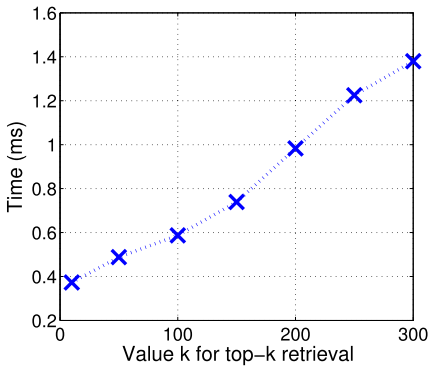

Fig. 7. The time cost for top-k retrieval.

Please note that our current implementation is not fully optimized. Further improvement on the implementation efficiency can be expected and is one of our important future work.

7.3.2 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. 7 list our search time cost against the value of k increases, for the same index constructed above.

8 Related Work

Searchable encryption. Traditional searchable encryption [8], [9], [10], [11], [12], [24], [25] has been widely studied as a cryptographic primitive, with a focus on security definition formalizations and efficiency improvements. Song et al. [8] 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 [9] developed a Bloom filter-based per-file index, reducing the workload for each search request proportional to the number of files in the collection. Chang and Mitzenmacher [11] also developed a similar per-file index scheme. To further enhance search efficiency, Curtmola et al. [12] 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. [10] presented the first public-key-based searchable encryption scheme, with an analogous scenario to that of [8]. 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 [26], [27], [28]. 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 [29]. Very recently, a privacy-assured similarity search mechanism over outsourced cloud data has been explored by Wang et al. in [34]. Note that all these schemes support only Boolean keyword search, and none of them support the ranked search problem which we are focusing on in this paper.

Following our research on secure ranked search over encrypted data, very recently, Cao et al. [30] propose a privacy-preserving multikeyword ranked search scheme, which extends our previous work in [1] with support of multikeyword query. They choose the principle of "coordinate matching," i.e., as many matches as possible, to capture the similarity between a multikeyword search query and data documents, and later quantitatively formalize the principle by a secure inner product computation mechanism. One disadvantage of the scheme is that cloud server has to linearly traverse the whole index of all the documents for each search request, while ours is as efficient as existing SSE schemes with only constant search cost on cloud server.

Secure top-k retrieval from Database Community [18], [22] 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 [22]. However, the order-preserving mapping function proposed in [22] 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. Zerr et al. [18] use a different order-preserving mapping based on presampling 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 nonoverlapping 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.

Other related techniques. Allowing range queries over encrypted data in the public key settings has been studied in [31], [32], where advanced privacy-preserving schemes were proposed to allow more sophisticated multiattribute search over encrypted files while preserving the attributes' privacy. Though these two schemes provide provably strong security, they are generally not efficient in our settings, as for a single search request, a full scan and expensive computation over the whole encrypted scores corresponding to the keyword posting list are required. Moreover, the two schemes do not support the ordered result listing on the server side. Thus, they cannot be effectively utilized in our scheme since the user still does not know which retrieved files would be the most relevant.

Difference from conference version. Portions of the work presented in this paper have previously appeared as an extended abstract in [1]. We have revised the paper a lot and improved many technical details as compared to [1]. The primary improvements are as follows: First, we provide new Section 6.1 to study and address some practical considerations of the RSSE design. Second, we provide Section 6.2 to thoroughly study the result completeness authentication. The mechanism design incur negligible overhead on data users, and further enhances the quality of data search service. Third, we extend our previous result on order-preserving one-to-many mapping and show in Section 6.3 that this mapping process is indeed reversible, which can be very useful in many practical applications. For completeness, we provide the corresponding algorithm for the reverse mapping and also include its performance results. Finally, the related work has been substantially improved, which now faithfully reflects many recent advancements on privacy-preserving search over encrypted data.

9 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. We also investigate some further enhancements of our ranked search mechanism, including the efficient support of relevance score dynamics, the authentication of ranked search results, and the reversibility of our proposed one-to-many order-preserving mapping technique. 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.

Acknowledgments

This work was supported in part by the US National Science Foundation (NSF) under grants CNS-1054317, CNS-1116939, CNS-1156318, and CNS-1117111, and by Amazon web service research grant. The authors would like to thank Nathan Chenette for helpful discussions. Thanks also to Reza Curtmola for valuable suggestions in preparing the earlier version of the manuscript. A preliminary version [1] of this paper was presented at the 30th International Conference on Distributed Computing Systems (ICDCS '10).

References

[1] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure Ranked Keyword Search over Encrypted Cloud Data," *Proc. IEEE 30th Int'l Conf. Distributed Computing Systems (ICDCS '10)*, 2010.

[2] P. Mell and T. Grance, "Draft Nist Working Definition of Cloud Computing," http://csrc.nist.gov/groups/SNS/cloud-computing/index.html, Jan. 2010.

[3] 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," Technical Report UCB-EECS-2009-28, Univ. of California, Berkeley, Feb. 2009.

[4] Cloud Security Alliance "Security Guidance for Critical Areas of Focus in Cloud Computing," http://www.cloudsecurityalliance.org, 2009.

[5] Z. Slocum, "Your Google Docs: Soon in Search Results?" http://news.cnet.com/8301-17939_109-10357137-2.html, 2009.

[6] B. Krebs, "Payment Processor Breach May Be Largest Ever," http://voices.washingtonpost.com/securityfix/2009/01/payment_processor_breach_may_b.html, Jan. 2009.

[7] I.H. Witten, A. Moffat, and T.C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, May 1999.

[8] D. Song, D. Wagner, and A. Perrig, "Practical Techniques for Searches on Encrypted Data," *Proc. IEEE Symp. Security and Privacy,* 2000.

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

[10] D. Boneh, G.D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public Key Encryption with Keyword Search," *Proc. Int'l Conf. Advances in Cryptology (EUROCRYP '04),* 2004.

[11] Y.-C. Chang and M. Mitzenmacher, "Privacy Preserving Keyword Searches on Remote Encrypted Data," *Proc. Int'l Conf. Applied Cryptography and Network Security (ACNS '05),* 2005.

[12] R. Curtmola, J.A. Garay, S. Kamara, and R. Ostrovsky, "Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions," *Proc. ACM Conf. Computer and Comm. Security (CCS '06),* 2006.

[13] A. Singhal, "Modern Information Retrieval: A Brief Overview," *IEEE Data Eng. Bull.,* vol. 24, no. 4, pp. 35-43, 2001.

[14] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill, "Order-Preserving Symmetric Encryption," *Proc. Int'l Conf. Advances in Cryptology (Eurocrypt '09),* 2009.

[15] J. Zobel and A. Moffat, "Exploring the Similarity Space," *SIGIR Forum,* vol. 32, no. 1, pp. 18-34, 1998.

[16] O. Goldreich and R. Ostrovsky, "Software Protection and Simulation on Oblivious Rams," *J. ACM,* vol. 43, no. 3, pp. 431-473, 1996.

[17] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and Efficiently Searchable Encryption," *Proc. Ann. Int'l Cryptology Conf. Advances in Cryptology (Crypto '07),* 2007.

[18] S. Zerr, D. Olmedilla, W. Nejdl, and W. Siberski, "Zerber+r: Top-k Retrieval from a Confidential Index," *Proc. Int'l Conf. Extending Database Technology: Advances in Database Technology (EDBT '09),* 2009.

[19] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling Public Verifiability and Data Dynamics for Storage Security in Cloud Computing," *IEEE Trans. Parallel and Distributed Systems,* vol. 22, no. 5, pp. 847-859, May 2011.

[20] C. Wang, Q. Wang, K. Ren, and W. Lou, "Towards Secure and Dependable Storage Services in Cloud Computing," *IEEE Trans. Service Computing,* to appear.

[21] C. Wang, S. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-Preserving Public Auditing for Secure Cloud Storage," *IEEE Trans. Computers,* to appear.

[22] 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," *Proc. Workshop Storage Security and Survivability,* 2007.

[23] RFC "Request for Comments Database," http://www.ietf.org/rfc.html, 2012.

[24] B. Waters, D. Balfanz, G. Durfee, and D. Smetters, "Building an Encrypted and Searchable Audit Log," *Proc. Ann. Network and Distributed Security Symp. (NDSS '04),* 2004.

[25] F. Bao, R. Deng, X. Ding, and Y. Yang, "Private Query on Encrypted Data in Multi-User Settings," *Proc. Int'l Conf. Information Security Practice and Experience (ISPEC '08),* 2008.

[26] P. Golle, J. Staddon, and B.R. Waters, "Secure Conjunctive Keyword Search over Encrypted Data," *Proc. Second Int'l Conf. Applied Cryptography and Network Security (ANCS '04),* pp. 31-45, 2004.

[27] L. Ballard, S. Kamara, and F. Monrose, "Achieving Efficient Conjunctive Keyword Searches over Encrypted Data," *Proc. Int'l Conf. Information and Comm. Security (ICICS '05),* 2005.

[28] Y.H. Hwang and P.J. Lee, "Public Key Encryption with Conjunctive Keyword Search and Its Extension to a Multi-User System," *Proc. Int'l Conf. Pairing-Based Cryptography (Pairing '07),* pp. 31-45, 2007.

[29] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy Keyword Search over Encrypted Data in Cloud Computing," *Proc. IEEE INFOCOM '10,* 2010.

[30] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data," *Proc. IEEE INFOCOM '11,* 2011.

[31] D. Boneh and B. Waters, "Conjunctive, Subset, and Range Queries on Encrypted Data," *Proc. Fourth Conf. Theory of Cryptography (TCC '07),* pp. 535-554, 2007.

[32] E. Shi, J. Bethencourt, H. Chan, D. Song, and A. Perrig, "Multi-Dimensional Range Query over Encrypted Data," *Proc. IEEE Symp. Security and Privacy,* 2007.

[33] K. Ren, C. Wang, and Q. Wang, "Security Challenges for the Public Cloud," *IEEE Internet Computing,* vol. 16, no. 1, pp. 69-73, 2012.

[34] C. Wang, K. Ren, S. Yu, K. Mahendra, and R. Urs, "Achieving Usable and Privacy-Assured Similarity Search over Outsourced Cloud Data," *Proc. IEEE INFOCOM,* 2012.

Cong Wang received the BE and ME degrees from Wuhan University, China, in 2004 and 2007, respectively. He is currently working toward PhD degree in the Electrical and Computer Engineering Department at Illinois Institute of Technology. He has been a summer intern at Palo Alto Research Centre in 2011. His research interests are in the areas of applied cryptography and network security, with current focus on secure data service outsourcing in Cloud Computing. He is a student member of the IEEE.

Ning Cao received the BE and ME degrees from Xi'an Jiaotong University, China, in 2002 and 2008, respectively. He is currently working toward the PhD degree in the Electrical and Computer Engineering Department at Worcester Polytechnic Institute. His research interests are in the areas of security, privacy, and reliability in Cloud Computing. He is a student member of the IEEE.

Kui Ren received the PhD degree from Worcester Polytechnic Institute. He is currently an assistant professor in the Electrical and Computer Engineering Department at the Illinois Institute of Technology. His research expertise includes Cloud Computing and Security, Wireless Security, and Smart Grid Security. His research is supported by US National Science Foundation (NSF) (TC, NeTS, CSR, NeTS-Neco), US Department of Energy (DoE), AFRL, and Amazon. He is a recipient of National Science Foundation Faculty Early Career Development (CAREER) Award in 2011. He received the Best Paper Award from IEEE ICNP 2011. He serves as an associate editor for *IEEE Wireless Communications* and *IEEE Transactions on Smart Grid*. He is a senior member of the IEEE and a member of the ACM.

Wenjing Lou received the BE and ME degrees in computer science and engineering from Xi'an Jiaotong University, China, in 1993 and 1996, respectively. She received the MASc degree from Nanyang Technological University, Singapore, in 1998 and the PhD degree in electrical and computer engineering from University of Florida in 2003. She joined the Computer Science Department at Virginia Polytechnic Institute and State University in 2011 and has been an associate professor with tenure since then. Prior to that, she had been on the faculty of the Department of Electrical and Computer Engineering at Worcester Polytechnic Institute, where she had been an assistant professor since 2003 and was promoted to associate professor with tenure in 2009. She is currently serving on the editorial board of five journals: *IEEE Transactions on Wireless Communications*, *IEEE Transactions on Smart Grid*, *IEEE Wireless Communications Letter*, *Elsevier Computer Networks*, and *Springer Wireless Networks*. She has served as TPC cochair for the security symposium of several leading IEEE conferences. She was named Joseph Samuel Satin Distinguished fellow in 2006 by WPI. She was the recipient of the US National Science Foundation (NSF) Faculty Early Career Development (CAREER) award in 2008. She received the Sigma Xi Junior Faculty Research Award at WPI in 2009. She is a senior member of the IEEE.