Efficient Multi-Keyword Ranked Search Over Encrypted Data for Multi-Data-Owner Settings

Mahmoud Nabil*, Ahmad Alsharif*, Ahmed Sherif§, Mohamed Mahmoud* , Mohamed Younis**
*Department of Electrical and Computer Eng., Tennessee Tech. University, TN 38505
§Department of Electrical and Computer Eng., Tennessee State University, TN 37209
**Department of Computer Science and Electrical Eng., University of Maryland Baltimore County, MD 21250

***Abstract*—The availability of high-performance computing platforms, large storage devices, and high-speed communications have boosted the popularity of cloud computing. Users exploit these capabilities by using the cloud as a repository for their data and sharing these data with others. However, since the cloud is usually owned and operated by private companies, storing sensitive data in the cloud servers raises privacy concerns. To address these concerns, privacy-preserving keyword search schemes have been developed. In these schemes, data owners store encrypted files in the server, and users should send encrypted queries with keywords of interest. The server should use the encrypted queries to search the encrypted files and return to the user the files that have the queried keywords, without learning any information about the keywords or the contents of the files. Nevertheless, most of the existing schemes are either inefficient for multi-data-owner settings or designed for single-data-owner settings, and becomes insecure and inefficient when used for multi-data-owner. This paper proposes an efficient multi-keyword ranked search scheme over encrypted data for multi-data-owner settings. The proposed scheme allows each data owner and each user to have a distinct key, and allows the server to efficiently search the files of different data owners using one encrypted query sent by the user. Our privacy analysis demonstrates that the proposed scheme can preserve the privacy of the data owners and users. In addition, our extensive performance evaluations demonstrate that our scheme is much more efficient than existing approaches in the literature.**

***Index Terms*—Privacy preservation, multi-keyword ranked search over encrypted data, and cloud computing.**

I. Introduction

A massive amount of data is received and stored every day by cloud servers. It is expected that by 2020 around 35 ZettaBytes of digital data will be generated annually [1]. Therefore, the emerged cloud computing services such as infrastructures, platforms, and software can become mainstream technologies. In addition, the concept of the delivery of the information technology services (software or hardware) by a third-party over the internet is evolving every day. With cloud computing, users can utilize very large computational resources with limited investment and have an easy access that is not restricted to a location. Thus, around 70% of the U.S. organizations migrated partially to the cloud, 16% are planning for migration within a year, and 14% within three years [2]. Nevertheless, several security and privacy concerns have been raised.

The Cloud Security Alliance pointed 14 threats in the cloud computing model such as data leakage and hardware failure [3]. In addition, since the cloud is usually owned and operated by private companies, some data owners still doubt storing their private data remotely. Some examples of cloud breaches includes Apple's iCloud services leaks in 2014 [4] and Verizon data leak by Amazon Web cloud storage services in 2017 where millions of customers records were exposed [5]. Recent statistics show that over 3000 data breaches have occurred in 2017 [6].

Whilst data encryption is essential to preserve data owners' privacy, it overburdens searching and retrieving the stored files. Naive solutions like downloading and decrypting all stored files locally would diminish the utility of the cloud. Therefore, privacy-preserving keyword search schemes have been developed to enable servers to search over encrypted data [7]–[9]. In these schemes, data owners should store encrypted files on the server, whereas users should encrypt queries with keywords of interest and send these encrypted queries to the server. The server should use the encrypted queries to search the encrypted files and return the files that have the queried keywords to the user, without learning any information about the keywords or the contents of the files. However, sharing encrypted data on the cloud could be useless if it is not easily searched as plaintext data. Thus, the development of efficient and practical searchable encryption schemes is essential. *The main challenge is to efficiently process large amount of encrypted data and serve users' queries in a reasonable time.*

Efficient searchable encryption schemes have been studied extensively in literature [7]–[9]. These schemes are efficient as they use efficient cryptographic operations and only rely on dot product operations between the data owner's encrypted document (called *index*) and the user encrypted query (called *trapdoor*). Nevertheless, these schemes are designed for single-data-owner scenario, but in many applications, the data owners are more than one entitiy and users need to search all their documents. For instance, in case of a multi-department company, each department owns some documents on the server and a set of users are allowed to search these documents.

In addition, using the existing single-data-owner schemes in multi-data-owner scenario is either insecure or inefficient. In one approach for adapting existing single-data-owner schemes to handle the multi-data-owner scenario, each data owner should use a unique key, whereas each user should use one key for each data owner in the system. This approach is inefficient because in addition to the large key storage overhead on the users' side, users need to encrypt their queries multiple times and send these encrypted queries to enable the server to search each data owner documents. In another approach, all data owners share the same key so that users send only one encrypted query. Obviously, this is not secure since it cannot

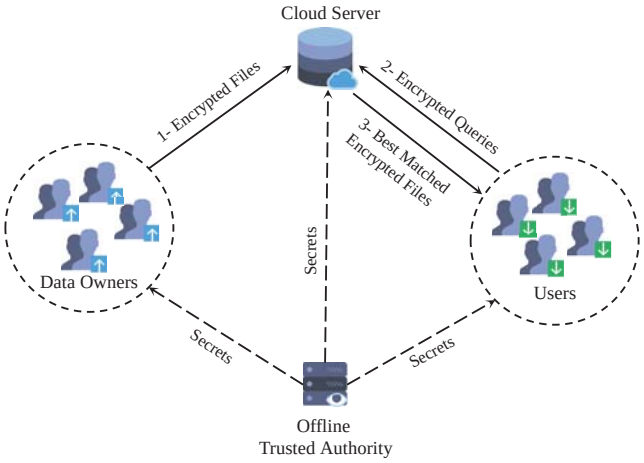


Figure 1: The considered network model.

prevent a data owner from decrypting other owners' data.

Recent schemes designed for multi-data-owenr settings [10], [11] lack both efficiency and scalability due to the high computations. Thus, in this paper, we opt to overcome the aforementioned issues and propose an efficient multi-keyword ranked search scheme over encrypted data for multi-data-owner and multi-user settings. The proposed scheme allows each data owner and each user to have a distinct key, and enables the server to search the owners' files using one encrypted query sent by the user. In addition, an issue in the existing schemes [7]–[9] is that anyone who obtains trapdoors and document indices, e.g., by eavesdropping the communications of users and data owners, can obtain side information about the similarity score of the queried keywords and the documents' keywords. This issue can be eliminated by using a symmetric key shared between the server and each data owner and user to encrypt the indices and trapdoors, however, more overhead is needed for key establishment/management and encryption. This issue is addressed in our scheme efficiently and only the server can do the similarity measurement between indicies and trapdoors. Our privacy analysis demonstrates that the proposed scheme can preserve the privacy of the data owners and users, and no party in the system can decrypt other party's indices. In addition, our extensive performance evaluations demonstrate that our scheme is much more efficient than existing schemes in the literature.

The remainder of this paper is organized as follows. The system models and design goals are discussed in Section II. The proposed scheme is explained in Section III. The privacy analysis and performance evaluation results are provided in Sections IV and V, respectively. Section VI surveys the related works. Finally, the paper is concluded in Section VII.

II. System Models and Design Goals

A. Network Model

As shown in Fig. 1, the considered network model consists of four main entities: data owners, users, cloud server, and an off-line trusted authority. The trusted authority should distribute a distinct secret key to the cloud server, each data owner, and each user. Data owners encrypt their documents and upload the encrypted documents to the cloud server to be shared with users. To download files of interest, users should create an encrypted query with keywords of interest and send the encrypted query to the server. The cloud server should use the encrypted query to search the encrypted documents by computing a similarity score between them, and then return the documents that have the highest scores to the user, without knowing any sensitive information about the queried keywords or the contents of the documents.

B. Threat Model

The attackers can be external eavesdroppers and internal, including, data owners, users, and the cloud server. The attackers are considered as "honest-but-curious", i.e., they execute the proposed scheme honestly and do not aim to disrupt the proper operation of the scheme, however, they are curious to infer sensitive information about the contents of the encrypted documents or the keywords of interest. Specifically, we consider the following two security models.

Known Ciphertext Model. In this model, the attacker has access to only the searchable encrypted document indices $\mathcal{I}$ and the trapdoors $\mathcal{I}_{\mathcal{T}}$, that are sent from data owners and users respectively.

Known Background Model. This is a stronger model, where the attacker has some background knowledge such as some statistical information about the documents or the correlation relationship among search queries. In this case, the attacker tries to use this information to infer some keywords.

C. Design Goals

The proposed privacy-preserving multi-keyword ranked search scheme should achieve the following design goals:

1) Multi-Keyword Ranked Search Over Encrypted Data for Multi-Data-Owner. The scheme should be able to use encrypted multi-keywords queries to calculate ranked search results over encrypted documents stored by different data owners.

2) Scalability and Efficiency. The scheme should be able to search a large number of encrypted documents in short time to quickly respond to users queries. In addition, the size of the trapdoors should be acceptable to reduce the communication overhead.

3) Index and Trapdoor Confidentiality. The cloud server should not be able to learn any useful information about the stored documents or trapdoors. In addition, analyzing the statistical information of the documents from the same domain should not help the server to identify any specific keyword in the query or the index.

4) Trapdoors Unlinkability. Given two trapdoors, the server should not ascertain whether the two trapdoors have the same set of keywords.

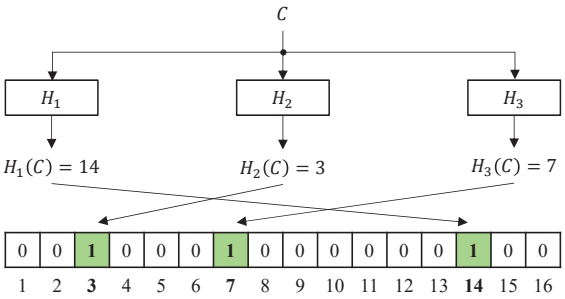


Figure 2: Adding element C to Bloom filter.

III. Proposed Multi-Keyword Ranked Search Scheme for Multi-Data-Owner Settings

In this section, we explain in details the proposed scheme. Table I gives the main notations used in the paper.

A. Initialization

In our scheme, Bloom filters are used to store the keywords compactly and protect against the attacks of known background model, as will be clarified in subsection IV-A. A Bloom filter is a probabilistic storage data structure with very minimal storing overhead [12]. It is represented by m bit vector that stores a set of $\mathcal{N}$ keywords, and it can determine efficiently and with high probability whether a queried keyword has been inserted before or not.

The trusted authority selects a set of ℓ independent keyed hash functions $H = \{h_i |\ h_i : C \rightarrow \mathfrak{n}_\mathfrak{i}\ ,\ 1 \leq i \leq \ell\ , and\, 1 \leq \mathfrak{n}_\mathfrak{i} \leq m\}$ to be used in the Bloom filter building algorithm and distributes this set to data owners and users. To insert keyword C in the filter, C is hashed using the set of keyed hash functions, and then the locations pointed by the hash values in a binary vector are set to one. Fig. 2 shows an example for inserting a keyword in a 16-bit Bloom filter using three hash functions. To determine whether a given keyword C' belongs to the stored keywords, C' is hashed using the set of keyed hash functions and if all the locations they point at are ones then with high probability C' was inserted before; otherwise, definitely C' was not inserted in the filter. The false-positive probability can be computed by $(1-(1-1/m)^{\ell\mathcal{N}})^\ell$ [12]. Obviously, increasing m can make the false-positive probability very small and thus negligible.

In addition, the trusted authority runs the following set of oracles sequentially to initialize the system:

1) $SystemSetup\,(1^m) \rightarrow \mathcal{TASK}$. The system setup algorithm takes the security parameter 1^m as an input and outputs the Trusted Authority Secret Key ($\mathcal{TASK}$), where, $\mathcal{TASK} = \{S, M_1, M_2, N_1, ..., N_8\}$, S is a random binary vector of length m, and a set of random invertible matrices $\in \mathbb{R}^{m\times m}$.

2) $KeyGenServer\,(\mathcal{TASK}) \rightarrow \mathcal{SSK}$. The trusted authority creates the Server Secret Key ($\mathcal{SSK}$), where $\mathcal{SSK} = X$, X is an invertible random matrix $\in \mathbb{R}^{m\times m}$.

3) $KeyGenDataOwner\,(\mathcal{O}_i, \mathcal{TASK}) \rightarrow \mathcal{DOSK}_i$. For each data owner in the system with identity (ID) $\mathcal{O}_i$, the trusted authority creates a Data Owner Secret Key ($\mathcal{DOSK}_i$) as

$$\mathcal{DOSK}_i = \{S,\ XN_1^{-1}A_i,\ XN_2^{-1}B_i,\ XN_3^{-1}A_i,\ XN_4^{-1}B_i,\ XN_5^{-1}C_i,\ XN_6^{-1}D_i,\ XN_7^{-1}C_i,\ XN_8^{-1}D_i\}$$

Table I: MAIN NOTATIONS

Notation	Description
m	Bloom filter size
ℓ	Number of hashes
$H = \{h_i \mid h_i : C \rightarrow \mathfrak{n}_\mathfrak{i}, 1 \leq i \leq \ell; 1 \leq \mathfrak{n}_\mathfrak{i} \leq m\}$	Bloom filter hash set
$\mathcal{TASK}$	Trusted authority secret key
S	Secret binary vector
$\{M_1, M_2, N_1, ..., N_8\} \in \mathbb{R}^{m\times m}$	Server secret matrices
$\mathcal{SSK}$	Server secret key
$\mathcal{O}_i$	Data owner i
$\mathcal{DOSK}_i$	$\mathcal{O}_i$'s secret key
$\{A_i, B_i, C_i, D_i\} \in \mathbb{R}^{m\times m}$	Random matrices for $\mathcal{O}_i$
$\mathcal{U}_x$	User x
$\mathcal{USK}_x$	$\mathcal{U}_x$'s secret key
$\{E_x, F_x, G_x, H_x\} \in \mathbb{R}^{m\times m}$	Random matrices for $\mathcal{U}_x$
$n_{\mathcal{O}_i}$	Number of $\mathcal{O}_i$'s documents
$D_i = \left\{d_{i,1}, d_{i,2}, \ldots, d_{i,n_{\mathcal{O}_i}}\right\}$	$\mathcal{O}_i$'s documents
$V_i = \left\{V_{i,1}, V_{i,2}, \ldots, V_{i,n_{\mathcal{O}_i}}\right\}$	$\mathcal{O}_i$'s documents' filters
$\mathcal{I}_i = \left\{I_{V_{i,1}}, I_{V_{i,2}} \ldots, I_{V_{i,n_{\mathcal{O}_i}}}\right\}$	$\mathcal{O}_i$'s indieces
$n_{\mathcal{U}_x}$	Number of $\mathcal{U}_x$'s queries
$Q_x = \left\{q_{x,1}, q_{x,2}, \ldots, q_{x,n_{\mathcal{U}_x}}\right\}$	$\mathcal{U}_x$'s queries
$T_x = \left\{T_{x,1}, T_{x,2}, \ldots, T_{x,n_{\mathcal{U}_x}}\right\}$	$\mathcal{U}_x$'s queries' filters
$\mathcal{I}_{\mathcal{T}x} = \{I_{T_{x,1}}, I_{T_{x,2}}, \ldots, I_{T_{i,n_{\mathcal{U}_x}}}\}$	$\mathcal{U}_x$'s trapdoors

where $\{A_i, B_i, C_i, D_i\}$ are random matrices $\in \mathbb{R}^{m\times m}$ such that $A_i + B_i = M_1^{-1}$, and $C_i + D_i = M_2^{-1}$.

4) $KeyGenSystemUser\,(\mathcal{U}_x, \mathcal{TASK}) \rightarrow \mathcal{USK}_x$. For user with ID $\mathcal{U}_x$, the trusted authority creates a User Secret Key ($\mathcal{USK}$) as follows.

$$\mathcal{USK}_x = \{S,\ E_xN_1,\ E_xN_2,\ F_xN_3,\ F_xN_4,\ G_xN_5,\ G_xN_6,\ H_xN_7,\ H_xN_8\}$$

where $\{E_x, F_x, G_x, H_x\}$ are random matrices $\in \mathbb{R}^{m\times m}$ such that $E_x + F_x = M_1$, and $G_x + H_x = M_2$

Moreover, for the users to be able to decrypt the documents, the trusted authority should distribute group keys shared between each data owner and users. The trusted authority should also distribute a key to the data owners/users to be used for the key hash functions H.

B. Encrypting Documents

Each data owner $\mathcal{O}_i$ builds a set of indices $\mathcal{I}_i = \left\{I_{V_{i,1}}, I_{V_{i,2}} \ldots, I_{V_{i,n_{\mathcal{O}_i}}}\right\}$ for his document set $D_i = \left\{d_{i,1}, d_{i,2}, \ldots, d_{i,n_{\mathcal{O}_i}}\right\}$ by invoking two oracles $BuildFilter()$ and $CreateIndex()$ and then sends $\mathcal{I}_i$ to the cloud server. Also, $\mathcal{O}_i$ should encrypt the documents with the group key shared with all users or those whom the owner allows to access the documents and send them to the server as well.

$BuildFilter\left(D_i = \left\{d_{i,1}, d_{i,2}, \ldots, d_{i,n_{\mathcal{O}_i}}\right\}\right) \rightarrow V_i = \{V_{i,1}, V_{i,2}, \ldots, V_{i,n_{\mathcal{O}_i}}\}$. For each document $d_{i,j}$ that has a set of keywords $\{w_{i,j,1}, w_{i,j,2}, \ldots\}$, and belongs to data owner $\mathcal{O}_i$, an m-bit Bloom filter vector $V_{i,j}$ should be built by setting the bits in $V_{i,j}$ corresponding to locations pointed by the

outputs of the hash functions on each keyword $w_{i,j,k}$. Note that, to allow the server to return ranked search results, the $V_{i,j}$ locations that are set to ones are replaced by the TF-IDF (Term Frequency - Inverse Document Frequency) [13] of each word in the document, which is computed as follows.

$$TF-IDF(w_{i,j,k}, d_{i,j}) = freq_{w_{i,j,k},d_{i,j}} * \log(\frac{N}{n_{w_{i,j,k}}})$$

where, $freq_{t,d_i}$ is the frequency of $w_{i,j,k}$ in $d_{i,j}$, N is the total number of vocabularies in all documents, and $n_{w_{i,j,k}}$ is the total number of the appearance of $w_{i,j,k}$ in D_i.

$CreateIndex\left(V_{i,j}, \mathcal{DOSK}_i\right) \rightarrow I_{V_{i,j}}$. For a Bloom filter vector $V_{i,j}$, $\mathcal{O}_i$ generates an index $I_{V_{i,j}}$ to be sent to the server. Then, $\mathcal{O}_i$ uses the secret S to split $V_{i,j}$ into two column vectors v'_{ij} and v''_{ij} of the same size, as follows. If the k^{th} bit of S is zero, $v'_{ij}(k)$ and $v''_{ij}(k)$ are set similar to $V_{ij}(k)$, while if it is one, $v'_{ij}(k)$ and $v''_{ij}(k)$ are set to two random numbers such that their summation is equal to $V_{i,j}$. Finally, $\mathcal{O}_i$ uses his secret key $\mathcal{DOSK}_i$ to compute the document index $I_{V_{i,j}}$ as

$$I_{V_{i,j}} = \Big[XN_1^{-1}A_iv'_{ij};\ XN_2^{-1}B_iv'_{ij};\ XN_3^{-1}A_iv'_{ij};\ XN_4^{-1}B_iv'_{ij};\ XN_5^{-1}C_iv''_{ij};\ XN_6^{-1}D_iv''_{ij};\ XN_7^{-1}C_iv''_{ij};\ XN_8^{-1}D_iv''_{ij}\Big]$$

where $I_{V_{i,j}}$ is a column vector of size $8m$.

C. Trapdoor Generation

Each user $\mathcal{U}_x$ generates encrypted search queries by invoking two oracles, called $BuildFilter()$ and $CreateTrapdoor()$, respectively.

$BuildFilter\left(Q_x = \{q_{i,1}, q_{i,2}, \ldots, q_{i,n_{\mathcal{U}_x}}\}\right) \rightarrow T_x = \{T_{x,1}, T_{x,2}, \ldots, T_{x,n_{\mathcal{U}_x}}\}$. For each query $q_{x,y} = \{w_{x,y,1}, w_{x,y,2}, \ldots\}$, $\mathcal{U}_x$ generates an m-bit Bloom filter $T_{x,y}$ by setting the bits in $T_{x,y}$ that are resulted from applying the hash set H on each keyword. Then, the user should invoke $CreateTrapdoor()$ oracle.

$CreateTrapdoor\left(T_{x,y}, \mathcal{USK}_x\right) \rightarrow I_{T_{x,y}}$. Given the bloom filter vector $T_{x,y}$, $\mathcal{U}_x$ uses S to split $T_{x,y}$ to two random row vectors t'_{xy} and t''_{xy} of the same size. The splitting method is described as follows. If the k^{th} bit of S is one, $t'_{xy}(k)$ and $t''_{xy}(k)$ are set similar to $T_{x,y}(k)$, while if it is zero, $t'_{xy}(k)$ and $t''_{xy}(k)$ are set to two random numbers such that their summation is equal to $T_{x,y}(k)$. Then, $\mathcal{U}_x$ uses his secret key $\mathcal{USK}_x$ to create the trapdoor $I_{T_{x,y}}$ as

$$I_{T_{x,y}} = \Big[t'_{xy}E_xN_1,\ t'_{xy}E_xN_2,\ t'_{xy}F_xN_3,\ t'_{xy}F_xN_4,\ t''_{xy}G_xN_5,\ t''_{xy}G_xN_6,\ t''_{xy}H_xN_7,\ t''_{xy}H_xN_8\Big]$$

where $I_{T_{x,y}}$ is a row vector of size $8m$.

D. Query Matching

The cloud server should use the trapdoors received from the users to measure its similarity with the documents' indices by invoking $Match()$ oracle. Then, the server sends back to the user the best matched encrypted documents.

$Match\left(\mathcal{SSK}, I_{T_{x,y}}, I_{V_{i,j}}\right) \rightarrow Score$. In this oracle, the cloud server should first use its secret X^{-1} to remove X from $I_{V_{i,j}}$ to obtain $I_{V_{i,j}}^-$, where

$$I_{V_{i,j}}^- = \Big[N_1^{-1}A_iv'_{ij};\ N_2^{-1}B_iv'_{ij};\ N_3^{-1}A_iv'_{ij};\ N_4^{-1}B_iv'_{ij};\ N_5^{-1}C_iv''_{ij};\ N_6^{-1}D_iv''_{ij};\ N_7^{-1}C_iv''_{ij};\ N_8^{-1}D_iv''_{ij}\Big]$$

Then, it can simply compute the similarity score between the trapdoor $I_{T_{x,y}}$ and the index $I_{V_{i,j}}^-$ by dot product operation $(I_{T_{x,y}} \cdot I_{V_{i,j}}^-)$, where $\cdot$ denotes dot product.

Theorem 1. *Using our scheme, the cloud server can measure the similarity score of the indices and the trapdoors.*

Proof: This can be done by computing $I_{T_{x,y}} \cdot I_{V_{i,j}}$ of a document filter $V_{i,j}$ and a query filter $T_{x,y}$, as follows.

$$\begin{aligned}
I_{T_{x,y}} \cdot I_{V_{i,j}}^- &= t'_{xy}E_xA_iv'_{ij} + t'_{xy}E_xB_iv'_{ij} + t'_{xy}F_xA_iv'_{ij} + \\
&\quad t'_{xy}F_xB_iv'_{ij} + t''_{xy}G_xC_iv''_{ij} + t''_{xy}G_xD_iv''_{ij} + \\
&\quad t''_{xy}H_xC_iv''_{ij} + t''_{xy}H_xD_iv''_{ij} \\
&= t'_{xy}(E_x + F_x).(A_i + B_i)v'_{ij} + \\
&\quad t'_{xy}(G_x + H_x).(C_i + D_i)v'_{ij} \\
&= t'_{xy}M_1M_1^{-1}v'_{ij} + t'_{xy}M_2M_2^{-1}v'_{ij} \\
&= T_{x,y} \cdot V_{i,j} \\
&= Score\left(T_{x,y} \cdot V_{i,j}\right)
\end{aligned}$$

■

IV. Privacy Analysis

A. Known Ciphertext and Known Background Models

We use the same security model of [14] to prove the security of our scheme. Basically, the cloud server should not be able to infer any information more than the access and the search patterns.

Theorem 2. *The proposed scheme is secure under both known ciphertext and known background models.*

Proof: The proof of this theorem is as follows.

History. For a set of documents D, the history is a set of indices $\mathcal{I} = \{I_{V_1}, \ldots, I_{V_n}\}$ over D and a set of trapdoor queries $\mathcal{I}_\mathcal{T} = \{I_{T_1}, I_{T_2}, \ldots, I_{T_k}\}$, denoted as $H = (\mathcal{I},\ \mathcal{I}_\mathcal{T})$.

Trace. A trace reflects the knowledge inferred by the cloud sever over the history H, denoted as $Tr(H)$, such as the search and access patterns, where $Tr(H)$ is defined over all the queries of H such as $Tr(H) = \{Tr(I_{T_1}), \ldots, Tr(I_{T_k})\}$

View. It represents the perception of the cloud server. It is the combination of the encrypted history and its trace, denoted as $V(Enc_{sk}(I), Enc_{sk}(W), Tr(H))$.

Consider a simulator $\mathcal{S}$ that can generate a false view V' that is indistinguishable from V by doing the following steps.

1) $\mathcal{S}$ calls *SystemSetup*(1^m) oracle to get a secret key $sk' = \mathcal{TASK}'$.
2) $\mathcal{S}$ generates a set of random documents $D' = \{d'_1, \ldots, d'_n\}$ such that $|d_i| = |d'_i|$, $1 \leq i \leq n$, and $d'_i = \{w'_1, w'_2, \ldots\}$.

3) $\mathcal{S}$ generates a set of trapdoors $\mathcal{I_T}' = \{I'_{T_1}, I'_{T_2}, \ldots, I'_{T_k}\}$, where $I'_{T_j} \in \{0,1\}^m$ is as follows.
-For each keyword $w_i \in W$, if $w_i \in \mathcal{I}_{\mathcal{T}j}$ and $1 \leq j \leq k$, add w'_i to $\mathcal{I}_{\mathcal{T}j}'$. Note that $\mathcal{I}_{\mathcal{T}j}'$ is a random copy of $\mathcal{I}_{\mathcal{T}j}$.
4) $\mathcal{S}$ generates a set of m-bit zero vectors denoted as indices $\mathcal{I}' = \{I'_{V_1}, \ldots, I'_{V_n}\}$ as follows:
-For each keyword $w_i \in W$, if $w_i \subset d_j$ and $1 \leq j \leq n$, add w'_i to I'_{V_j}. Note that I'_{V_j} is a random copy of I_{V_j} .
5) $\mathcal{S}$ generates encrypted index $Enc_{sk'}(\mathcal{I}')$ and trapdoor $Enc_{sk'}(\mathcal{I_T}')$ using secret sk'.

From the previous construction $\mathcal{S}$ has a history $H' = (\mathcal{I}', \mathcal{I_T}')$ with trace $Tr(H')$ similar to $Tr(H)$ such that in no probabilistic polynomial-time (P.P.T.), adversary can distinguish between the two views $V(Enc_{sk}(\mathcal{I}), Enc_{sk}(\mathcal{I_T}), Tr(H))$ and $V'(Enc_{sk'}(\mathcal{I}'), Enc_{sk'}(\mathcal{I_T}'), Tr(H'))$ with non-negligible advantage where the correctness of the construction implies this conclusion. In particular, the indistinguishability follows directly from the semantic security of kNN encryption scheme.

Moreover, for the known background model, the ℓ keyed hashes used in the construction of the Bloom filter can prevent the cloud server from obtaining any statistical information, such as keyword frequencies, and any information related to the domain of the data. ■

B. Trapdoors Unlinkability

In multi-data-owner scheme proposed in [10], called SRMSM, sending the same trapdoor in different occasions results in the same ciphertext, which violates the trapdoor unlinkability requirement. On the contrary, in our scheme, sending the same trapdoor in different occasions guarantees different ciphertexts. Another advantage of the proposed scheme over SRMSM is that, even if an eavesdropper gets the messages communicated with the server, $\mathcal{I} = \{I_{V_1}, \ldots, I_{V_n}\}$ and $\mathcal{I_T} = \{I_{T_1}, I_{T_2}, \ldots, I_{T_k}\}$, he cannot measure the similarity scores because the server secret key $\mathcal{SSK}$ is needed.

V. Performance Evaluations

A. Experiment Setup and Metrics

1) Experiment Setup: To evaluate the proposed scheme, we implemented it using Python and a server with an Intel® Xeon® Processor E5-2420 @2.2GHZ (2 processors) and 32 GB RAM. In our experiments, we used Request For Comments (RFC) dataset of size 5089 files [15]. We processed the dataset using *scikit-learn* Python library [16] and extracted the top 60 TF-IDF scored keywords. The computation and the communication overhead of the proposed scheme are compared to SRMSM. In addition, we used *Charm* cryptographic library [17] to measure the execution times of the cryptographic operations used in both schemes. In our scheme, the Bloom filter size is set to 592 $bits$ to make the false positive probability less than 0.01 in case of adding 60 keywords. All the experiments were run 1000 times and average results are reported.

Table II: Computation Overhead.

	Operation	Time
Our Scheme	T_{KeyGen}	0.06 sec
	T_{Dot}	4.4 μsec
	T_{Bh}	9 μsec
	T_{Enc}	6.63 $msec$
SRMSM [10]	T_{Pm}	1.43 μsec
	T_{Exp}	0.217 $msec$
	T_h	0.13 $\mu sec/Byte$
	T_{Add}	0.2 μsec

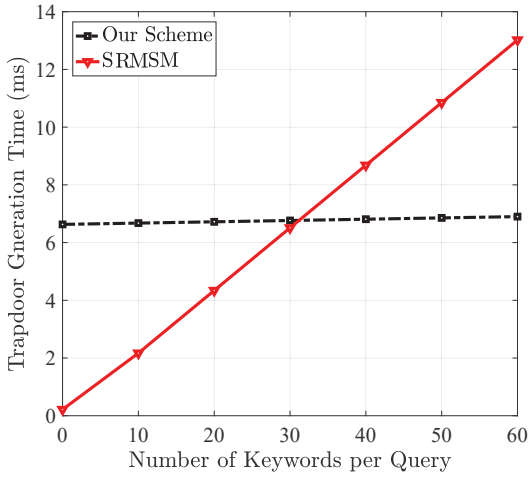


Figure 3: Trapdoor generation time (ms)

2) Performance Metrics: Three performance metrics are used for comparison and assessment of our scheme.

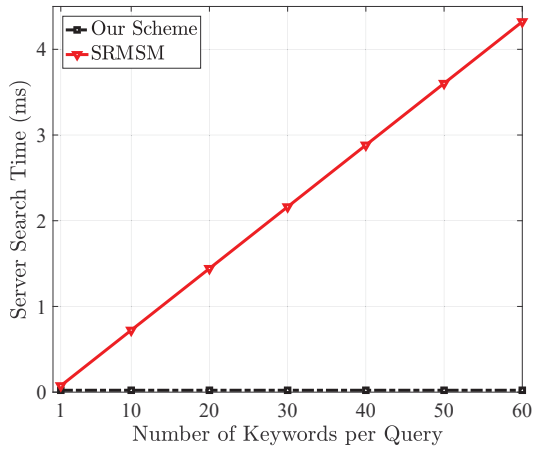
(1) *Average search time.* The time needed by the cloud server to search the documents.

(2) *Computation overhead.* The time needed by users to compute a trapdoor to be sent to the server.

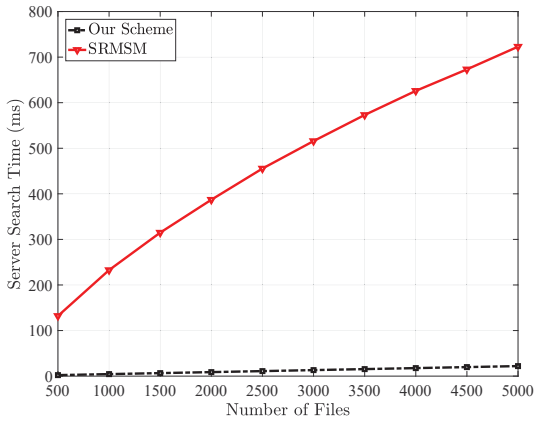
(3) *Communication overhead.* The amount of data transmitted during the communication between the users and the server.

B. Experiment Results

Table II gives the times required for the cryptographic operations used in SRMSM and our scheme. For our scheme, T_{KeyGen}, T_{Dot}, T_{Bh}, and T_{Enc} denote the times required for single key generation, indices and trapdoors dot product, Bloom Filter single hash operation, and encryption time, respectively. For SRMSM, T_{Pm}, T_h, T_{Exp}, and T_{Add} denote the times required for group points multiplication, single hash operation, modular exponentiation, and score addition, respectively. We denote V, Q, and D as the total number of vocabularies in the dataset, the number of keywords per query, and the number of documents respectively. In our evaluations, all computations that can be done off-line, such as uploading the encrypted documents to the server, are excluded from the comparison.



(a) Server search time as the number of keywords per query increases. $D = 5000$



(b) Server search time as the number of files increases. $Q = 10$

Figure 4: Search Time for our scheme versus SRMSM

1) Computation Overhead: Fig. 3 gives the time needed to generate a trapdoor versus the number of keywords per query. In SRMSM, each keyword is hashed first then mapped to an element in a finite field and used as exponent for the group generator, with a total cost of $Q(T_h + T_{Exp})$. In our scheme, each keyword is hashed ℓ times to generate a Bloom filter vector, and then this vector which has a fixed size of m bits is encrypted as explained in III-C. Therefore, our trapdoor generation time can be formulated as $Q\ell T_{Bh}+T_{Enc}$. As shown in the figure, the trapdoor generation time of both schemes grow linearly as the number of keywords per query increases, however, the growth rate of our scheme is much slower than that of SRMSM.

For the server search time in SRMSM, the server should search an inverted index matrix sent by each data owner where the rows of the matrix encode the keywords to points in group and the columns represent the score of this keyword in each document of the data-owner dataset. Thus, the server has to search over two stages namely: locate keywords, and rank relevance score. In the first stage, the server has to perform VQ group point multiplications, while in the second stage, the server has to perform D multiplied by the number of matched keywords q' relevance score additions. Therefore, the total search time for SRMSM can be formulated as $VQT_{Pm} + (q' - 1)DT_{Add}$ to search over each data owner documents. In our scheme, the server needs only to compute the dot product between a trapdoor and each document index regardless of the total number of vocabularies V. Therefore, the total search time in our scheme is DT_{dot}.

Using the measurements given in Table II, we compare in Fig. 4 the server search time for our scheme and SRMSM. As shown in Fig 4(a), the search time of SRMSM exhibits a linear rate as the number of keywords per query increases given that the number of files $D = 5000$ and at most $q' = 3$ matched keywords during the locate keywords stage. On the other hand, our scheme exhibits a constant rate as the number of keywords per query increases. In addition, Fig. 4(b) gives the server search time versus the number of documents. As shown in the figure, the search time of both schemes increase with the number of documents, however, SRMSM has a much faster rate because as the number of documents (D) increases, the total number of vocabularies (V) increases accordingly which results in more time required for locating keywords in SRMSM. Therefore, our scheme outperforms SRMSM in terms of total computation complexity.

2) Communication Overhead: For SRMSM scheme, each keyword in the query is represented as a point in an integer group of size of $1024\,bits$. Therefore, the size of the trapdoor in SRMSM grows linearly with the number of keywords in query. In our scheme, the size of the trapdoor is constant and depends on the size of the encrypted Bloom filter which is $9.5\ KBytes$. Our scheme outperforms SRMSM in the communication overhead when the number of keywords is large. Moreover, in SRMSM, the server can know the number of keywords in the queries since each keyword is mapped to a group point, while in the our scheme, this side information is completely hidden.

VI. Related Work

Various schemes have been proposed to enable searching over encrypted data. The first provably secure scheme was introduced by Song et al. [18]. The scheme relies on a symmetric key using a 2-layered encryption model. Later, different symmetric searchable encryption (SSE) schemes have been proposed. In [19], Goh proposed an efficient SSE scheme and devised a formal security model. Boneh et al [20], introduced the concept of public key encryption with keyword search (PEKS). The proposed approach is based on Identity Based Encryption (IBE) cryptography. Overall, these preliminary schemes do not support ranked search and complex queries with multiple keywords and logical operators.

Boolean search schemes that support logical operators, such as conjunctions and disjunctions, were presented in [21]–[23]. In [21] a conjunctive encrypted keyword search scheme was developed where users can search with a complete set of keywords. Disjunctive encrypted keywords search is proposed in [22] where users can search with a subset of keywords. Then, generalized boolean search schemes that support both conjunction and disjunction operators were discussed in [23],

[24]. Most of these boolean search schemes are limited to certain applications as they do not support ranking capabilities.

Ranked search [25], [26] was introduced to imitate the unencrypted search scenarios in real world applications, in which top-k relevant results are provided for a user query. Cao. et al. [26] proposed a privacy preserving multi-keyword ranked search over encrypted data (MRSE) for single data owner where the k-nearest-neighbor (kNN) scheme is used in similarity matching. Recently, Fu et al. [27] have proposed a multi-keyword fuzzy search that can tolerate spelling mistakes in the user input by utilizing locality sensitive hashing and stemming techniques. However, these schemes were developed for single-data-owner scenarios and it is either insecure or inefficient to use them in multi-data-owner settings.

For multi-data-owner ranked search, Zhang. et al. [10] proposed a ranked multi-keyword search scheme based on additive order and privacy preserving function that allows the data owners to securely send the keyword scores for each document to the server. The scheme has two stages: locate keywords and score calculation. To match a trapdoor, the server has to compare each encrypted keyword in the trapdoor with all the vocabularies of each data owner during the first stage, which increases the computation dramatically. Then, the server has to add all the document scores of all the matched keywords for comparison. Moreover, in this scheme query unlinkability cannot be achieved because encrypting the same query at different times results in the same ciphertext. Our proposed scheme overcomes these shortcoming.

VII. Conclusion

Most of the existing privacy-preserving multi-keyword search schemes focus on single-data-owner scenario, and using these schemes in multi-data-owner settings is either insecure or inefficient. In this paper, we have proposed an efficient multi-keyword search scheme over encrypted data for multi-data-owner settings. The proposed scheme allows each data owner and each user to have a distinct key, and allows the server to search the files of the different data owners using one encrypted query sent by the users. Our privacy analysis has demonstrated that the proposed scheme is more secure than the existing schemes and can preserve the privacy of data owners and users in both known ciphertext and known background models. Moreover, our extensive performance evaluations have demonstrated that our scheme is much more efficient than the existing approaches, especially in the search time.

References

[1] D. Karr, "Big data brings marketing big numbers," *https://martech.zone/ibm-big-data-marketing/*, 2017.

[2] I. Enterprise, "Cloud computing survey," *Executive Summary,*, pp. 1–8, 2016.

[3] C. S. Alliance, "Security guidance for critical areas of focus in cloud computing," *Report,*, pp. 1–177, 2011.

[4] K. M. Brady, "Revenge in modern times: The necessity of a federal law criminalizing revenge porn," *Hastings Women's LJ*, vol. 28, p. 3, 2017.

[5] D. Deahl, "Verizon data breach," *https://www.theverge.com/2017/7/12/15962520/verizon-nice-systems-data-breach-exposes-millions-customer-records*, 2017.

[6] "Cyber risk analytics," *https://www.cyberriskanalytics.com/#statistics*, 2017.

[7] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-Preserving Multi-keyword Ranked Search over Encrypted Cloud Data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 222–233, jan 2014.

[8] W. K. Wong, D. W.-L. Cheung, B. Kao, and N. Mamoulis, "Secure kNN Computation on Encrypted Databases," *Proceddings of the 35th SIGMOD International Conference on Management of Data*, pp. 139–152, 2009.

[9] C. Yang, W. Zhang, J. Xu, J. Xu, and N. Yu, "A fast privacy-preserving multi-keyword search scheme on cloud data," *Proceedings of the 2012 International Conference on Cloud Computing and Service Computing, CSC 2012*, pp. 104–110, 2012.

[10] W. Zhang, S. Xiao, Y. Lin, T. Zhou, and S. Zhou, "Secure ranked multi-keyword search for multiple data owners in cloud computing," in *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*. IEEE, 2014, pp. 276–286.

[11] W. Zhang, Y. Lin, S. Xiao, J. Wu, and S. Zhou, "Privacy preserving ranked multi-keyword search for multiple data owners in cloud computing," *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1566–1577, 2016.

[12] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[13] J. Beel, B. Gipp, S. Langer, and C. Breitinger, "paper recommender systems: a literature survey," *International Journal on Digital Libraries*, vol. 17, no. 4, pp. 305–338, 2016.

[14] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," *Journal of Computer Security*, vol. 19, no. 5, pp. 895–934, 2011.

[15] "Request for comments database," *http://www.ietf.org/rfc.html*, 2017.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[17] J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin, "Charm: a framework for rapidly prototyping cryptosystems," *Journal of Cryptographic Engineering*, vol. 3, no. 2, pp. 111–128, 2013.

[18] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*. IEEE, 2000, pp. 44–55.

[19] Goh and E. Jin, "Secure indexes." *IACR Cryptology ePrint Archive*, vol. 2003, p. 216, 2003.

[20] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Eurocrypt*, vol. 3027. Springer, 2004, pp. 506–522.

[21] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *ACNS*, vol. 4. Springer, 2004, pp. 31–45.

[22] B. Zhang and F. Zhang, "An efficient public key encryption with conjunctive-subset keywords search," *Journal of Network and Computer Applications*, vol. 34, no. 1, pp. 262–267, 2011.

[23] T. Okamoto and K. Takashima, "Hierarchical predicate encryption for inner-products." in *Asiacrypt*, vol. 5912. Springer, 2009, pp. 214–231.

[24] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," *Advances in Cryptology–EUROCRYPT 2008*, pp. 146–162, 2008.

[25] 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 *Proceedings of the 2007 ACM workshop on Storage security and survivability*. ACM, 2007, pp. 7–12.

[26] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Transactions on parallel and distributed systems*, vol. 25, no. 1, pp. 222–233, 2014.

[27] Z. Fu, X. Wu, C. Guan, X. Sun, and K. Ren, "Toward Efficient Multi-Keyword Fuzzy Search over Encrypted Outsourced Data with Accuracy Improvement," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 12, pp. 2706–2716, 2016. [Online]. Available: http://millenniumsoftsol.com/courses/IEEETitles/Java/Kalai-Toward-Efficient.pdf