Multi-client Secure Encrypted Search Using Searching Adversarial Networks

Kai Chen[1], Zhongrui Lin[2], Jian Wan[2], Lei Xu[1], and Chungen Xu[1](✉)

[1] School of Science, Nanjing University of Science and Technology, Nanjing, CHN
[2] School of Computer Science and Engineering, NJUST, Nanjing, CHN
{kaichen,xuchung}@njust.edu.cn

Abstract. With the rapid development of cloud computing, searchable encryption for multiple data owners model (multi-owner model) draws much attention as it enables data users to perform searches on encrypted cloud data outsourced by multiple data owners. However, there are still some issues yet to be solved nowadays, such as precise query, fast query, dimension disaster and flexible system dynamic maintenance. To target these issues, this paper proposes a secure and efficient multi-keyword ranked search over encrypted cloud data for multi-owner model based on searching adversarial networks (MRSM_SAN). Specifically, we exploit searching adversarial networks to achieve optimal pseudo-keyword filling, and obtains the optimal game equilibrium for query precision and privacy protection strength. In order to achieve fast query, maximum likelihood search balanced tree is proposed, which brings the query complexity closer to $O(\log N)$. we reduce data dimension with fast index clustering, and enable low-overhead system maintenance based on balanced index forest. In addition, attribute based encryption is used to achieve more secure and convenient key management as well as authorized access control. Compared with previous work, our solution maintains query precision above 95% while ensuring adequate privacy protection, significantly improving search efficiency, enabling more flexible system dynamic maintenance, and reducing the overhead on computation and storage.

Keywords: Searchable Encryption· Multi-keyword Ranked Search· Multi-owner Model· Searching Adversarial Networks· Maximum Likelihood Search Balanced Tree· Balanced Index Forest

1 Introduction

Background and Motivation: The rapid development of cloud computing has brought great convenience to the scientific allocation and efficient use of computing resources, creating more possibilities for the internet industry. Cloud storage is an important part of cloud computing, not only reduces the burden of local storage resources, but also provides the cloud data management platform for many information technology services, which significantly improves the efficiency of system operation and maintenance. Currently, cloud storage services are increasingly attracting individuals and enterprises to outsource data into cloud servers. But traditional data outsourcing has the potential to leak private information to the "honest but curious" cloud servers [29,34], which poses a serious threat to privacy protection. In order to solve the privacy protection problem of outsourced data and improve cloud security, academia and industry have been working hard [19].

Previous Works and Challenges: *Searchable encryption* technology is recognized as the effective mean to solve this problem because it can perform searches on encrypted cloud data outsourced by data owners [17]. Song et al. [26] proposed the first *searchable symmetric encryption* scheme, and advanced security definitions and improvements were given by Goh [10], Chang et al. [6], and Curtmola et al. [7]. Boneh et al. proposed public key encryption with keyword search [2]. Until now, more feature-rich solutions were proposed: *Boolean search* [1,3], *single keyword ranked search* [28], *multi-keyword ranked search* [5,22,27], *fuzzy search* [9,21], *authorized search* [20], *verifiable search* [16], *personalized search* [8] and *dynamic search* [31]. However, the above solutions only support searchable encryption for single data owner model. Due to the diverse demand of the application scenario, such as emerging authorised searchable technology for multi-client (authority) encrypted medical databases that focuses on privacy protection [32,33], research on searchable encryption technology for multiple data owners model (multi-owner model) is increasingly active, and effective searchable encryption schemes for the multi-owner model were proposed [14,20,23,35]. Currently, in secure cloud storage, searchable encryption technology for single owner model is relatively mature, but multi-owner model still face some urgent and unresolved problems: **(1)** a large amount of different data from data owners make data characteristics to be sparse, which is likely to bring "dimension disaster"; **(2)** when accessing and querying encrypted cloud data, query precision and query speed are difficult to satisfy the user experience; **(3)** frequent updates of data challenge the dynamic maintenance and scalability of the system. To meet these challenges, we first analyze the root cause of the problem: limited computing power and rationality of system design will affect the performance of the system, but the most critical factor is that the dimension of data to be processed is too large. If we can significantly reduce the dimension of data processing, the system will become stronger and have higher performance.

Existing Solutions and Their Shortcomings: Cao et al. [5] first proposed privacy-preserving multi-keyword ranked search over encrypted cloud data for single owner model (MRSE), and established strict privacy requirements. They first used *asymmetric scalar-product preserving encryption* (ASPE) approach [30] to obtain the similarity scores of the query vector and the index vector, so that the cloud server can return the *top-k* documents. However, they did not provide the optimal balance of query precision and privacy protection. For better query precision, Sun et al. [27] proposed MTS with the $TF \times IDF$ keyword weight model. The keyword weight depends on the frequency of the keyword in the document and the ratio of the documents containing this keyword to the total documents, which means that this method cannot handle the differences between data from different owners in multi-owner model. Since each owner's data is different, there is no uniform standard to measure keyword weights. Based on MRSE, Li et al. [22] proposed a better solution (MKQE), where a new index construction algorithm and trapdoor generation algorithm are designed to realize the dynamic expansion of the keyword dictionary and improve the system performance. However, there is still no major breakthrough in improving search efficiency. For flexible *dynamic search*, Xia et al. [31] provided EDMRS to support dynamic operation in multi-keyword ranked search. For tree-based index structures, search efficiency is improved by the greedy depth-first search (GDFS) algorithm and parallel computing. However, when migrating to multi-owner model, ordinary balanced binary tree they employed is not optimistic [14]. Zhang et al. [35] firstly implemented secure multi-keyword ranked search over encrypted cloud data scheme for multi-owner model. They utilized the modular exponentiation to encrypt the keywords of documents and queries, so that all owners can use their own keys to encrypt the index without having to generate multiple trapdoors for the query, but this produces significant computational overhead and communication costs. Guo et al. [14] designed a heuristic weight generation algorithm based on the relationships among keywords, documents and owners (KDO). They considered the correlation among documents and the impact of documents' quality on search results, their scheme (MKRS_MO) is better than the schemes using traditional $TF \times IDF$ keyword weight model [27]. Last but not least, the *trapdoor unlinkability* cannot be completely protected in their scheme, and they did not provide a secure solution in *known background model* [5] (see Section 2.3), therefore, the security of their scheme is insufficient.

Our Contributions: This paper first proposes a secure and efficient multi-keyword ranked search over encrypted cloud data for multi-owner model based on searching adversarial networks (MRSM_SAN). Specifically, including the following four core techniques. **(1) Optimal pseudo-keyword filling based on searching adversarial networks.** To improve the privacy protection strength of encrypted cloud data is a top priority. The current popular method is to fill random noise into the data (filling the pseudo-keyword into the index vector and the query vector) to interfere with the analysis and evaluation that are from the cloud server, which protects the document content and keyword information better. However, such an operation will reduce the query precision [5]. In response to this problem, we creatively use the *searching adversarial networks* to obtain the *optimal game equilibrium* for the query precision and the privacy protection strength, and obtain the *optimal probability distribution function* for controlling pseudo-keyword filling, so that the query precision exceeds 95%, is higher than MRSE [5] while ensuring adequate privacy protection. **(2) Fast query based on maximum likelihood search balanced tree.** The construction of the index tree is the biggest factor affecting the search time. If the index tree is ordered in the signification of maximum probability (the ranking of the index vectors from high to low depends on the probability of being searched), the searching algorithm complexity will be very close to $O(\log N)$ [18]. Our method is to perform 10000 random searches, get the sum of the matching scores of each index vector and all random query vectors, and then sort the index vectors according to the score from high to low. Follow the bottom-up strategy and build the balanced index tree based on the "greedy" method. We named it as the *maximum likelihood search balanced tree*. The experiments based on real-world data prove that by executing the greedy depth-first search (GDFS) algorithm, our query speed is faster and more stable than the related works that use tree-based index for searching (MKRS_MO [14],EDMRS [31]). **(3) Data dimension reduction method based on fast index clustering.** "Dimension disaster" exacerbates the computational burden of the system. In multi-owner model, data from different data owners can be vary widely. As the amount of data increases, the data characteristics become very sparse, which makes the keyword dictionary for index construction very large (in our experiments on the real-world data set, the keyword dictionary contains about 80,000 different keywords), which may lead to "dimension disaster" in computing. For all index vectors from different owners, we group them into different categories according to whether they contain the same keyword, and then group index vectors of 20,000 documents into 80 categories, and divide the total keyword dictionary into 80 sub-dictionaries accordingly. Moreover, each sub-dictionary contains only 1000 keywords on average, which significantly improves the effective utilization of storage space, and the calculation efficiency is 80 times higher than the original, and the search efficiency will increase by more than 100 times. In our experiments, the dimensions of sub-dictionaries are almost different, which makes it easier to perform differentiated query, improves query efficiency and query precision. **(4) Low-overhead system maintenance based on balanced index forest.** Using fast index clustering, all index vectors are classified into multiple index partitions, and a corresponding balanced index tree is constructed for each index partition and then the *balanced index forest* is obtained. Since our index is distributed, in the dynamic maintenance of the system, we only need to maintain the corresponding index partition without touching all indexes, which greatly improves the efficiency of the index "add, delete, change and investigate" operations, reduces system maintenance overhead and enhances the strength of privacy protection (better than MKQE [22] and EDMRS [31]). Because of the index partition is relatively independent, even if the cloud server obtains part of the private information of any one index partition by evaluation, it cannot directly obtain the private information of other index partitions.

Our main contributions are summarized as follows:

1. *Searching adversarial networks* (SAN) is proposed to find the optimal balance of query precision and privacy protection strength.
2. *Maximum likelihood search balanced tree* (MLSB-Tree) is proposed to improve the search efficiency significantly.
3. *Fast index clustering method* is employed to reduce the dimension of data processing greatly.
4. *Balanced index forest* (BIF) is proposed to significantly improve the flexibility of system dynamic maintenance.

Table 1. Comparison of related works.

Item	MRSE[5]	MKQE[22]	MTS[27]	EDMRS[31]	MKRS_MO[14]	Ours
privacy-preserving query	√	√	√	√	×	√
high-precision query	√	√	√	√	√	√
differentiated query	×	×	×	×	×	√
efficient search	×	×	√	√	√	√
dynamic search	√	√	√	√	√	√
high-quality ranked search	×	√	√	√	√	√
flexible system maintenance	×	×	×	×	×	√
authorized access control	×	√	×	×	×	√

The remainder of this paper is organized as follows: Section 2 describes the problem formulation. Section 3 describes the details of our solution. In Section 4, we conduct a comprehensive analysis of the performance of MRSM_SAN. Section 5 discusses our solution and its implications.

2 Problem Formulation

2.1 Notations

- F_i : the plaintext document collection that belongs to DO_i (the data owners collection $DO = \{DO_1, \ldots, DO_m\}$), denoted as $F_i = \{F_{i,1}, \ldots, F_{i,n_i}\}$, which contains n_i documents.
- C_i : the ciphertext document corresponding to the plaintext document F_i that stored in the cloud server, denoted as $C_i = \{C_{i,1}, \ldots, C_{i,n_i}\}$.
- D : the dictionary consisting of p keyword, that contains all keywords extracted from documents outsourced by all owners, denoted as $D = \{d_1, \ldots, d_p\}$, it is a public ordered list shared by all participants in our scheme.
- $I_{i,j}$: the searchable binary index vector of document $F_{i,j}$, if the document $F_{i,j}$ has keyword d_t, $I_{i,j}[t] = 1$, otherwise $I_{i,j}[t] = 0$.
- $\widetilde{I}_{i,j}$: the searchable weighted index vector of document $F_{i,j}$ after index clustering and normalized processing.
- D_i : the sub-dictionary consisting of l_i keyword, contains all keywords for the i-th index partition which extracted from the keyword dictionary D, denoted as $D_i = \{d_{i,1}, \ldots, d_{i,l_i}\}$, where $i \in \{1, 2, \ldots, s\}$, s is the number of segmentation of the keyword dictionary D.
- τ_i : the unencrypted form of weighted balanced index tree for all documents in the i-th index partition.
- $\mathcal{F}$: the balanced index forest, that denoted as $\mathcal{F} = \{\tau_1, \ldots, \tau_s\}$.
- Q : the weighted query vector collection generated based on query request, denoted as $Q = \{Q_1, \ldots, Q_s\}$.
- T : the trapdoor for the query request, denoted as $T = \{T_1, \ldots, T_s\}$.

2.2 System Model

The system model proposed in this paper consists of four parties, is depicted in Fig. 1. Data owners (DO) are responsible for encrypting the data (document and

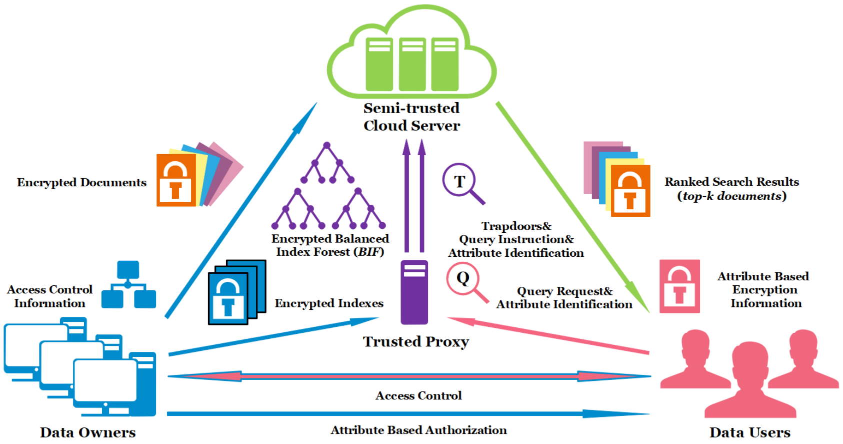

Fig. 1: The basic architecture of MRSM_SAN

index) and sending them to cloud server or trusted proxy; Data users (DU) are consumers of cloud services. Once the license is granted, they can retrieve the encrypted cloud data; Trusted proxy (TP) is responsible for index processing, query and trapdoor generation, user authority authentication; Cloud server (CS) provides cloud service, including running authorized access controls, performing searches for encrypted cloud data based on query requests, and returning *top-k* documents to data users. In addition, the system protocol is detailed in Section 4.

2.3 Threat Model

In general, CS is considered "honest but curious" in a searchable encryption system [29,34]. Specifically, CS follows and implements specified processes, algorithms, and protocols in an "honest" manner, but infers and analyzes the flow of information received during the protocol in a "curious" way (such as collecting query keywords and information about the index). According to the acquired information, CS could evaluate the correspondence between the keyword and the document, deduce/identify the private information of the encrypted cloud data and carry out an attack. Obviously, the more private information CS knows, the data security face the greater threat. According to the information CS knows, we consider two threat models with different attack capabilities as follows [5]: **(1)*****Known Ciphertext Model.*** CS only knows encrypted data (outsourced from DO) and searchable index (from TP). **(2)*****Known Background Model.*** CS also knows information other than encrypted data and searchable index, such as trapdoor statistics, access patterns, and keyword frequencies.

2.4 Design Goals

Data Privacy: the system should support multi-owner model that not only protects the security of outsourced data from DO, but also allows authorized

DU to legitimately access and easily search for the outsourced data they need.
Index Privacy: CS can not evaluate the correspondence between encrypted documents and keywords through encrypted indexes, nor can it evaluate the keyword weight information in the index.
Query Privacy: CS can not collect valid statistics during the query, nor could it deduce/identify the query keyword information through the trapdoor.
Trapdoor Unlinkability: the same query should be able to generate different trapdoors randomly, while CS can not distinguish between any two different trapdoors generated by the same query.
Rank Privacy: it is privacy-preserving, under the guarantee of query precision P_k, if the difference between the rank of *top-k* documents returned by CS (filling pseudo-keyword) and the real rank of these documents (without filling pseudo-keyword) is greater, the rank privacy protection P'_k is stronger (where $P_k = k'/k$, $P'_k = \sum |r_i - r'_i|/k^2$, k' and r_i are respectively the number of real *top-k* documents and the rank number of document in the retrieved k documents, and r'_i is document's real rank number in the whole ranked results) [5].
Efficient Search: by constructing a special tree-based index, the query complexity can be close to $O(\log N)$ [18].
Ranked Search: it supports multi-keyword ranked search, retrieves high-quality matching documents related to the query, and returns *top-k* documents to DU.
Dynamic Search: it supports dynamic operations, easily adding, deleting, changeing and investigating indexes, expanding keyword dictionaries and keys.

3 Secure and Efficient MRSM_SAN

In MRSM_SAN, we use *secure inner product* [13] to quantify the similarity between the query vector and the index vector, and obtain *top-k* documents based on the calculated score. Different from MRSE [5], MKQE [22], EDMRS [31] and MKRS_MO [14], the elements of index vector and query vector are not binary number, but floating-point number between 0 and 1 (as weights for keywords). We create the MLSB-Tree as index, whose leaf nodes ordered follow maximum probability, which significantly improves the search efficiency. We use the pseudo-keyword filling to achieve *privacy-preserving scheme* in *known background model* [5,4] that has higher privacy protection requirements. SAN is used to optimize the pseudo-keyword filling, which significantly improves the query precision when adding random noise. In MRSM_SAN, DO grasp the authorization for data access control, but DO only need to outsource the encrypted documents to CS and send the initial indexes to TP, subsequent processing is done by TP, so we mainly introduce the work content based on TP in the following sections.

3.1 MRSM_SAN Framework

***Setup*:** based on the results of index clustering (get s index partitions, i-th index partition corresponds to a keyword sub-dictionary D_i), TP determines the

sub-dictionarys D_i size l_i, the number of pseudo-keyword U_i, sets the parameter $V_i = U_i + l_i$, $V = \{V_1,\ldots,V_s\}$, $U = \{U_1,\ldots,U_s\}$, $l = \{l_1,\ldots,l_s\}$.
***KeyGen* (V):** TP generates secret key $SK = \{SK_1,\ldots,SK_s\}$, where $SK_i = \{S_i, M_{i,1}, M_{i,2}\}$, $M_{i,1}$ and $M_{i,2}$ are two invertible matrices that with the dimension $V_i \times V_i$, and S_i is a random V_i-length vector.
***Extended-KeyGen* (SK_i, z_i):** for dynamic search, if z_i new keywords are added into the i-th sub-dictionary, the TP generates a new $SK'_i = \{S'_i, M'_{i,1}, M'_{i,2}\}$, two invertible matrices $M'_{i,1}$ and $M'_{i,2}$ with the dimension $(V_i + z_i) \times (V_i + z_i)$, and a new $(V_i + z_i)$-length vector S'_i.
***BuildIndex* (I, SK):** for the weighted index vectors with the dimension l_i that in the i-th index partition, TP fills them with U_i pseudo-keywords according to the optimal probability distribution function, and obtains secure index vectors with high privacy protection strength. Then TP uses secure index vectors to build the MLSB-Tree τ_i and encrypts τ_i to $\widetilde{\tau_i}$ using SK_i. After generating index tree τ_i for all index partitions, TP obtains the BIF $\mathcal{F} = \{\tau_1,\ldots,\tau_s\}$. TP sends the encrypted balanced index forest $\widetilde{\mathcal{F}} = \{\widetilde{\tau_1},\ldots,\widetilde{\tau_s}\}$ to CS.
***Trapdoor* (Q, SK):** DU sends query request (keywords and their weights) to TP. TP generates query $Q = \{Q_1,\ldots,Q_s\}$ (where Q_i is a weighted vector with dimension V_i) and calculates the trapdoor $T = \{T_1,\ldots,T_s\}$ using an SK and sends T to the CS.
***Query* (T, k, I):** TP sends the query information to CS and specifies the index partition to be queried. CS performs searches based on the query, and returns *top-k* documents to the DU.

3.2 MRSM_SAN Details

Binary Index Vector Generation. Based on *vector space model*(VSM [25]), DO_i builds the index $I_i = \{I_{i,1},\ldots,I_{i,n_i}\}$ of the binary form for the documents $F_i = \{F_{i,1},\ldots,F_{i,n_i}\}$, then sends binary index vectors to TP.

Index Clustering. As illustrated in Fig. 2, using the algorithms for clustering data [15], for all index vectors from $DO = \{DO_1,\ldots,DO_m\}$, TP firstly performs clustering (local clustering) on each owner's index vectors to form a plurality of initial partitions, then performs clustering (global clustering) on all owners' index vectors by separation and recombination, form final index partitions. According to the obtained s index partition, the keyword dictionary is divided into s sub-dictionaries, and the keywords contained in the sub-dictionary are the same as the keywords contained in all the documents included in the corresponding index partition. Therefore, after the index clustering, the index vector has a smaller dimension and a large number of high similarity index vectors are gathered in the same index partition, which facilitates the normalization of the weights. This not only solves "dimension disaster" caused by data sparsity, but also solves the problem of document quality differences between different data owners.

Weighted Index Generation. **(I)** *Correlativity Matrix Generation*: In order to calculate keyword weights more scientifically and reasonably, it is necessary

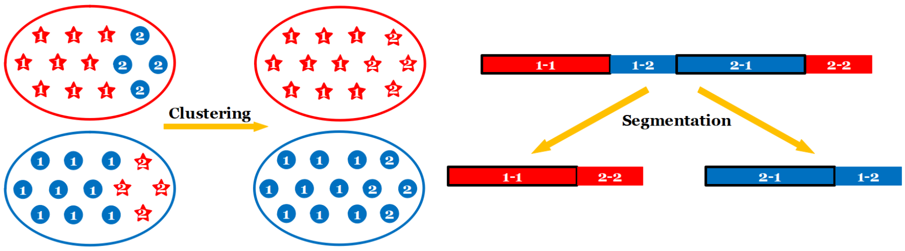

Fig. 2: Index clustering and keyword dictionary segmentation

to consider the semantic relationship between keywords, that is, to assess the degree of influence between different keywords. We use the corpus to determine the semantic relationship between different keywords (keyword relevance). Then we obtain the *correlativity matrix* $S_{l_i \times l_i}$ (symmetric matrix). **(II)** *Weight Generation*: We use the KDO weight model [14] to generate the raw weight. Specifically, TP can construct the average keyword popularity (denoted as AKP) about different DO. AKP_i (the average keyword popularity of DO_i) can be computed as: $AKP_i = (P_i \cdot \widehat{I_i}) \otimes \alpha_i$ (where $\widehat{I_i}$ is the index after index clustering, the operator $\otimes$ denotes the product of two vectors corresponding elements, $\alpha_i = (\alpha_{i,1},\ldots,\alpha_{i,l_i})$, if $|L_i(d_t)| \neq 0$ (the number of documents containing keyword d_t), $\alpha_{i,t} = \frac{1}{|L_i(d_t)|}$,otherwise $\alpha_{i,t} = 0$) Calculate the raw weight information for DO_i, $W_i^{raw} = S_{l_i \times l_i} \cdot AKP_i$, where $W_i^{raw} = (W_{i,1}^{raw},\ldots,W_{i,l_i}^{raw})$. **(III)** *Normalized Processing*: TP gets the maximum raw weight of every keyword among different DO, $W_{max} = (W_{i,1}^{raw},W_{i',2}^{raw},\ldots)$. Based on the W_{max}, calculated $W_{i,t}$ $= \frac{W_{i,t}^{raw}}{W_{max}[j]}$. **(VI)** *The weighted index generation*: $\widetilde{I}_{i,j} = \widehat{I}_{i,j} \otimes W_i$, where $\widetilde{I}_{i,j}$ denoted as weighted index vector of document $F_{i,j}(j \in \{1, 2, \ldots, n_i\})$.

Balanced Index Tree and Balanced Index Forest Generation. **(I)** *Balanced Index Tree Generation*: TP performs 10000 random searches, gets the sum of the matching scores of each index vector and all random query vectors, and then sorts the index vectors according to the score from high to low. TP follows the bottom-up strategy and builds the balanced index tree based on the "greedy" method, and then obtain the MLSB-Tree. This makes the complexity of the query vector and index vector matching search process close to $O(\log N)$. **(II)** *Balanced Index Forest Generation*: TP builds all balanced index tree τ_i in the same way for all index partitions, then obtains the balanced index forest $\mathcal{F}$ $= \{\tau_1,\ldots,\tau_s\}$.

Encrypted Index Tree and Index Forest Generation. **(I)** *Encrypted Index Tree Generation*: TP encrypts weighted index tree τ_i with the secret key SK_i ($SK_i = \{S_i, M_{i,1}, M_{i,2}\}$) to obtain an encrypted index tree $\widetilde{\tau}$. The encryption process is as follows: (a) For each node of τ_i that denoted u_i, TP "splits" the vector $u_i.v$ into two random vectors $u_i.v_1, u_i.v_2$ Specifically, if $S_i[t] = 0$, $u_i.v_1[t]$ $= u_i.v_2[t] = u_i.v[t]$; else if $S_i[t] = 1$, $u_i.v_1[t]$ is a random value, set $u_i.v_2[t] =$

$u_i.v[t] - u_i.v_1[t]$. (b) TP encrypts $u_i.v$ with reversible matrices $M_{i,1}$ and $M_{i,2}$ to obtain two V_i-length vectors $\widetilde{u_i.v} = \{M_{i,1}^T u_i.v_1, M_{i,2}^T u_i.v_2\}$. (c) After encrypting the vectors in all tree nodes, TP sends the encrypted index tree τ_i to the CS, the operation of encrypted index tree generation is completed. Because of the index tree is described by a set of nodes and a set of pointers indicating all parent-child relationships, TP only encrypts the vector $u_i.v$ contained in each node u_i and all pointers are unchanged, the structure of the tree is unchanged. Therefore, the unencrypted index tree τ_i and the encrypted index tree $\widetilde{\tau_i}$ are isomorphic($\tau_i \cong \widetilde{\tau i}$). **(II)** *Encrypted Index Forest Generation*: TP encrypts all balanced index tree in the same way for all index partitions, then obtains the encrypted index forest $\widetilde{\mathcal{F}} = \{\widetilde{\tau}_1,\ldots,\widetilde{\tau}_s\}$.

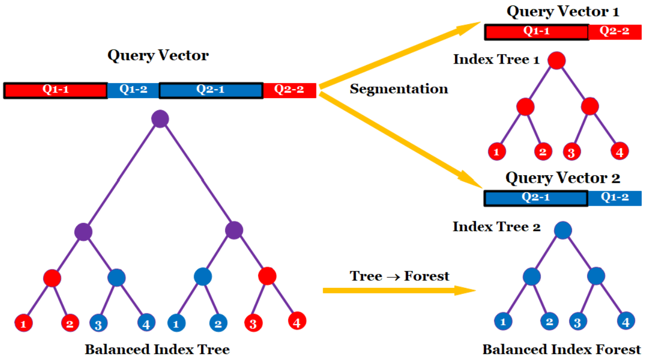

Fig. 3: Balanced Index Tree and Balanced Index Forest

Differentiated Query Based on Balanced Index Forest. As illustrated in Fig. 3, DU sends query request (query keywords and their weights) to TP, and TP allocates keywords to different keyword sub-dictionaries based on the keyword dictionary segmentation and balanced index forest features, so as to form multiple different query vectors, while determining the index tree that matches the query. Then, the query vectors are encrypted with different keys to form a plurality of different trapdoors, trapdoors are sent to the cloud server and index trees of the query is specified. After the cloud server verifies the authorization information, it performs a search only on the specified matching index tree, and finally returns the *top-k* documents of each partition to the data consumer. (The number of *top-k* documents for each valid query partition follows the drawer principle).

Trapdoor Generation. When DU wants to search the interested documents in whole encrypted document collection, he/she only need to send query request to TP. TP generates $Q = \{Q_1,\ldots,Q_s\}$ based on query request, then encrypts Q to get $T = \{T_1,\ldots,T_s\}$. Specifically, Q_i is a $(l_i + U_i)$-length weighted vector, then

Q_i could be encrypted by SK_i like index encryption. The only difference is the “split” process. Specifically, if $S_i[t] = 0$, $Q_{i,1}[t]$ is a random value, and $Q_{i,2}[t] = Q_i[t] - Q_{i,1}[t]$; else if $S_i[t] = 1$, $Q_{i,1}[t] = Q_{i,2}[t] = Q_i[t]$, where $t \in \{1, 2, \dots, l_i\}$. Finally, TP encrypts Q_i as trapdoor $T_i = \{M_{i,1}^{-1}Q_{i,1}, M_{i,2}^{-1}Q_{i,2}\}$ and sends T_i to the cloud server

Search Process of MRSM_SAN. **(I)** *Preparation before Query* : DU sends query keywords and part of the attributes to TP. After verifying the validity of the query from DU, TP generates trapdoors and submits them with attribute information to CS. Using this information, CS first determines whether TP can access the data: if access control passes, CS uses the index tree to search for the encrypted index vector that matches the query vector, and calculates the matching score for the authorized document index, CS returns *top-k* documents to DU based on the matching score. Otherwise CS will not perform a search.
(II) *Calculate Matching Score for Query on the i-th Index Tree* τ_i:

$$Score(\widetilde{u_i.v}, T_i) = \{M_{i,1}^T u_i.v_1, M_{i,2}^T u_i.v_2\} \cdot \{M_{i,1}^{-1}Q_{i,1}, M_{i,2}^{-1}Q_{i,2}\} = u_i.v \cdot Q_i \quad (1)$$

(III) *Search Process for MLSB-Tree*: As illustrated in Fig. 4, greedy depth-first search (GDFS) algorithm is executed to perform the query. When $k = 2$, it only need to perform 4 times calculation of the matching score between the query vector and the vector on the tree node, then return *top-k* documents ($F_{1,1}$, $F_{1,2}$). When $k = 3$, it needs to perform 4 times calculation to return *top-k* documents ($F_{1,1}$, $F_{1,2}$, $F_{1,3}$). The number of calculation does not exceed the number of index vectors.

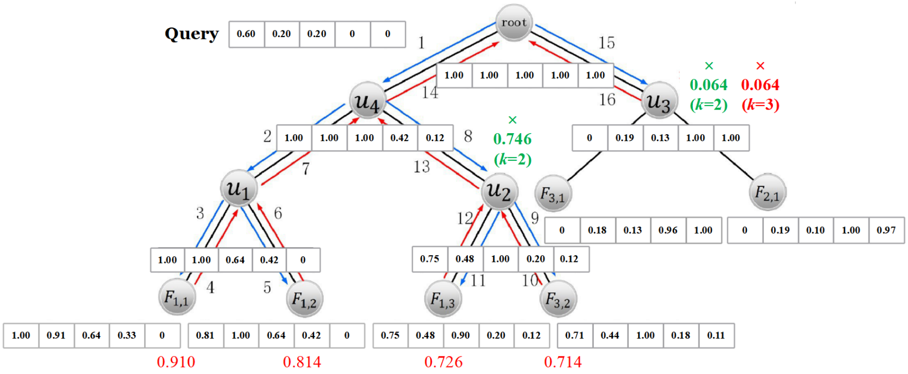

Fig. 4: The search process for MLSB-Tree

3.3 Searching Adversarial Networks (SAN)

As described in [5], when random pseudo-keyword is introduced (in their scheme, the filling of random pseudo-keyword follows the *Gaussian distribution*), the strength of privacy protection increases, but the accuracy of the query is impaired. Therefore, it is necessary to optimize the probability distribution function that controls the pseudo-keyword filling. Although the *Gaussian distribution* has good symmetry, different data sets have different feature distributions. Therefore, we need to customize the pseudo-keyword probability distribution suitable for the data set. Inspired by the *generative adversarial networks* (GAN) [11], we propose *searching adversarial networks* (SAN), as illustrated in Fig. 5. ***Searcher Network S(ε)*** **:** The search result is generated by taking the random noise ε (the object probability distribution $p(\varepsilon)$) as an input and performing a search, and supplies the search result to the discriminator network $D(x)$. ***Discriminator Network D(x)*:** The input has an accurate actual result or search result and attempts to predict whether the current input is an actual result or a search result. One of the inputs x is obtained from the real search result distribution $p(x)$, and then one or two are solved. Classify the problem and generate scalars ranging from 0 to 1. Finally, in order to reach a balance point which is the best point of the minimax game(as formula 2). The searcher network $S(\varepsilon)$ generates search results, and the discriminator network $D(x)$ considers the probability that the searcher network $S(\varepsilon)$ produces the accurate real results is 0.5.

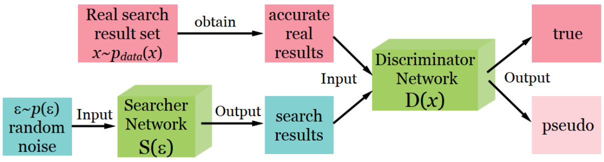

Fig. 5: Searching Adversarial Networks

To learn the searcher's distribution p_s over data x, we define a prior on input noise variables $p_\varepsilon(\varepsilon)$, then represent a mapping to data space as $S(\varepsilon;\theta_s)$, where S is a differentiable function represented by a multi-layer perception with parameters θ_g. We also define a second multi-layer perception $D(x;\theta_d)$ that outputs a single scalar. $D(x)$ represents the probability that x came from the data rather than p_s. We train D to maximize the probability of assigning the correct label to both training examples and samples from S. We simultaneously train S to minimize $\log(1-D(S(\varepsilon)))$: In other words, D and S play the following two-player min-max game with value function $V(S,D)$:

$$\min_S \max_D V(D,S) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{x\sim p_\varepsilon(\varepsilon)}[\log(1-D(S(\varepsilon)))] \tag{2}$$

As illustrated in the Fig. 6, similar to GAN [11], SAN is trained by simultaneously updating the discriminative distribution (D, blue, dashed line) so that it discriminates between samples from the real search result set (black, dotted line) p_x from those of the searching distribution $p_s(S)$ (green, solid line). The lower horizontal line is the domain from which ε is sampled, in this case uniformly.

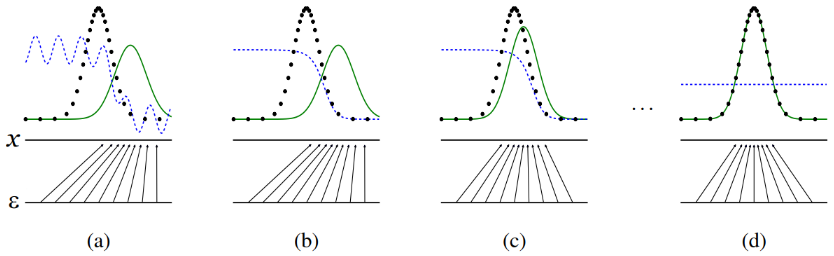

Fig. 6: The training process of SAN obtains the optimal probability distribution function for controlling the pseudo-keyword filling (it is similar to the principle of GAN [11]).

The horizontal line above is part of the domain of x. The upward arrows show how the mapping $x = S(\varepsilon)$ imposes the non-uniform distribution p_s on transformed samples. S contracts in regions of high density and expands in regions of low density of p_s. **(a)** Consider an adversarial pair near convergence: p_s is similar to p_{data} and D is a partially accurate classifier. **(b)** In the inner loop of the algorithm, D is trained to discriminate samples from data, converging to $D^{*}(x) = \frac{p_{data}(x)}{p_{data}(x)+p_s(x)}$. **(c)** After an update to S, gradient of D has guided $S(\varepsilon)$ to flow to regions that are more likely to be classified as data. **(d)** After several steps of training, if S and D have enough capacity, they will reach a point at which both can not improve because $p_s = p_{data}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(x) = \frac{1}{2}$.

3.4 Security Analysis

Data Privacy: outsourced data is encrypted using *symmetric encryption techniques*, such as *advanced encryption standard* (AES), therefore the privacy of the data is protected by an encryption key. In our solution, different DO have their own document encryption keys. For any DO_i, even if the attacker cooperates with the other DO, the attacker can not decrypt any encrypted documents belonging to DO_i without the correct key.

***Index Privacy*:** in MRSM_SAN, the original index is generated by DO, and the weighted index vectors and index trees are generated by TP. So, on the one hand, TP does not disclose any information about the plaintext index and the encrypted index key SK; on the other hand, in order to get the ciphertext index, the ASPE method [30] is widely used in many secure keyword search schemes to protect index privacy and its security has been proven . In addition, according to the security analysis proposed by Wong et al. [30], the intensity of the 1024-bit RSA key is roughly equivalent to the 80-bit symmetric key, and with key length increasing, the system will achieve greater security. According to our actual data experiments, the size of the dictionary is between 1000 and 80,000, far exceeding 80 (equal to the length of the key), and the security far exceeds the security of RSA for 2048-bit key encryption. Therefore, index privacy is well protected.

***Trapdoor Privacy*:** we use random pseudo-keywords to fill the query vector, which improves the randomness of generation for trapdoor, making it impossible for CS to distinguish the two trapdoors generated by any one query, thus ensuring the unlinkability of the trapdoor T.
***Query Privacy*:** it is the same as the index privacy. On the one hand, the plaintext form of the query vector is only known by TP, and TP does not reveal any information about it; on the other hand, the ASPE method is also used to encrypt the query vector to generate trapdoors. Therefore, the privacy of the query and the privacy of the index are protected with the same extent.
***Key management*:** in MRSM_SAN, document key management is implemented with *attribute-based encryption* (ABE) technology [12,24]. DO uses the access control information to encrypt the document key and then stores the encryption key in the CS. Access control information is associated with DU's attributes. Under the composite three-party Diffie-Hellman assumption and the bilinear Diffie-Hellman assumption, the ABE scheme is selectively secure. The CS can use the partial private key as the attribute information to determine whether the user can access the document key, but can not decrypt the ciphertext to obtain the document key. In this way, the security of the key management scheme is guaranteed.

4 Experiment and Performance Evaluation

Preparation for Real-world Data Experiment. MRSM_SAN solution is implemented in the Windows 10 operating system using the *Python* language and tested its accuracy and efficiency on real-word data sets. We used the academic papers provided by IEEE xplore[3] to collect the original data set and structure the data: $Item_{i,j} = (DO_i, ID, popularity, keyword)$, which corresponds to $F_{i,j}$. Specifically, to implement the *multi-owner model*, we randomly selected 400 academic conferences (represented as DO) involving multiple domains. Different academic conferences have different themes, and each paper has it's ID as a unique identifier. The popularity of documents includes authority and enthusiasm. The authority is represented by the number of times the paper is cited, and the enthusiasm is represented by the number of times the paper is viewed. The keywords of the document include the IEEE keyword, the IEEE control index, and the author keyword. Our experiments consist of: **(I)** pre-processing collected raw data; **(II)** index construction, index tree construction, query vector generation, trapdoor generation and random query; **(III)** comparing query accuracy and search efficiency with MRSE [5], EDMRS [31] and MKRS_MO [14] schemes. The experiment was performed on a PC with the Intel(R)Core(TM)i5-6200U processor running at 2.40 GHz, 4.00GB RAM (it is notable that our computing resources are only comparable to ordinary personal computer, but our solution can still be implemented quickly on this platform). All results represent the average of 1000 trials.

[3] IEEE xplore, https://ieeexplore.ieee.org/. Last accessed 3 May ,2019.

Search Efficiency of MLSB-Tree. This part of the experiments intend to reveal the superiority of the MLSB-Tree. Search efficiency is mainly described by query speed, and our experimental objects are index trees that are structured with different strategy: EDMRS [31](single owner model, with ordinary balanced binary tree), MKRS_MO [14](multi-owner model, with grouped balanced binary tree), MRSM_SAN(our solution without MLSB-Tree), MRSM_SAN_MLSB-Tree(our solution with MLSB-Tree). We first randomly generate 1000 query vectors, and then perform search operations in each index tree respectively, and finally take the results of 20 repeated experiments for analysis. As shown in Table 2 and Fig. 7a, the query speed and query stability based on MLSB-Tree are better than other index trees. Compared with EDMRS and MKRS_MO, the query speed increased by 21.72% and 17.69% respectively. In terms of stability, the MLSB-Tree is also significantly better than other index trees.

Table 2. Comparison of related works (search efficiency).

Item	EDMRS[31]	MKRS_MO[14]	MRSM_SAN	**MLSB-Tree**
highest value/s	4.4720	4.5099	4.0841	**3.2922**
lowest value/s	3.8318	3.6213	3.5395	**2.9950**
average value/s	3.9592	3.7655	3.6476	**3.0994**
variance/s	0.0193	0.0515	0.0159	**0.0061**

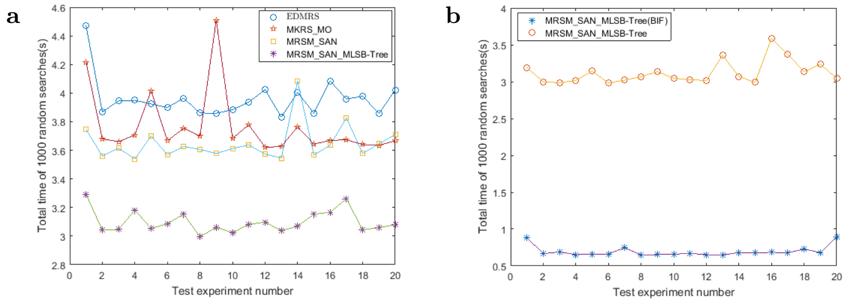

Fig. 7: Time cost of query for 1000 random searches in 500 sizes of data set

Search Efficiency of BIF. This part of the experiments intend to reveal the superiority of the BIF. As shown in Fig. 7b,the search speed of MRSM_SAN (with MLSB-Tree and BIF) is significantly higher than MRSM_SAN (only with MLSB-Tree), and the search efficiency is improved by 5 times and the stability increase too. This is just the experimental result of 500 documents set with the 4000-dimension keyword dictionary. After the index clustering operation, the keyword dictionary is divided into four sub-dictionaries with a dimension of approximately 1000. As the amount of data increases, the dimension of the keyword dictionary will become extremely large, and the advantages of BIF will become more apparent. In our analytical experiments, the theoretical efficiency ratio before and after segmentation is: $\eta = s\frac{O(\log N)}{O(\log N)-O(\log s)}$,where s is the number of index partitions after fast index clustering, and N is the number of documents included. When the amount of data increases to 20,000, the total keyword dictionary dimension is as high as 80,000. If the keyword sub-dictionary dimension is 1000, the number of index partitions after fast index clustering is 80, the search efficiency will increase by more than 100 times ($\eta = 143$). This will bring huge benefits to large information systems, and our solutions can exchange huge returns with minimal computing resources.

Optimal Pseudo-keyword Filling. The purpose of this part of the experiments is to find the optimal solution for pseudo-keyword filling. After SAN finds the optimal probability distribution (close to the *Gaussian distribution* because the data set is large enough), we adjust the parameters of the control probability distribution function to find the optimal game equilibrium for *query precision* (denoted as x) and *rank privacy protection* (denoted as y). The definition of query precision and ranking privacy protection has been given in the system design goals, see Section 2.4. We choose 95% query precision and 80% rank privacy protection as benchmarks to get the *game equilibrium score* calculation formula: $f(x,y) = \frac{1}{95}x^2 + \frac{1}{80}y^2$ (objective function to be optimized). As illustrated in Fig. 8, we find the optimal game equilibrium ($\max f(x,y) = 174$) at $\sigma_1 = 0.05$, $\sigma_2 = 0.08$, $\sigma_3 = 0.12$. And the corresponding query precision are: 98%, 97%, 93%. The corresponding rank privacy protection are: 78%,79%,84%. Based on the results, we can choose the best value of σ to achieve optimal pseudo-keyword filling so that it can satisfy our query protection requirement and maximize rank privacy protection.

Comparison of Search Efficiency (Larger Data Set). The efficiency of MRSM_SAN (without BIF) and related works [5,14,22,31] are show as Fig. 9a, and the efficiency of MRSM_SAN(without BIF) and MRSM_SAN(with BIF) are show as Fig. 9b. In Fig. 9a, experiments on the real-world data set show that our solution achieves near binary search efficiency and is superior to other existing comparison schemes. As the amount of data increases, our solution has a greater advantage. However, it should be noted that this is only based on the performance of the MLSB-Tree, and does not employ the BIF. In Fig. 9b, it shows the charm of BIF. By comparing the experimental results of MRSM_SAN with BIF with MRSM_SAN without BIF, we conclude that when the data volume grows exponentially the data features become more sparse. If all index vectors rely on only an index tree to complete the search task, the computational complexity will be getting farther away from $O(\log N)$. Due to sparse data features, the similarity between index vectors is mostly close to zero or even equal to zero, which brings a lot of trouble to the pairing of index vectors, and the construction of balanced index tree is not global order, so it is necessary to traverse many nodes in the search, which proves the limitation of the *grouped balanced binary*

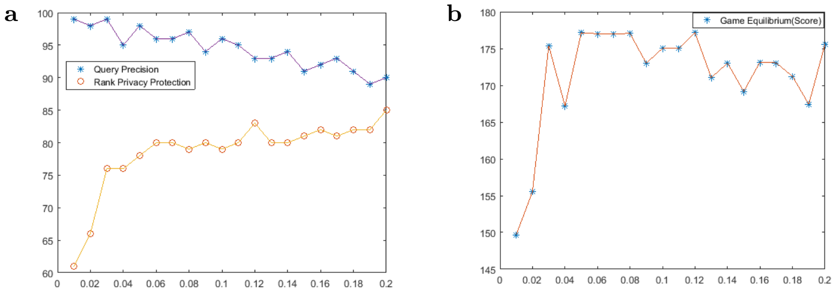

Fig. 8: With different choice of standard deviation σ for the random variable ε, (a) query precision(%) and rank privacy protection(%); (b) game equilibrium (score). explanation for $\sigma \in [0.01, 0.2]$: When σ is greater than 0.2, the weight of the pseudo-keyword may be greater than 1, which violates our weight setting (between 0 and 1), so we only need to find the best game equilibrium point when $\sigma \in [0.01, 0.2]$.

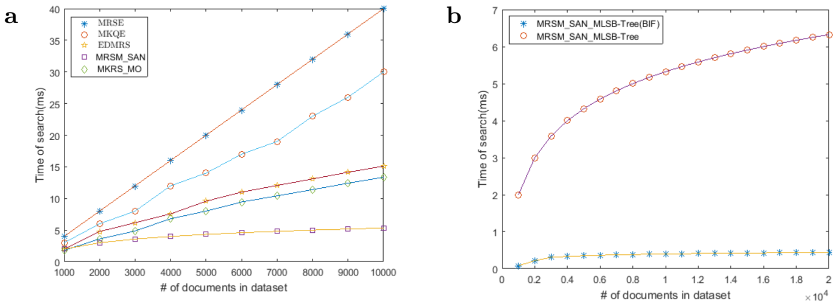

Fig. 9: Time cost of query for the same query keywords (10 keywords) in different sizes of data set

tree in MKRS_MO [14]. We use the maximum likelihood method to construct MLSB-Tree. We use random searches to build a tree and in the probabilistic and statistical sense, the closer the number of random searches is to infinity, the higher the search efficiency of the obtained index tree. The computational complexity of search can converge to $O(\log N)$, which is the excellence of our scheme. And it is more notable that the maintenance cost of scheme based on the BIF is much lower than the cost of scheme only based on a balanced index tree. When the data owner has added a new document to the cloud server, and TP needs to insert a new index node in the index tree of the cloud server accordingly. If it is only based on the index tree, it must search for at least $O(\log N)$ times search and at least $O(\log N)$ times data updates so that the total cost is $2O(\log N)$ (where N is the number of index vectors that contained by the index tree). But the BIF is very different, because we group all index vectors into s different partitions. We assume that the number of index vectors in each partition is equal so we need to spend the same update operation for each partition, which makes the overhead is only $2(O(\log N)$ - $O(\log s))$. In addition, the larger the amount of data and the more sparse the data, the more partitions and the more significant the efficiency improvement is. In summary, the BIF is derived from the balanced index tree, but more excellent than the balanced index tree.

5 Discussion

In this paper, we propose secure and efficient multi-keyword ranked search over encrypted cloud data for multi-owner model based on *searching adversarial networks* (MRSM_SAN), introduce our core techniques and conduct in-depth performance analysis. Creatively using *game equilibrium theory* to find the best balance between query precision and privacy protection strength, and combining traditional *searchable encryption* with *optimal control theory*, which opens a door to the research of *intelligent methods* in searchable encryption. To classify index vectors from different data owners into multiple index partitions and correspondingly divide the keyword dictionary into multiple sub-dictionaries, on the one hand, the problem of document quality differences between different data owners in multi-owner model can be better solved; on the other hand, the dimension of the index vector is significantly reduced to avoid "dimension disaster" caused by big data sparsity, which significantly improves the efficiency of secure inner product calculation based on secure *kNN* scheme [30]. In addition, we propose *maximum likelihood search balanced tree*, which generated by a sufficient amount of random searches, brings the query complexity closer to $O(\log N)$. It means that in an *uncertain system* (owner's data is uncertain, user's query is uncertain), using the probability learning method to optimize the query is effective, and it is also verified in our experimental results. Last but not least, we implement differentiated query based on *balanced index forest* and make full use of the distributed architecture to simplify system construction, which not only reduces the overhead of system dynamic maintenance, but also improves the search efficiency and achieves fine-grained search. This is beneficial to improve the availability, flexibility and efficiency of complex and large information systems.

Acknowledgment

This work was supported by "the Fundamental Research Funds for the Central Universities" (No. 30918012204) and "the National Undergraduate Training Program for Innovation and Entrepreneurship" (Item number: 201810288061). NJUST graduate Scientific Research Training of 'Hundred, Thousand and Ten Thousand' Project "*Research on Intelligent Searchable Encryption Technology*".

References

1. Ballard, L., Kamara, S., Monrose, F.: Achieving efficient conjunctive keyword searches over encrypted data. In: ICICS 2005. pp. 414–426. Springer (2005)

2. Boneh, D., Crescenzo, G.D., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: EUROCRYPT 2004. pp. 506–522. Springer (2004)
3. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: TCC 2007. pp. 535–554. Springer (2007)
4. Cao, N., Wang, C., Li, M., Ren, K., Lou, W.: Privacy-preserving multi-keyword ranked search over encrypted cloud data. In: IEEE INFOCOM 2011. pp. 829–837. IEEE (2011)
5. Cao, N., Wang, C., Li, M., Ren, K., Lou, W.: Privacy-preserving multi-keyword ranked search over encrypted cloud data. IEEE TPDS **25**(1), 222–233 (2014)
6. Chang, Y., Mitzenmacher, M.: Privacy preserving keyword searches on remote encrypted data. In: ACNS 2005. pp. 442–455. Springer (2005)
7. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: ACM CCS 2006. pp. 79–88. ACM (2006)
8. Fu, Z., Ren, K., Shu, J., Sun, X., Huang, F.: Enabling personalized search over encrypted outsourced data with efficiency improvement. IEEE TPDS **27**(9), 2546–2559 (2016)
9. Fu, Z., Wu, X., Guan, C., Sun, X., Ren, K.: Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement. IEEE TIFS **11**(12), 2706–2716 (2016)
10. Goh, E.: Secure indexes. IACR Cryptology ePrint Archive **2003**, 216 (2003)
11. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A.C., Bengio, Y.: Generative adversarial networks. CoRR **abs/1406.2661** (2014)
12. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: ACM CCS 2006. pp. 89–98. ACM (2006)
13. Gu, C., Gu, J.: Known-plaintext attack on secure knn computation on encrypted databases. Security and Communication Networks **7**(12), 2432–2441 (2014)
14. Guo, Z., Zhang, H., Sun, C., Wen, Q., Li, W.: Secure multi-keyword ranked search over encrypted cloud data for multiple data owners. Journal of Systems and Software **137**(3), 380–395 (2018)
15. Jain, A.K., Dubes, R.C.: Algorithms for Clustering Data. Prentice-Hall (1988)
16. Jiang, X., Yu, J., Yan, J., Hao, R.: Enabling efficient and verifiable multi-keyword ranked search over encrypted cloud data. Inf. Sci. **403**, 22–41 (2017)
17. Kamara, S., Lauter, K.E.: Cryptographic cloud storage. In: WLC 2010. pp. 136–149. Springer (2010)
18. Knuth, D.E.: The art of computer programming, Volume III, 2nd Edition. Addison-Wesley (1998)
19. Kumar, D.V.N.S., Thilagam, P.S.: Approaches and challenges of privacy preserving search over encrypted data. Inf. Syst. **81**, 63–81 (2019)
20. Li, H., Liu, D., Jia, K., Lin, X.: Achieving authorized and ranked multi-keyword search over encrypted cloud data. In: IEEE ICC 2015. pp. 7450–7455. IEEE (2015)
21. Li, J., Wang, Q., Wang, C., Cao, N., Ren, K., Lou, W.: Fuzzy keyword search over encrypted data in cloud computing. In: IEEE INFOCOM 2010. pp. 441–445. IEEE (2010)
22. Li, R., Xu, Z., Kang, W., Yow, K., Xu, C.: Efficient multi-keyword ranked query over encrypted data in cloud computing. Future Generation Comp. Syst. **30**, 179–190 (2014)

23. Miao, Y., Ma, J., Liu, X., Jiang, Q., Zhang, J., Shen, L., Liu, Z.: VCKSM: verifiable conjunctive keyword search over mobile e-health cloud in shared multi-owner settings. Pervasive and Mobile Computing **40**, 205–219 (2017)
24. Ostrovsky, R., Sahai, A., Waters, B.: Attribute-based encryption with non-monotonic access structures. In: ACM CCS 2007. pp. 195–203. ACM (2007)
25. Salton, G., Wong, A., Yang, C.: A vector space model for automatic indexing. Commun. ACM **18**(11), 613–620 (1975)
26. Song, D.X., Wagner, D.A., Perrig, A.: Practical techniques for searches on encrypted data. In: IEEE S&P 2000. pp. 44–55. IEEE Computer Society (2000)
27. Sun, W., Wang, B., Cao, N., Li, M., Lou, W., Hou, Y.T., Li, H.: Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. IEEE TPDS **25**(11), 3025–3035 (2014)
28. Wang, C., Cao, N., Li, J., Ren, K., Lou, W.: Secure ranked keyword search over encrypted cloud data. In: ICDCS 2010. pp. 253–262 (2010)
29. Wang, C., Wang, Q., Ren, K., Lou, W.: Privacy-preserving public auditing for data storage security in cloud computing. In: IEEE INFOCOM 2010. pp. 525–533. IEEE (2010)
30. Wong, W.K., Cheung, D.W., Kao, B., Mamoulis, N.: Secure knn computation on encrypted databases. In: ACM SIGMOD 2009. pp. 139–152. ACM (2009)
31. Xia, Z., Wang, X., Sun, X., Wang, Q.: A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. IEEE TPDS **27**(2), 340–352 (2016)
32. Xu, L., Sun, S., Yuan, X., Liu, J.K., Zuo, C., Xu, C.: Enabling authorized encrypted search for multi-authority medical databases. IEEE TETC **1**(3), 1–1 (2019)
33. Xu, L., Xu, C., Liu, J., Zuo, C., Zhang, P.: Building a dynamic searchable encrypted medical database for multi-client. Inf. Sci. **1**(5), 1–1 (2019)
34. Yu, S., Wang, C., Ren, K., Lou, W.: Achieving secure, scalable, and fine-grained data access control in cloud computing. In: IEEE INFOCOM 2010. pp. 534–542. IEEE (2010)
35. Zhang, W., Xiao, S., Lin, Y., Ting, Zhou, S.: Secure ranked multi-keyword search for multiple data owners in cloud computing. In: IEEE/IFIP DSN 2014. pp. 276–286. IEEE (2014)