Privacy-preserving Multi-keyword Text Search in the Cloud Supporting Similarity-based Ranking

Wenhai Sun[†,††]
whsun@xidian.edu.cn

Bing Wang[††]
bingwang@vt.edu

Ning Cao[§]
ncao@ece.wpi.edu

Ming Li[‡]
ming.li@usu.edu

Wenjing Lou[††]
wjlou@vt.edu

Y. Thomas Hou[††]
thou@vt.edu

Hui Li[†]
lihui@mail.xidian.edu.cn

[†]The State Key Laboratory of Integrated Services Networks, Xidian University, China
[††]Virginia Polytechnic Institute and State University, USA
[§]Worcester Polytechnic Institute, USA
[‡]Utah State University, USA

ABSTRACT

With the increasing popularity of cloud computing, huge amount of documents are outsourced to the cloud for reduced management cost and ease of access. Although encryption helps protecting user data confidentiality, it leaves the well-functioning yet practically-efficient secure search functions over encrypted data a challenging problem. In this paper, we present a privacy-preserving multi-keyword text search (MTS) scheme with similarity-based ranking to address this problem. To support multi-keyword search and search result ranking, we propose to build the search index based on *term frequency* and the *vector space model* with *cosine similarity measure* to achieve higher search result accuracy. To improve the search efficiency, we propose a tree-based index structure and various adaption methods for *multi-dimensional (MD) algorithm* so that the practical search efficiency is much better than that of linear search. To further enhance the search privacy, we propose two secure index schemes to meet the stringent privacy requirements under strong threat models, i.e., known ciphertext model and known background model. Finally, we demonstrate the effectiveness and efficiency of the proposed schemes through extensive experimental evaluation.

Categories and Subject Descriptors

E.3 [**Data Encryption**]; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

Keywords

Cloud Computing; Privacy-preserving Search; Multi-keyword Search; Similarity-based Ranking

1. INTRODUCTION

Cloud computing is a new model of enterprise IT infrastructure that enables ubiquitous, convenient, and on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) [8]. Due to the centralized management of elastic resources, all players in this emerging X-as-a-service (XaaS) model, including the cloud provider, application developers, and end-users, can reap benefits. Especially, for the end-users, they can outsource large volumes of data and workloads to the cloud and enjoy the virtually unlimited computing resources in a pay-per-use manner. Indeed, many companies, organizations, and individual users have adopted the cloud platform to facilitate their business operations, research, or everyday needs [24].

Despite the tremendous business and technical advantages, privacy concern is one of the primary hurdles that prevents the widespread adoption of the cloud by potential users, especially if their sensitive data are to be outsourced to and computed in the cloud. Examples may include financial and medical records, and social network profiles. Cloud service providers (CSPs) usually enforce users' data security through mechanisms like firewalls and virtualization. However, these mechanisms do not protect users' privacy from the CSP itself since the CSP possesses full control of the system hardware and lower levels of software stack. There may exist disgruntled, profiteered, or curious employees that can access users' sensitive information for unauthorized purposes [14, 26]. Although encryption before data outsourcing [15, 33] can preserve data privacy against the CSP, it also makes the effective data utilization, such as search over encrypted data, a very challenging task. Without being able to extract useful information from the outsourced data in a secure and private manner, the cloud will merely be a remote storage which provides limited value to all parties.

One fundamental and common form of data utilization is the search operation, i.e., to quickly sort out information of interest from huge amount of data. The information retrieval community has the state-of-the-art techniques that are readily available to achieve rich search functionalities, such as result ranking and multi-keyword queries, on plaintext. For example, *cosine measure* in the *vector space model* [30] is a state-of-the-art similarity measure widely used in plaintext information retrieval community, which incorporates the "term frequency (TF) × inverse document frequency (IDF)" weight to evaluate the similarity between a document and a particular query, and yield accurate ranked search result. However, implementing a secure version of such techniques over outsourced encrypted data in the cloud is not straightforward, and is susceptible to privacy breach [29]. Although inverted index (a.k.a. inverted file) is the most popular and efficient index data structure used in document retrieval systems, it is not directly applicable in TF-based multi-keyword encrypted text search environment [28, 29, 34].

In the literature, searchable encryption (SE) techniques can partially address the need for secure outsourced data search. Many researchers have developed SE schemes that allow searches over encrypted keyword indexes, either based on public key cryptography (PKC) [3,4,12,13] or symmetric key cryptography (SKC) [6, 10, 11, 16, 27]. In general, although the PKC-based schemes allow more expressive queries than SKC-based ones, they are much less efficient. Thus, there has been significant interest in developing efficient SKC-based SE mechanisms. Curtmola et al. were the first to propose a symmetric SE scheme with security guarantees under rigorous definitions [10]. Other works [28, 29, 34] targeted on providing ranked search. These schemes only support single-keyword queries which is too restrictive for practical use. To enrich search functionality, Cao et al. [5] attempted privacy-preserving multi-keyword ranked search over encrypted cloud data. Nevertheless, the search complexity is linear to the number of documents in the dataset, which becomes undesirable and inefficient when a huge amount of documents are present. In addition, the heuristic ranking function, i.e., "coordinate matching", failed to yield more accurate search result, compared to the state-of-the-art multi-keyword search over plaintext. Thus, the quest for secure data search mechanisms that can simultaneously achieve high efficiency and functionality (such as expressive/usable queries) still remains open up to date.

In this paper, we address the challenges of constructing practically efficient and flexible encrypted search functionalities that support result ranking and multi-keyword queries. In particular, to support multi-keyword queries and search result ranking functionalities, we propose to build the search index based on the vector space model , i.e., cosine measure, and incorporate the *TF* × *IDF* weight to achieve high search result accuracy. To improve the search efficiency, we propose a tree-based index structure, where each value in a node is a vector of term frequency related information. We then apply the search algorithm, adapted from the MD-algorithm [19], so as to realize efficient search functionality. Our basic scheme for multi-keyword text search with similarity-based ranking (BMTS) is secure under the known ciphertext model. In order to further enhance the search

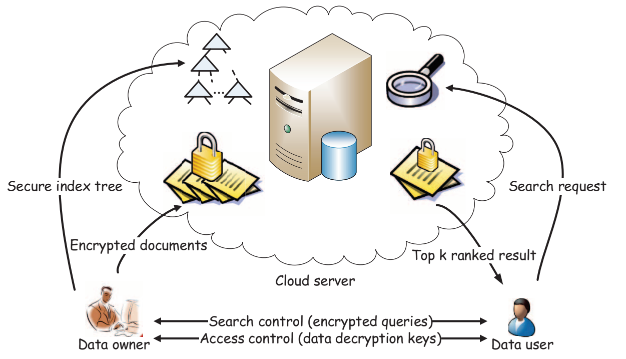


Figure 1: Framework of the search over outsourced encrypted cloud data

privacy, we propose another enhanced secure index scheme (EMTS) against sensitive frequency information leakage to meet more stringent privacy requirements under a stronger threat model, i.e., known background model. Finally, we demonstrate the effectiveness and efficiency of the proposed schemes through extensive experimental evaluation.

Our contributions are summarized as follows:

1. By incorporating the state-of-the-art information retrieval techniques, we propose a privacy-preserving multi-keyword text search scheme supporting similarity-based ranking, which enjoys the same flexibility and search result accuracy as the existing state-of-the-art multi-keyword search over plaintext.

2. We propose a randomization (phantom terms) approach in the enhanced scheme to prevent sensitive frequency information leakage thus achieving better privacy of keywords. We show that with the proposed methods, user can balance between search precision and privacy.

3. With improved security guarantee, EMTS is still comparable in search time to BMTS. In addition, we investigated various index building methods to speed up the search of common cases. The results demonstrate much improved search efficiency compared with [5].

2. PROBLEM FORMULATION

2.1 System Model

The system model considered in this paper consists of three entities: the *data owner*, the *data user*, and the *cloud server*, as illustrated in Fig. 1. The data owner outsources a huge size of document collection $\mathcal{DC}$ in the encrypted form $\mathcal{C}$, together with an encrypted searchable index tree $\mathcal{I}$ generated from $\mathcal{DC}$, to the cloud server. We assume that the data user has the mutual authentication capability with the data owner. As such, search control mechanisms can be applied here, e.g., broadcast encryption [10], through which the data user obtains the encrypted search query $\widetilde{Q}$. Upon the receipt of $\widetilde{Q}$, the cloud server starts searching the index tree $\mathcal{I}$ and will return the corresponding set of encrypted documents, which have been well-ranked by our frequency based similarity measures (as will be introduced shortly).

An additional feature is that the data user may not want to receive all the relevant documents. Instead, the data user may send a search parameter k along with the search query $\widetilde{Q}$ such that the cloud server only returns the top-k most relevant documents. The capability of the user to decrypt the received documents [15,33] is a separate issue and is out of the scope of this paper.

2.2 Threat Model

We assume that the data user is honest but that the cloud server acts in an "honest-but-curious" manner, which is also employed by related works on secure cloud data search [5, 29]. In other words, the cloud server honestly follows the protocol execution, but curiosity propels him/her to the speculation and analysis over the data and searchable index tree available at the server. Depending on the available information to the cloud server, two threat models are considered here.

Known Ciphertext Model: Only the encrypted document set $\mathcal{C}$, searchable index tree $\mathcal{I}$ and encrypted query vector $\widetilde{Q}$, all of which are outsourced from the data owner, are available to the cloud server. Specifically, we intend to protect the plaintext query/index information against the cloud server and keep the dictionary $\mathcal{T}$ as secret that was used to build the searchable index tree $\mathcal{I}$.

Known Background Model: In this stronger model, the cloud server is equipped with more knowledge than what can be accessed in the known ciphertext model. In particular, the attacker may extract the statistical information from a known comparable dataset which bears the similar nature to the targeting dataset, e.g., the TF distribution information of a specific keyword. Given such statistical information, the cloud server is able to launch statistical attack to deduce/identify specific keywords in the query [28, 29, 34].

2.3 Design Goals

To enable effective, efficient and secure multi-keyword ranked search over encrypted cloud data under the aforementioned models, our mechanism is aiming to achieve the following design goals.

Accuracy-improved Multi-keyword Ranked Search: To design an encrypted cloud data search scheme which not only supports the effective *multi-keyword search* functionality, but also, by adoption of the vector space model, achieves the accuracy-improved *similarity-based search result ranking.*

Search Efficiency: Instead of linear search [5], we explore a tree-based index structure and an efficient search algorithm to achieve better practical search efficiency.

Privacy Goals: The general goal is to protect user privacy by preventing the cloud server from learning information of the document set, the index tree, and the queries. In particular, search privacy requirements that we are concerned with are 1) *Index Confidentiality*: the underlying plaintext information pertaining to the encrypted index tree, e.g., keywords and TF of keywords; 2) *Query Confidentiality*: the plaintext information regarding the encrypted query, e.g., keywords in the query and document frequency (DF) of these keywords; 3) *Query Unlinkability*: whether two or more encrypted queries are from the same search request; 4) *Keyword Privacy*: the identification of specific keyword in the index tree, in the query or in the document set. Note that protecting *access pattern*, i.e., the sequence of returned documents, is extremely expensive since the algorithm has to "touch" the whole document set [7]. We do not aim to protect it in this work for efficiency concerns.

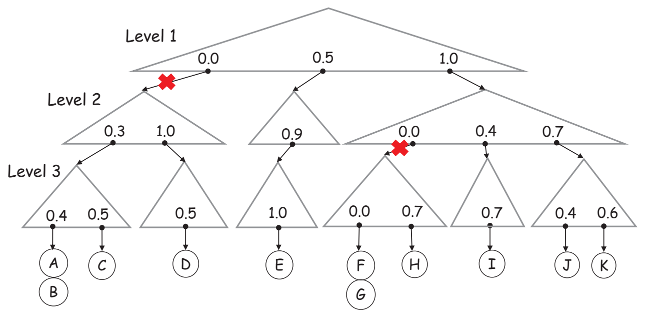


Figure 2: Illustration of the MD-algorithm on the MDB-tree

2.4 Notations

For the sake of clarity, we introduce the main notations used in this paper.

- $\mathcal{DC}$ – the plaintext document collection, denoted as a set of m documents $\mathcal{DC} = \{d|d_1, d_2, \ldots, d_m\}$.
- $\mathcal{C}$ – the encrypted form of $\mathcal{DC}$ stored in the cloud server, denoted as $\mathcal{C} = \{c|c_1, c_2, \ldots, c_m\}$.
- $\mathcal{T}$ – the dictionary, composed of n keywords, denoted as $\mathcal{T} = \{t|t_1, t_2, \ldots, t_n\}$.
- $\bar{\mathcal{T}}$ – a subset of $\mathcal{T}$, indicating the keywords in a search request.
- $\mathcal{I}$ – the h-level searchable index tree for the whole document set $\mathcal{DC}$. Each level corresponds to an exclusive subset of keywords. Documents c_i's are associated with the leaf nodes.
- $\mathcal{T}_i$ – a subset of $\mathcal{T}$, which constitutes the i^{th} level of $\mathcal{I}$, $i = 1, \ldots, h$.
- D_d – the index vector of document d for all the keywords in $\mathcal{T}$.
- Q – the query vector for the keyword set $\bar{\mathcal{T}}$.
- $\widetilde{D_d}$ – the encrypted form of D_d.
- $\widetilde{Q}$ – the encrypted form of Q.

2.5 Preliminaries

Vector Space Model: Among many similarity measures in plaintext information retrieval, vector space model [30] is the most popular one, supporting both conjunctive search and disjunctive search. Specifically, document rankings are realized by comparing the deviation of angles, i.e., cosine values, between each document vector and the query vector. The cosine measure allows accurate rankings due to the "TF×IDF rule", where TF denotes the occurrence count of a term within a document (it is used to measure how important a specific term is to a particular document), and IDF is obtained by dividing the total number of documents in the collection by the number of documents containing the term (it implies that this frequency of a term tends to be inversely proportional to its ranking). We adopt the similarity evaluation function for cosine measure from [30], where

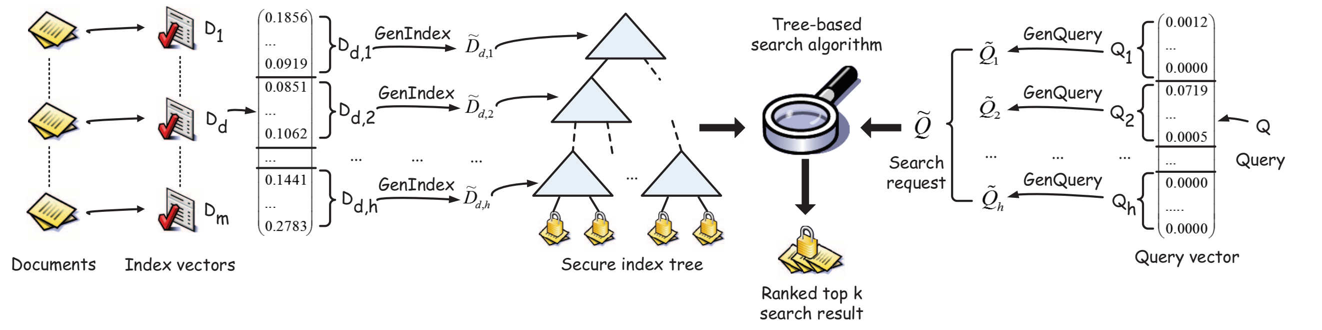


Figure 3: Overview of secure index scheme

the calculation methods for TF weight and IDF weight are equally effective compared to others [35]. The following statistical values are used in our similarity evaluation function:

- $f_{d,t}$, the TF of the keyword t within the document d;
- f_t, the number of documents containing the keyword t;
- N, the total number of documents in the document set;
- $w_{d,t}$, the TF weight for $f_{d,t}$;
- $w_{q,t}$, the IDF weight (query weight);
- W_d, the Euclidean length of $w_{d,t}$;
- W_q, the Euclidean length of $w_{q,t}$.

The definition of the similarity function is as follows:

$$Cos(D_d, Q) = \frac{1}{W_d W_q} \sum_{t \in Q \cap D_d} w_{d,t} \cdot w_{q,t} \qquad (1)$$

where $W_d = \sqrt{\sum_{t \in Q \cap D_d} w_{d,t}^2}$, $W_q = \sqrt{\sum_{t \in Q \cap D_d} w_{q,t}^2}$, $w_{d,t} = 1 + \ln(f_{d,t})$, $w_{q,t} = \ln(1 + \frac{N}{f_t})$, and hence, the index vector D_d and query vector Q are both unit vectors.

MD-algorithm: The MD-algorithm [19] is used to find the k-best matches in a database that is structured as an MDB-tree [22], as shown in Fig. 2. In the database scenario, each level of the MDB-tree represents an attribute domain and each attribute in that domain is assigned an attribute value. All the attributes sharing the same value in the upper domain forms a child node. As such, a set of objects is allowed to be indexed in one data structure. An important search parameter, the prediction threshold value $\hat{P}_i$ for each level i, is obtained from the maximum attribute value P_i at each level, for example, in Fig. 2, $\hat{P}_i = P_i = 1.0$. In a depth-first manner, MD-algorithm starts from the root node with a recursive procedure upon this tree. Specifically, search process selects the unused maximum attribute value when it enters a node, and based on $\hat{P}_i$'s below this level, predicts the maximum possible final score to be obtained. The criteria for node selection is that if this predicted final score is less than or equal to the minimum score of the top-k objects which have been selected, search process returns to the parent node, otherwise, it goes down to the child node at the next level. This procedure is executed recursively until the objects with top-k scores are selected. The search can be done very efficiently due to the relatively accurate final score prediction, and thus only part of the objects in the tree are accessed. Fig. 2 shows an example that, when $k = 3$, the set of objects, E, K, and J, are returned to the user and the cross signs in the figure indicate that it is not necessary to access the nodes below. More details of the MD-algorithm and MDB-tree can be found in [19].

3. SECURE INDEX SCHEME

To achieve accurate multi-keyword ranked search, we adopt the cosine measure to evaluate similarity scores. In particular, we divide the original long document index vector D_d into multiple sub-vectors such that each sub-vector $D_{d,i}$ represents a subset of keywords $\mathcal{T}_i$ of $\mathcal{T}$, and becomes a part of the i^{th} level of the index tree $\mathcal{I}$, as shown in Fig. 3. The query vector Q is divided in the same way D_d is done. Let Q_i be the query sub-vector at the i^{th} level. As such, the final similarity score for document d can be obtained by summing up the scores from each level. Based on these similarity scores, the cloud server determines the relevance of document d to the query Q and sends the top-k most relevant documents back to the user. By using the level-wise secure inner product scheme, similar to the techniques applied in [5, 31], the document index vector $D_{d,i}$ and the query vector Q_i are both well protected, and we show that this basic scheme is secure in the known ciphertext model. To further protect the sensitive frequency information from leakage, we also propose an enhanced scheme in the known background model.

3.1 BMTS in Known Ciphertext Model

In order to facilitate the relevance rankings, the similarity scores, i.e., cosine values, are revealed to the cloud server, which differs from the schemes adopted in [5, 31]. In other words, we do not apply the dimension extension technique to our basic scheme in the known ciphertext model. For each level i of $\mathcal{I}$, our basic secure index scheme can be described as follows:

- **Setup** In this initialization phase, the secret key SK_i is produced by the data owner, including: 1) a $|\mathcal{T}_i|$-bit randomly generated vector S_i, where $|\mathcal{T}_i|$ is the length of $\mathcal{T}_i$; 2) two $(|\mathcal{T}_i| \times |\mathcal{T}_i|)$ invertible random matrices $\{M_{1,i}, M_{2,i}\}$. Hence, SK_i can be denoted as a 3-tuple $\{S_i, M_{1,i}, M_{2,i}\}$.

- **GenIndex** $(\mathcal{DC}, SK_i)$ For each document d, the data owner generates an index vector $D_{d,i}$ according to $\mathcal{T}_i$, and each dimension is a normalized TF weight $w_{d,t}$.

Next, the splitting procedure is applied to $D_{d,i}$, which splits $D_{d,i}$ into two random vectors as $\{D_{d,i}', D_{d,i}''\}$. Specifically, with the $|\mathcal{T}_i|$-bit vector S_i as a splitting indicator, if the j^{th} bit of S_i is 0, $D_{d,i}'[j]$ and $D_{d,i}''[j]$ are set as the same as $D_{d,i}[j]$; if the j^{th} bit of S_i is 1, $D_{d,i}'[j]$ and $D_{d,i}''[j]$ are set to two random numbers so that their sum is equal to $D_{d,i}[j]$. Finally, the encrypted index vector $\widetilde{D_{d,i}}$ is built as $\{M_{1,i}^T D_{d,i}', M_{2,i}^T D_{d,i}''\}$.

- **GenQuery**$(\bar{\mathcal{T}}, SK_i)$ With the keywords of interest in $\bar{\mathcal{T}}$, the query vector Q_i is generated, where each dimension is a normalized IDF weight $w_{q,t}$ ($w_{q,t} = 0$ for any keyword t not present in Q_i). Subsequently, Q_i is split into two random vectors as $\{Q_i', Q_i''\}$ with the similar splitting procedure. The difference is that if the j^{th} bit of S_i is 0, $Q_i'[j]$ and $Q_i''[j]$ are set to two random numbers so that their sum is equal to $Q_i[j]$; if the j^{th} bit of S_i is 1, $Q_i'[j]$ and $Q_i''[j]$ are set as the same as $Q_i[j]$. Finally, the encrypted query vector $\widetilde{Q_i}$ is yielded as $\{M_{1,i}^{-1} Q_i', M_{2,i}^{-1} Q_i''\}$.
- **SimEvaluation** $(\widetilde{D_{d,i}}, \widetilde{Q_i})$ The cloud server executes similarity evaluation with query vector $\widetilde{Q_i}$ as in Eq. 2.

The similarity score at the i^{th} level is computed as follows:

$$\begin{aligned}
&Cos(\widetilde{D_{d,i}}, \widetilde{Q_i}) \\
=&\{M_{1,i}^T D_{d,i}', M_{2,i}^T D_{d,i}''\} \cdot \{M_{1,i}^{-1} Q_i', M_{2,i}^{-1} Q_i''\} \\
=&D_{d,i}' \cdot Q_i' + D_{d,i}'' \cdot Q_i'' \\
=&D_{d,i} \cdot Q_i.
\end{aligned} \tag{2}$$

Hence, the final similarity score for document d is $\sum_{i=1}^{h} D_{d,i} \cdot Q_i = D_d \cdot Q$.

Security Analysis We analyze BMTS concerning the search privacy requirements as described in section 2.

1) Index confidentiality and Query confidentiality: In BMTS, $\widetilde{D_{d,i}}$ and $\widetilde{Q_i}$ are obfuscated vectors. As long as the secret key SK_i is kept confidential, the cloud server cannot infer the original vectors $D_{d,i}$ or Q_i. Neither can it deduce the keywords nor the TF and IDF information included in the documents or queries from the result similarity scores, which appear to be random values to the server. This has been proven in the known ciphertext model in [31]. Therefore, *index confidentiality* and *query confidentiality* are well protected.

2) Query unlinkability: The adopted vector encryption method provides non-deterministic encryption, in light of the random vector splitting procedure. Thus the same search request (e.g. same search keywords) will be encrypted to different query vector $\widetilde{Q}$. The non-linkability of search requests can be provided to this extent. However, if a cloud server is capable of tracking the nodes visited and the intermediate similarity results, it is possible for the cloud server to link the same search request based on the same similarity scores. In this case the *search pattern* or the *access pattern* will be leaked even in the known ciphertext model.

3) Keyword privacy: In the known background model, the cloud server may have the knowledge of not only the TF distributions, but also the normalized TF distributions of some sensitive keywords from a known comparable dataset.

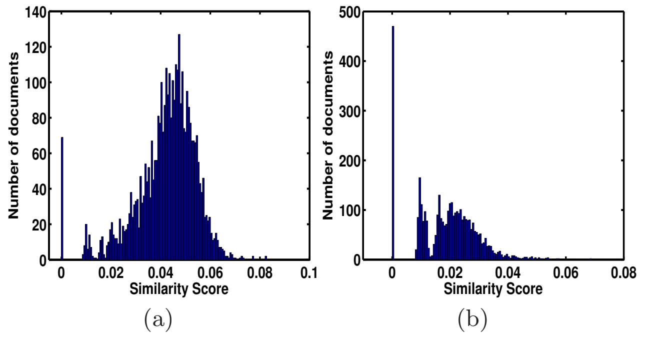


Figure 4: Distribution of similarity score when a single keyword in a query vector with our basic scheme. (a) For keyword "network". (b) For keyword "search".

It is worth noting that these distributions are keyword specific respectively, as shown in Fig. 4, such that the corresponding keywords can be differentiated by the slope and value range of these distributions [28, 29, 34]. In the worst case, where only one keyword t appears in the query vector Q (the normalized $w_{q,t}$ is 1), the normalized TF distribution of this keyword is exposed directly.

In order to enhance security and boost search efficiency, search evaluation may be only executed at certain levels where the user-intended keywords reside; for the other levels, we can render these similarity scores some fixed values, e.g., 0, during the execution of both similarity score prediction and evaluation in the search process.

3.2 EMTS in Known Background Model

The previous security analysis shows that in the known background model, *keyword privacy* breach is possible, due to the distance-preserving property of BMTS, i.e., the cosine value calculated from $\widetilde{D_{d,i}}$ and $\widetilde{Q_i}$ is equal to the one from $D_{d,i}$ and Q_i. In order to break such equality, we introduce some tunable randomness into the similarity score evaluation, by which the cloud server cannot differentiate keywords from the particular similarity score distributions. In addition, this randomness can be calibrated by the user to represent the user's preference for the more accurate ranked search result versus better-protected *keyword privacy*. Specifically, U_i phantom terms are added into the query vector Q_i, and we extend the index vector $D_{d,i}$ from $|\mathcal{T}_i|$ dimensions to $|\mathcal{T}_i| + U_i$ dimensions. We denote the subset of h levels where the keywords of interest reside as w and its size $|w| \leq h$.

Our EMTS scheme is performed almost the same as BMTS, except that at the i^{th} level: 1) in Setup phase, S_i becomes $(|\mathcal{T}_i| + U_i)$-bit long. $M_{1,i}$ and $M_{2,i}$ are $(|\mathcal{T}_i| + U_i) \times (|\mathcal{T}_i| + U_i)$ dimensional matrices; 2) in GenIndex phase, by choosing V_i out of U_i phantom terms, the corresponding entries in the $(|\mathcal{T}_i| + U_i)$-dimensional index vector $D_{d,i}$ are set to 1; 3) when generating encrypted query, the $(|\mathcal{T}_i| + j)^{th}$ entry in Q_i where $j \in [1, U_i]$ is set to a random number $\varepsilon_{i,j}$; 4) The cloud server executes similarity evaluation and obtains the final similarity score for document d equal to $(D_d \cdot Q + \sum_{i \in w} \sum_{j \in \bar{V}_i} \varepsilon_{i,j})$, where $\bar{V}_i$ is the set of the V_i selected phantom terms, and it is different for each index vector at level i.

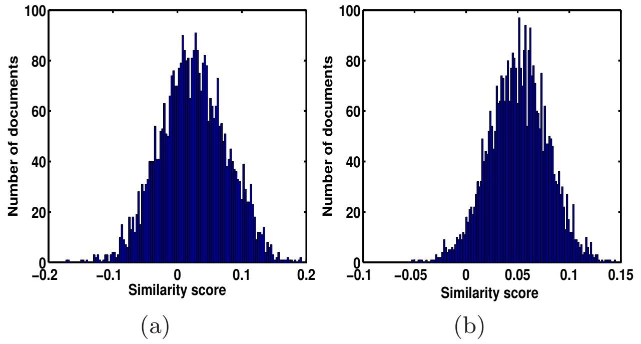


Figure 5: Distribution of similarity score for keyword "network" with different standard deviations, 3500 documents, only one keyword "network" in the query, in our enhanced scheme. (a) $\sigma = 0.05$. (b) $\sigma = 0.03$.

Security Analysis We analyze EMTS again with respect to the aforementioned search privacy requirements.

1) Index confidentiality and Query confidentiality: EMTS can protect *index confidentiality* and *query confidentiality* in both the known ciphertext model and the known background model, which is inherited from BMTS.

2) Query unlinkability: The introduction of randomly generated $\varepsilon_{i,j}$ will allow EMTS to produce different similarity scores even for the same search request. The value of $\varepsilon_{i,j}$ can be adjusted to control the level of variance thus the level of unlinkability. It is worth noting that this query-side randomization technique significantly differs from [5], where randomization occurs on the index vector side and is not possible to be tweaked as an effective privacy-preserving parameter for users. Query unlinkability is thus much enhanced compared with BMTS to the extent that there is no easy way for the attacker to link the queries. However, since we do not intend to protect access pattern for efficiency reasons, the returned results from the same request will always bear some similarity which could be exploited with powerful statistical analysis by the very motivated cloud server. This is a trade-off that one has to make between efficiency and privacy.

3) Keyword privacy: At level i, the number of the index vectors is denoted as l_i. On one hand, to render separate $\sum_{j\in\bar{V}_i}\varepsilon_{i,j}$ of this level the different values for one search request, we set $\binom{U_i}{V_i} \geq l_i$. On the other hand, $\sum_{j\in\bar{V}_i}\varepsilon_{i,j}$ for the same $D_{d,i}$ is different with multiple search requests, since $\varepsilon_{i,j}$ is generated uniformly at random upon each search request. The cloud server cannot eliminate the impact of these phantom terms from the final similarity scores without the exact values of them. Furthermore, every $\varepsilon_{i,j}$ follows the same uniform distribution $M(\mu' - c, \mu' + c)$, where the mean is μ' and the variance as ${\sigma'}^2$ is $c^2/3$. According to the central limit theorem, the $\sum_{i\in w}\sum_{j\in\bar{V}_i}\varepsilon_{i,j}$ follows the Normal distribution $N(\mu, \sigma^2)$, where the mean as μ is $\sum_{i\in w} V_i\mu'$ and the variance as σ^2 is $\sum_{i\in w} V_i c^2/3$. Therefore, we may generate $\varepsilon_{i,j}$ with the value of μ' as $\mu/\sum_{i\in w} V_i$ and the value of c as $\sqrt{3/\sum_{i\in w} V_i}\cdot\sigma$. As shown in Fig. 5, with larger σ selected by the user, it is more difficult for the cloud server to infer the corresponding statistical information, and further reverse-engineer the keyword, from the well-obfuscated distribution of the similarity score. However, it does not suffice to protect *keyword privacy*. For simplicity, we assume that there are 2 levels, i.e., only one keyword t at level 1 and two or more other keywords at level 2. At level 1, $\sum_{j\in\bar{V}_1}\varepsilon_{1,j}$ does not follow the Normal distribution with σ selected by the user, in that with the smaller V_1, the value of ${\sigma_1}^2$ as $V_1c^2/3$ is smaller than the value of σ^2 as $(V_1 + V_2)c^2/3$. It is possible that σ_1 is too small to obfuscate the distribution of the similarity score, so that the cloud server may identify the keyword t at level 1. For better protecting *keyword privacy*, the user chooses an appropriate σ_1, i.e., large enough to obfuscate the distribution, to generate $\varepsilon_{1,j}$ accordingly, while σ remains as the overall search parameter. Hence, the variance ${\sigma_2}^2$ for level 2 can be set as $\sigma^2 - {\sigma_1}^2$ and $\varepsilon_{2,j}$ is generated accordingly, as the normal random variables $\sum_{j\in\bar{V}_1}\varepsilon_{1,j}$ and $\sum_{j\in\bar{V}_2}\varepsilon_{2,j}$ are independent to each other. Finally, *keyword privacy* can be well protected by these phantom keywords.

Remarks Recently Yao et al. [32] find that this underlying encryption method [31] is susceptible to chosen plaintext attack. However, it is not applicable under our defined threat models, since in order to launch such attack, the cloud server has to acquire plaintext query information, i.e., the normalized IDF weights, which are *only* possessed by the data owner and protected by BMTS and EMTS.

4. EFFICIENCY OF THE TREE-BASED SEARCH ALGORITHM

In the plaintext information retrieval community, many well-developed techniques have been adopted to accelerate the search process, e.g., inverted index [18], B-tree [9], etc. However, in the ciphertext scenario, they cannot be implemented in a straightforward manner. In [10, 28, 29, 34], the inverted index based search methods are employed to achieve an extremely efficient search process. However, these schemes are only designed for single keyword search. Efficient range search in database [17] can be realized by using B^+-tree, but it is not applicable to the text search scenario. The similarity score in our scheme is a value depending on the query and has to be evaluated in the runtime, which makes the fixed tree structures, such as B-tree or B^+-tree, not suitable here. In this paper, we propose a tree-based search algorithm, which is adapted from MDB-tree based MD-algorithm, to enable efficient multi-keyword ranked search. In what follows, we briefly introduce our tree-based search algorithm and present some experimental results from our implementation of the proposed tree-based search algorithm on a real-world document set: the recent ten years' INFOCOM publications. We identify key factors that affect the search efficiency and propose strategies in building the index tree that effectively speed up the search process.

4.1 Tree-based Search Algorithm

The MD-algorithm is originally designed for plaintext database search. In the case of privacy-preserving similarity-based multi-keyword ranked text search, it cannot be applied in a straightforward manner. Instead of a numerical "attribute value" for each attribute in the MDB-tree, our index tree structure has to be built on vectors. The secure index scheme described in section 3 is for this purpose and it enables the

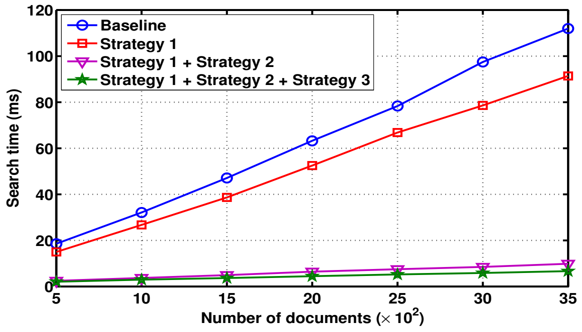


Figure 6: Comparison of search efficiency with different efficiency-improving strategies

search algorithm to take the inputs of the encrypted searchable index tree and the encrypted query, and ensures that the search algorithm is conducted in a secure way to protect important search privacy in the whole search process.

Another remarkable difference between our search algorithm and MD-algorithm is that we cannot set $\hat{P}_i$ to P_i as running the MD-algorithm in database scenario, since P_i varies for queries in our scenario and has to be securely evaluated (as described in section 3) in the runtime. The pseudo code for our tree-based search algorithm is presented in Appendix.

4.2 Impact of Prediction Threshold Value

An important factor that affects the search efficiency is the prediction threshold value $\hat{P}_i$ at each level i. To ensure the search precision, $\hat{P}_i \geq P_i$ should hold where P_i is the maximum similarity score at level i. As shown in Tab. 1, the tighter the prediction value of $\hat{P}_i$, the higher the search efficiency. The reason is that the search process can be terminated earlier without going into unnecessary nodes. On the other hand, when $\hat{P}_i < P_i$, the search precision (a quantitative measure for search accuracy, cf. section 5) drops below 100% while the rank privacy (a privacy measure. cf. section 5) increases.

Strategy 1 Based on this observation, our first efficiency enhancement aims to produce a better estimation of $\hat{P}_i$ that approximates to its ideal value P_i. We propose the following strategy to achieve this. During the index tree generation phase, the data owner retains a vector E_i for each level i. This vector consists of the maximum values at each dimension among all the indexes at this level. Subsequently, during the query generation phase, $\hat{P}_i$ is equal to the inner product of E_i and Q_i, and $\hat{P}_i$ will be set to 1 if it is greater than 1, thus $P_i \leq \hat{P}_i \leq 1$. $\hat{P}_i$ can be taken as an additional search parameter to be sent with $\widetilde{Q_i}$ to the cloud server. As for EMTS, we add the maximum $\sum_{j \in \bar{V}_i} \varepsilon_{i,j}$ from Q_i to $\hat{P}_i$, and refer to this sum as the final prediction threshold value. In Fig. 6, it is shown that the search efficiency is improved with this strategy, compared to the baseline search, in which $\hat{P}_i$ is always set to 1, the upper bound of cosine function.

4.3 Impact of Intended Keyword Position

Another factor we observed that affects the search efficiency is the position of the search keywords on the index tree. As shown in Tab. 2, the higher level the intended keywords reside, the higher the search efficiency. This is very different from using the MD-algorithm in database scenario where all the attributes are involved in searching the relevant objects. In the text search scenario, people are likely to complete a search with a query only comprising five keywords or less [1]. Consequently, the search algorithm needs to go through a larger number of nodes to evaluate an intended keyword if it resides at a lower level.

Strategy 2 The insight from this observation is that the average search time can be improved by strategically arranging keyword position in the index tree – the most frequently searched keywords on the top levels. In our experiment, we collected a set of 100 search requests from the volunteering users of this prototype secure search system. We then build the index tree where keywords are re-ordered by their search popularity. In practice, the information on the search keyword distribution can be extracted from the user search history. As shown in Fig. 6, the efficiency of the search algorithm is ameliorated significantly when applying this strategy.

Table 1: Impact of prediction threshold

$\hat{P}_i$	Time (ms)	# of accessed nodes	Precision	Rank privacy
1	33	17007	100%	0%
0.05	32	15012	100%	0%
0.02	28	14326	90.3%	6.5%
0.01	11	6410	10.6%	87.2%

Table 2: Impact of keyword position

Keyword position	Time (ms)	# of accessed nodes
1st	5	491
15th	32	13304
30th	33	23844
50th	36	49001

Table 3: Impact of clustering

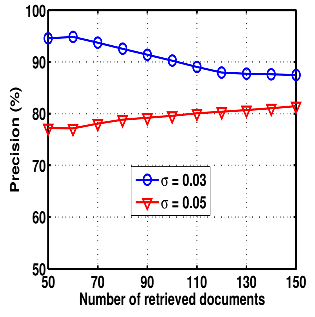
Ed	Time (ms)	# of accessed nodes	Precision	Rank privacy
0.02	7.4	7062	100%	0%
0.05	6.24	6728	97.00%	2.1%

4.4 Impact of Index Vector Clustering

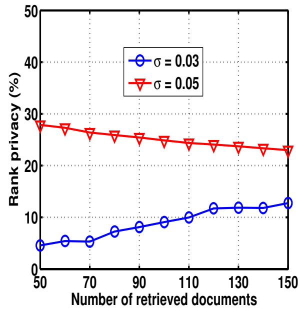
Another idea for improving the search efficiency is to cluster "similar" index vectors, as shown in Tab. 3. The improved efficiency comes from the reduced number of accessed nodes in the index tree, but at the expense of lower search precision. The bigger each cluster is, the higher the search efficiency, but the lower the search precision.

Strategy 3 To maximize the possibility of clustering, the length of the index vector at each level should be as short as possible (but at least achieve 80-bit symmetric key security [31]) in order to group the "similar" indexes. Inspired by the k-means method, which is the most widely used clustering technique in the data mining community [21], we use Euclidean distance (Ed) as a metric to cluster "close enough" vectors, e.g., when $Ed < 1$. For EMTS, we may first cluster original index vectors, and then execute dimension extension. The time cost for the search scheme after combining all the three efficiency-improving strategies, where $Ed = 0.02$, is shown in Fig. 6 as well.

Remarks The original combination of the MD-algorithm

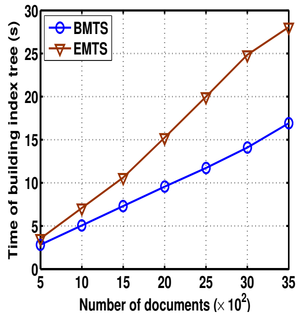


(a)

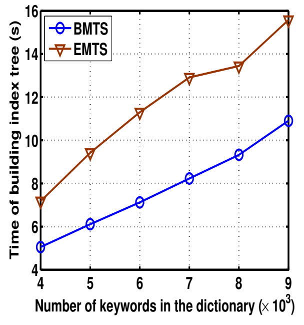


(b)

Figure 7: By choosing different standard deviation σ, the trade-off, between (a) Precision, and (b) Rank privacy, can be achieved.



(a)



(b)

Figure 8: Time cost for building index tree. (a) For the different size of document set with the same dictionary, $n = 4000$. (b) For the different size of dictionary with the same document set, $m = 1000$.

and the MDB-tree is not directly applicable for efficient search over vector indexes. From the extensive experiments on our prototype implementation of the search algorithm, we identified three efficiency-crucial factors and proposed effective strategies to improve the practical search efficiency with our vector indexes. Although the worst case search complexity is no better than linear search that is the state-of-the-art search efficiency in the multi-keyword encrypted text search scenario, this much less time-consuming tree-based search algorithm represents a solid step forward on the utilization of encrypted cloud data in practice. From the security point of view, the entire search process does not introduce new privacy vulnerability when used with our secure index scheme. In particular, our scheme is secure against search time analysis, i.e., the cloud server cannot infer specific keyword by the difference of search time, even if he/she knows that the keyword resides at a certain level. In fact, for efficiency, the cloud server performs level-selected similarity evaluation (see section 3), such that he/she has already possessed the keyword position information. There are at least 80 dimensions in an index vector at each level, and all the words falling into this level have almost the same search time. This effectively blinds one keyword within at least 79 other keywords at the same level. In addition, the cloud server has no knowledge about which concrete set of keywords are selected to build this level without the dictionary $\mathcal{T}$. Therefore, it is not possible to differentiate these keywords or identify a particular keyword of interest.

5. PERFORMANCE ANALYSIS

To evaluate the overall performance of our proposed techniques, we implemented the entire secure search system using JAVA on a Linux Server with Intel Core i3 Processor 3.3GHz. The document set is built from the recent ten years' IEEE INFOCOM publications, including about 3600 publications, from which we extract about 9000 keywords. In this section we present the detailed performance result. The documents and keywords used in the evaluation are selected randomly from the created document sets.

5.1 Precision and Privacy

To evaluate the impact on the accuracy of search result introduced by phantom terms in EMTS, we adopt the definition of "precision" in [5]. Namely, the "precision" of a top-k search is defined as $P_k = k'/k$ where k' is the number of the real top-k documents that are returned by the cloud server. Fig. 7(a) shows that with a small σ, the effectiveness of the search scheme is not affected much. The user can still enjoy almost the same search result as BMTS. On the other hand, we evaluate the "rank privacy" obtained from introducing phantom terms. The definition of "rank privacy" is also adopted from [5], i.e., the rank privacy at point k is calculated as $\widetilde{P_k} = \sum \widetilde{p_k}/k^2$. For every document d in the returned top-k documents, let the rank perturbation $\widetilde{p_k}$ be $|u_d - u_d{}'|$, where u_d is the rank number of document d in the returned top-k documents and it is set to k if greater than k, and $u_d{}'$ is its rank number in the real ranked documents. As shown in Fig. 7(b), large σ provides better protection of rank information in EMTS.

It is worth noting that σ is a tunable search parameter at the discretion of the user. The selection of different σ reflects his/her predilection for the better effectiveness of the search scheme or the better protected rank privacy and *keyword privacy* (see section 3.2).

Table 4: Size of index tree

Size of dictionary	2000	4000	6000	8000
BMTS (MB)	23.16	46.34	69.51	92.68
EMTS (MB)	26.87	53.77	80.66	107.54

5.2 Construction for Index Tree

Apparently, the time cost for generating the proposed index tree structure mainly depends on the size of the document set $\mathcal{DC}$ and dictionary $\mathcal{T}$. At a particular level i, the major computation is the encryption of the divided index vectors, which involves the splitting process and two multiplications of an $n_i \times n_i$ matrix and an n_i-dimensional vector where $n_i = |\mathcal{T}_i|$ in BMTS and $n_i = |\mathcal{T}_i| + U_i$ in EMTS. Fig. 8(a) shows that the time cost for building the index tree is nearly linear to the number of the documents, given the same dictionary, i.e., n_i is fixed at each level i when the tree structure has a predefined h levels. Fig. 8(b) shows that with the same document set, the index construction time is proportional to the number of keywords in the dictionary. Besides, building the index tree for EMTS is slightly more time-consuming than BMTS due to the dimension extension, which is shown in both Fig 8(a) and Fig 8(b). Note

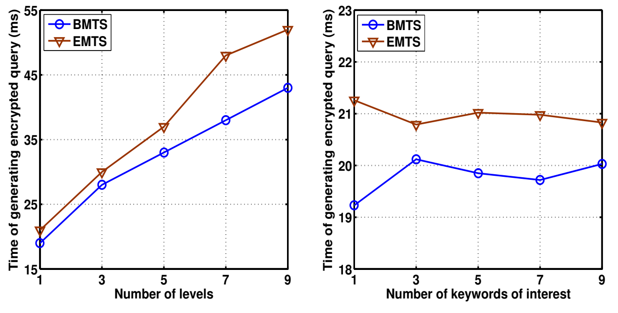


Figure 9: Time cost for generating encrypted query, when $|\mathcal{T}_i| = 80$. (a) For the different number of levels where user intended keywords reside. (b) For the different number of keywords of interest in one level.

that this computation is a one-time cost on the data owner side. On the other hand, the storage overhead, listed in Tab. 4, of the index tree with the fixed size of document set m=3000 is determined by the number of keywords in the dictionary. EMTS is very close to BMTS in the size of the index tree, and considering the massive storage capacity and low storage cost in the cloud, it is practical and completely affordable.

5.3 Query Generation

Typical search requests only consist of a few keywords [1]. Therefore, it is inefficient to generate a query that involves all the keywords in the dictionary, which not only makes the search process costly (similarity evaluation needs to be carried out on long vectors) but also brings some unnecessary computation overhead into query generation procedure. Fig. 9(a) demonstrates that when $|\mathcal{T}_i|$ is fixed, the time cost for generating an encrypted query is only linear to the number of levels where the searched keywords reside. Furthermore, the strategy to place the most frequently searched keywords on the top levels of the index tree will lead to the result that a good portion of queries are only generated for a few limited levels. As such, when $|\mathcal{T}_i|$ is chosen properly, the average query generation can be extremely efficient, as shown in Fig. 9(b). The major computation overhead stems from the vector encryption process, and due to the dimension extension, the time cost in EMTS is a bit more expensive than that in BMTS.

5.4 Search Efficiency

The search process executed at the cloud server is comprised of computing and ranking the similarity scores of relevant documents. The search algorithm terminates after the top-k documents have been selected. We evaluate the search efficiency of BMTS and EMTS with our proposed efficiency-improving strategies. Fig. 10(a) shows the search time for BMTS and EMTS, compared with [5] and baseline search with respect to the size of document set. In the baseline search, all the keywords in the dictionary are dispersed uniformly at random within the index tree, but it is still far efficient than [5] due to our proposed search algorithm and the tree-based index structure. It is noteworthy that the time cost of our proposed encrypted cloud data search is much more efficient than [5] and baseline search. In addition,

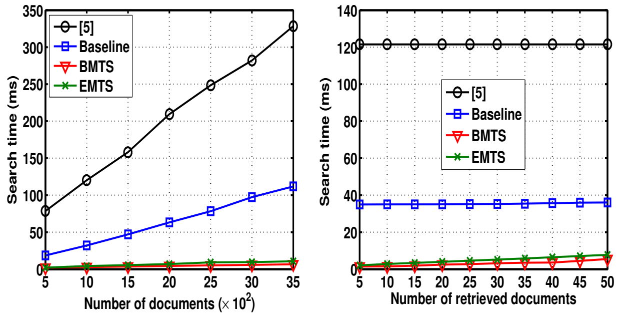


Figure 10: Search efficiency, with the same 10 keywords of interest. (a) For the different size of document set with the same dictionary, $n = 4000$, $k = 10$. (b) For the different number of retrieved documents with the same document set and dictionary, $m = 1000$, $n = 4000$.

with the increased size of document set, our two proposed schemes enjoy almost the same and nearly constant search time. Fig. 10(b) demonstrates that when user requests more relevant documents, our search algorithm is still extremely efficient.

6. RELATED WORK

6.1 Searchable Encryption with Single Keyword

Song et al. [27] propose the first SE scheme, where, to search a certain keyword in a document, user has to go through the whole document. After this work, many improvements and novel schemes [6, 10, 11, 16] have been proposed. Curtmola et al. [10] propose an inverted index based SE scheme with extremely efficient search process, but the *keyword privacy* will be revealed if the corresponding keywords have been searched. Frequency information is not involved in the similarity evaluation processes of the above techniques to provide accurate search functionality. In [28, 29, 34], the order-preserving techniques are utilized to protect the rank order. Due to the index and query built from frequency related information and the inverted index as the underlying index structure, they can achieve accurate and efficient search at the same time. Boneh et al. [3] propose the first PKC-based SE scheme, where anyone with public key can write to the data stored on server but only authorized users with private key can search. However, all of the aforementioned solutions only support single keyword search.

6.2 Searchable Encryption with Multiple Keywords

In the public key setting, a lot of works have been done to realize the conjunctive keyword search, subset search, or range queries [4, 12, 13], but they are too computationally intensive to be implemented for practical use. Predicate encryption is another promising technique to fulfill the search over encrypted data [2, 23, 25]. In [17], a logarithmic-time search scheme is presented to support range queries, which is orthogonal to our text search scenario. Furthermore, no similarity measure is adopted in these works to provide multi-keyword ranked search functionality. In text retrieval scenario, Pang et al. [20] also propose a vector space model based secure search scheme. An access manager is supposed to exist in their protocol except for a document server, and additional overhead occur on the user side, which is apparently not applicable to cloud environment. Without the security analysis for frequency information in their scheme, it is not clear whether such sensitive information disclosure could lead to *keyword privacy* infringement. Besides, the practical search performance is absent from the demonstration of their experiment. Cao et al. [5] propose a privacy-preserving multi-keyword ranked search scheme. Although with "coordinate matching", this scheme can produce the ranked search result by the number of matched keywords, more accurate ranked search result is not considered there, and the search complexity is constant in that the cloud server has to traverse all the indexes of the document set for each search request.

7. CONCLUSION

In this paper, as an initial attempt to achieve practical and effective multi-keyword text search over encrypted cloud data, we make contributions in two major aspects, supporting similarity-based ranking for more accurate search result and a tree-based search algorithm that achieves better-than-linear search efficiency. For the accuracy aspect, we first exploit the popular similarity measure, i.e., vector space model with cosine measure, to effectively procure the accurate search result. We propose two secure index schemes to meet various privacy requirements in the two threat models. Eventually, the leakage of sensitive frequency information can be avoided. To boost search efficiency, we propose a tree-based index structure for the whole document set. From the utilization of the prototype of our secure search system, we identify three essential efficiency-related factors, by which the efficiency of the search algorithm upon our index tree can be significantly improved. Finally, thorough analysis on the real-world document set demonstrates the performance of BMTS and EMTS in terms of search effectiveness, efficiency and privacy.

Acknowledgment We would like to thank Matúš Ondreička for his helpful comments and sharing the source code of MD-algorithm. This work was supported in part by the NSFC 61272457, the FRFCU K50511010001, the PCSIRT 1078, and the NSF grants CNS-1217889, CNS-1155988 and CNS-1218085.

8. REFERENCES

[1] Keyword and search engines statistics. `http://www.keyworddiscovery.com/keyword-stats.html?date=2013-01-01`, 2013.

[2] N. Attrapadung and B. Libert. Functional encryption for inner product: Achieving constant-size ciphertexts with adaptive security or support for negation. In *Proc. of PKC*, pages 384–402, 2010.

[3] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Proc. of EUROCRYPT*, pages 506–522, 2004.

[4] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *Proc. of TCC*, pages 535–554, 2007.

[5] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou. Privacy-preserving multi-keyword ranked search over encrypted cloud data. In *Proc. of IEEE INFOCOM*, pages 829–837, 2011.

[6] Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Proc. of ACNS*, pages 391–421, 2005.

[7] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of the ACM*, 45(6):965–981, 1998.

[8] Cloud Security Alliance. Security guidance for critical areas of focus in cloud computing v3.0. `http://www.cloudsecurityalliance.org`, 2011.

[9] D. Comer. Ubiquitous b-tree. *ACM computing surveys*, 11(2):121–137, 1979.

[10] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proc. of ACM CCS*, pages 79–88, 2006.

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

[12] P. Golle, J. Staddon, and B. R. Waters. Secure conjunctive keyword search over encrypted data. In *Proc. of ACNS*, pages 31–45, 2004.

[13] Y. Hwang and P. Lee. Public key encryption with conjunctive keyword search and its extension to a multi-user system. In *Pairing*, pages 2–22, 2007.

[14] B. Krebs. Payment processor breach may be largest ever. `http://voices.washingtonpost.com/securityfix/2009/01/payment_processor_breach_may_b.html`, 2009.

[15] M. Li, S. Yu, K. Ren, and W. Lou. Securing personal health records in cloud computing: Patient-centric and fine-grained data access control in multi-owner settings. In *Proc. of SecureComm*, pages 89–106, 2010.

[16] P. Liesdonk, S. Sedghi, J. Doumen, P. Hartel, and W. Jonker. Computationally efficient searchable symmetric encryption. *Secure Data Management*, pages 87–100, 2010.

[17] Y. Lu. Privacy-preserving logarithmic-time search on encrypted data in cloud. In *Proc. of NDSS*, 2012.

[18] NIST. NIST's dictionary of algorithms and data structures: inverted index. `http://xlinux.nist.gov/dads/HTML/invertedIndex.html`.

[19] M. Ondreička and J. Pokorný. Extending fagin's algorithm for more users based on multidimensional b-tree. In *Proc. of ADBIS*, pages 199–214, 2008.

[20] H. Pang, J. Shen, and R. Krishnan. Privacy-preserving similarity-based text retrieval. *ACM Transactions on Internet Technology*, 10(1):4, 2010.

[21] A. Rajaraman and J. D. Ullman. Mining of massive datasets. Cambridge University Press, Dec. 2011.

[22] P. Scheuermann and M. Ouksel. Multidimensional b-trees for associative searching in database systems. *Information systems*, 7(2):123–137, 1982.

[23] E. Shen, E. Shi, and B. Waters. Predicate privacy in encryption systems. In *Proc. of TCC*, pages 457–473, 2009.

[24] J. Sheridan and C. Cooper. Defending the cloud. `http://www.reactionpenetrationtesting.co.uk/Defending%20the%20Cloud%20v1.0.pdf`, 2012.

[25] E. Shi, J. Bethencourt, H. Chan, D. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *Proc. of S&P*, pages 350–364, 2007.

[26] Z. Slocum. Your google docs: Soon in search results? `http://news.cnet.com/8301-17939_109-1035713%207-2.html`, 2009.

[27] D. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Proc. of S&P*, pages 44–55, 2000.

[28] 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 2007 ACM Workshop on Storage Security and Survivability*, pages 7–12, 2007.

[29] C. Wang, N. Cao, K. Ren, and W. Lou. Enabling secure and efficient ranked keyword search over outsourced cloud data. *IEEE Transactions on Parallel and Distributed Systems*, 23(8):1467–1479, 2012.

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

[31] W. K. Wong, D. W. Cheung, B. Kao, and N. Mamoulis. Secure knn computation on encrypted databases. In *Proc. of SIGMOD*, pages 139–152, 2009.

[32] B. Yao, F. Li, and X. Xiao. Secure nearest neighbor revisited. `http://www.cs.utah.edu/~lifeifei/papers/snnicde.pdf`, to appear in *ICDE*, 2013.

[33] S. Yu, C. Wang, K. Ren, and W. Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *Proc. of IEEE INFOCOM*, pages 1–9, 2010.

[34] S. Zerr, D. Olmedilla, W. Nejdl, and W. Siberski. Zerber+r: Top-k retrieval from a confidential index. In *Proc. of EDBT*, pages 439–449, 2009.

[35] J. Zobel and A. Moffat. Exploring the similarity space. *SIGIR Forum*, 32(1):18–34, 1998.

APPENDIX

The Proposed Tree-based Search Algorithm

The following notations are used in the pseudo code of the algorithm:

- O – it is denoted as a subtree of the searched index tree, or as a document array residing in the leaf node. For the subtree O at the i^{th} level, it can be defined by the tree identifier $(\widetilde{D_{d,1}}, \widetilde{D_{d,2}}, ..., \widetilde{D_{d,i-1}})$.
- x – an encrypted document in a document array O. For simplicity, it could be a document ID.
- S_i – the similarity score that is equal to $Cos(\widetilde{D_{d,i}}, \widetilde{Q_i})$ as Eq. 2.
- $PF(\widetilde{D_{d,i}})$ – the function for predicting the maximum possible final similarity score from index vector $\widetilde{D_{d,i}}$. $PF(\widetilde{D_{d,i}}) = \sum_{t=1}^{i} S_t + \sum_{j=i+1}^{h} \hat{P}_j$, where h is the total number of levels of the index tree.
- $F(\cdot)$ – the final similarity score of a document, e.g., $F(x) = \sum_{i=1}^{h} S_i$.
- L_k – the list for selected top-k documents that are stored in descending order according to $F(x)$.
- DL_i – the list for index vectors $\widetilde{D_{d,i}}$'s stored in a node at the i^{th} level of a subtree of the index tree.
- M_k – the similarity score of the k^{th} document in L_k.

Algorithm 1 Proposed Search Algorithm for Top-k Ranking on Index Tree

```
begin
for ($i = 1 \to h$) do
    if ($\widetilde{Q_i}$ does not exist in the search request) then
        $\hat{P}_i \leftarrow 0$ and all $S_i \leftarrow 0$ at the $i^{th}$ level;
    else
        use the received $\hat{P}_i$ with the search request;
    end if
end for
findTopK($\mathcal{I}$, ($\emptyset$), $F(.)$, $k$);
return $L_k$;
end

procedure findTopK(IndexTree $\mathcal{I}$, Identifier ($\widetilde{D_{d,1}}$,
            $\widetilde{D_{d,2}}, ..., \widetilde{D_{d,i-1}}$), Score $F(\cdot)$, int $k$);
if ($O$ is a document array) then
    while (there is next document in $O$) do
        if ($|L_k| < k$) then
            insert document $x$ into $L_k$ according to $F(x)$;
        else
            if ($F(x) > M_k$) then
                delete $k^{th}$ document from $L_k$;
                insert $x$ into $L_k$ according to $F(x)$;
            end if
        end if
    end while
else
    $DL_i$ = getIndexList($\mathcal{I}$, ($\widetilde{D_{d,1}}, \widetilde{D_{d,2}}, ..., \widetilde{D_{d,i-1}}$));
    $counter \leftarrow 0$;
    while (there is next index vector in the subtree) do
        if ($|L_k| = k$ and $FP(DL_i(counter)) \le M_k$) then
            return;
        else
            findTopK($\mathcal{I}$, ($\widetilde{D_{d,1}}, ..., \widetilde{D_{d,i-1}}, \widetilde{D_{d,i}}$), $F(\cdot)$, $k$);
        end if
        $counter \leftarrow counter + 1$;
    end while
end if
end procedure

procedure getIndexList(IndexTree $\mathcal{I}$, Identifier ($\widetilde{D_{d,1}}$,
            $\widetilde{D_{d,2}}, ..., \widetilde{D_{d,i-1}}$));
if ($\widetilde{Q_i}$ exists in the search request) then
    for (each $\widetilde{D_{d,i}}$ within the node at the $i^{th}$ level in
        subtree of $\mathcal{I}$ with identifier ($\widetilde{D_{d,1}}, \widetilde{D_{d,2}}, ..., \widetilde{D_{d,i-1}}$))
do
        compute $S_i = Cos(\widetilde{D_{d,i}}, \widetilde{Q_i})$;
    end for
    sort $DL_i$ in descending order according to $S_i$;
    return $DL_i$;
else
    return the original $DL_i$;
end if
end procedure
```