Multi-owner 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. Searchable symmetric encryption (SSE) for multi-owner model draws much attention as it enables data users to perform searches over encrypted cloud data outsourced by data owners. However, implementing secure and precise query, efficient search and flexible dynamic system maintenance at the same time in SSE remains a challenge. To address this, this paper proposes secure and efficient multi-keyword ranked search over encrypted cloud data for multi-owner model based on searching adversarial networks. We exploit searching adversarial networks to achieve optimal pseudo-keyword padding, and obtain the optimal game equilibrium for query precision and privacy protection strength. Maximum likelihood search balanced tree is generated by probabilistic learning, which achieves efficient search and brings the computational complexity close to $\mathcal{O}(\log N)$. In addition, we enable flexible dynamic system maintenance with balanced index forest that makes full use of distributed computing. Compared with previous works, our solution maintains query precision above 95% while ensuring adequate privacy protection, and introduces low overhead on computation, communication and storage.

Keywords: Searchable Symmetric Encryption· Multi-owner · Ranked Search· Searching Adversarial Networks· Maximum Likelihood

1 Introduction

Background and Motivation. In cloud computing, searchable symmetric encryption (SSE) for multiple data owners model (multi-owner model, MOD) draws much attention as it enables multiple data users (clients) to perform searches over encrypted cloud data outsourced by multiple data owners (authorities). Unfortunately, none of the previously-known traditional SSE scheme for MOD achieve secure and precise query, efficient search and flexible dynamic system maintenance at the same time [9]. This severely limits the practical value of SSE and decreases its chance of deployment in real-world cloud storage systems.

Related Work and Challenge. SSE has been continuously developed since it was proposed by Song et al. [12], and *multi-keyword ranked search* over encrypted cloud data scheme is recognized as outstanding [9]. Cao et al. [1] first

proposed privacy-preserving multi-keyword ranked search scheme (MRSE), and established strict privacy requirements. They first employed *asymmetric scalar-product preserving encryption* (ASPE) approach [15] 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 strength. For better query precision and query speed, Sun et al. [13] proposed MTS with the TF×IDF keyword weight model, where 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. This means that TF×IDF cannot handle the differences between data from different owners in MOD, since each owner's data is different and there is no uniform standard to measure keyword weights. Based on MRSE, Li et al. [8] 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, their scheme only realized the linean search efficiency. Xia et al. [16] provided EDMRS to support flexible dynamic operation by using balanced index tree that builded following the bottom-up strategy and "greedy" method, and they used parallel computing to improve search efficiency. However, when migrating to MOD, ordinary balanced binary tree they employed is not optimistic [6]. It is frustrating that the above solutions only support SSE 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 [17,18], research on SSE for MOD is increasingly active. Guo et al. [6] proposed MKRS_MO for MOD, they 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, so that the KDO is more suitable for MOD than the TF×IDF. However, they ignored the secure search scheme in *known background model* [1](a *threat model* that measures the ability of "honest but curious" cloud server [14,20] to evaluate private data and the risk of revealing private information in SSE system). Currently, SSE for MOD is still these challenges: **(1)** comprehensively optimizing query precision and privacy protection is difficult; **(2)** a large amount of different data from multiple data owners make the data features sparse, and the calculation of high-dimensional vectors can cause "curse of dimensionality"; **(3)** frequent updates of data challenge the scalability of dynamic system maintenance.

Our Contribution. This paper proposes 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 three techniques: **(1) optimal pseudo-keyword padding based on searching adversarial networks (SAN):** To improve the privacy protection strength of SSE is a top priority. Padding random noise into the data [1, 8, 19] is a current popular method designed to interfere with the analysis and evaluation from cloud server, which protects the document content and keyword information better.

However, such an operation will reduce the query precision [1]. In response to this, we creatively use *adversarial learning* [4] to obtain the *optimal probability distribution* for controlling pseudo-keyword padding and the *optimal game equilibrium* for the query precision and the privacy protection strength. This makes query precision exceeds 95% while ensuring adequate privacy protection, which is better than traditional SSE [1, 6, 8, 13, 16]; **(2) efficient search based on maximum likelihood search balanced tree (MLSB-Tree):** The construction of the index tree is the biggest factor affecting the search efficiency. If the leaf nodes of the index tree are sorted by maximum probability (the ranking of the index vectors from high to low depends on the probability of being searched), the computational complexity will be close to $\mathcal{O}(\log N)$ [7]. *Probabilistic learning* is employed to obtain MLSB-Tree, which is ordered in a maximum probability. The experimental evaluation shows that MLSB-Tree-based search is faster and more stable compare with related works [6, 16]; **(3) flexible dynamic system maintenance based on balanced index forest (BIF):** Using *unsupervised learning* [3, 10] to design a fast index clustering algorithm to classify all indexes into multiple index partitions, and a corresponding balanced index tree is constructed for each index partition, thus all index trees form BIF. Owing to BIF is distributed, it only needs to maintain the corresponding index partition without touching all indexes in dynamic system maintenance, which improves the efficiency of index update operations and introduces low overhead on the computation, communication and storage. In summary, MRSM_SAN increases the possibility of deploying dynamic SSE in real-world cloud storage systems.

Organization and Version Notes. Section 2 describes scheme. Section 3 conducts experimental evaluation. Section 4 discusses our solution. Compared with the preliminary version [2], this paper adds algorithms, enhances security analysis, and conducts more in-depth experimental analysis of the proposed scheme.

2 Secure and Efficient MRSM_SAN

2.1 System Model

The proposed system model consists of four parties, is depicted in Fig. 1. **Data owners** (DO) are responsible for constructing searchable index, encrypting data and sending them to cloud server or trusted proxy; **Data users** (DU) are consumers of cloud services. Based on *attribute-based encryption* [5], once DO authorize DU attributes related to the retrieved data, DU can retrieve the corresponding 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 over encrypted cloud data based on query requests, and returning *top-k* documents to DU. CS is considered"honest but curious" [14, 20], so that it is necessary to provide a secure search scheme to protect privacy. Our goal is to protect index privacy, query privacy and keyword privacy in dynamic SSE.

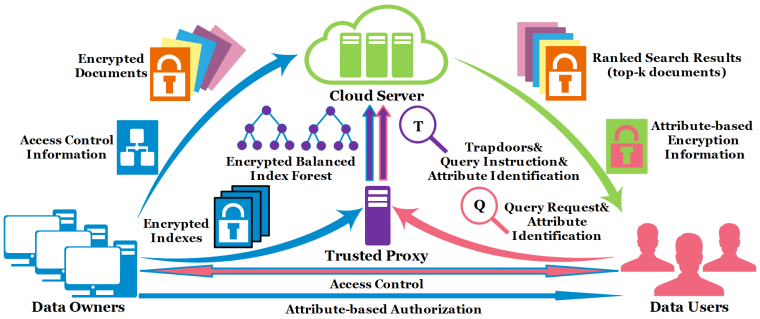


Fig. 1. The basic architecture of MRSM_SAN

2.2 MRSM_SAN Framework

***Setup*:** Based on index clustering results (s index partitions) and privacy requirements in *known background model* [1], TP determines the size N_i of sub-dictionary D_i, the number U_i of pseudo-keyword, sets the parameter $V_i = U_i + N_i$. Thus $V = \{V_1,\ldots,V_s\}$, $U = \{U_1,\ldots,U_s\}$, $N = \{N_1,\ldots,N_s\}$.

***KeyGen*(V):** TP generates 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 $V_i \times V_i$-dimensional invertible matrices, S_i is a random V_i-dimensional vector. Symmetric key $\overline{SK}_i = \{S_i, M_{i,1}^{-1}, M_{i,2}^{-1}\}$.

***Extended-KeyGen*(SK_i, Z_i):** For dynamic search [8, 16], if Z_i new keywords are added into the i-th sub-dictionary, TP generates a new key $SK_i' = \{S_i', M_{i,1}', M_{i,2}'\}$, where $M_{i,1}'$ and $M_{i,2}'$ are two $(V_i + Z_i) \times (V_i + Z_i)$-dimensional invertible matrices, S_i is a new random $(V_i + Z_i)$-dimensional vector.

***BuildIndex*(F, SK):** To realize secure search in *known background model* [1], TP pads U_i pseudo-keyword into weighted index $\overline{I}$ (associated with document F) to obtain secure index $\widetilde{I}$, and uses $\widetilde{I}$ and SK to generate BIF $\mathcal{F} = \{\tau_1,\ldots,\tau_s\}$ and encrypted BIF $\widetilde{\mathcal{F}} = \{\widetilde{\tau_1},\ldots,\widetilde{\tau_s}\}$. Finally, TP sends $\widetilde{\mathcal{F}}$ to CS.

***Trapdoor*(Q, SK):** DU send query requests (keywords and their weights) and attribute identification to TP. TP generates query $Q = \{Q_1,\ldots,Q_s\}$ and generates trapdoor $T = \{T_1,\ldots,T_s\}$ using SK. Finally, TP sends T to CS.

***Query*$(T, \widetilde{\mathcal{F}}, t, k)$:** TP sends query information to CS and specifies t index partitions to be queried. CS performs searches and retrieves *top-k* documents.

2.3 Algorithms for Scheme

1. ***Binary Index Generation*:** DO_i uses algorithm 1 to generate the binary index (vector) I_i for the document F_i, and sends I_i to TP.
2. ***Fast Index Clustering & Keyword Dictionary Segmentation*:** We employ algorithm 2 to solve "curse of dimensionality" issue in computing.
3. ***Weighted Index Generation*:** TP exploits the KDO weight model [6] to generate the weighted index, as shown in algorithm 3.
4. ***MLSB-Tree and BIF Generation*:** TP uses algorithm 4 to generate MLSB-Tree $\tau_1,\ldots,\tau_s$ and BIF $\mathcal{F} = \{\tau_1,\ldots,\tau_s\}$.

Algorithm 1 *Binary Index Generation*

Input: Document set $F = \{F_1, \ldots, F_m\}$, keyword dictionary $D = \{w_1, \ldots, w_n\}$.
Output: Binary index set $I = \{I_1, \ldots, I_m\}$.

1: **for** $i = 1$, **to** m **do**
2: Based on ***Vector Space Model*** [11] and keyword dictionary D, DO_i generates binary index $I_i = \{I_{i,1}, \ldots, I_{i,n_i}\}$ for document $F_i = \{F_{i,1}, \ldots, F_{i,n_i}\}$, where $I_{i,j}$ is a binary index vector.
3: **return** Binary index Index I

Algorithm 2 *Fast Index Clustering & Keyword Dictionary Segmentation*

Input: Binary index (vector) I from DO, where $I = \{I_1, \ldots, I_m\}$, $DO = \{DO_1, \ldots, DO_m\}$.
Output: s index partitions, s sub-dictionaries and new binary index $\widehat{I} = \{\widehat{I}_1, \ldots, \widehat{I}_s\}$.

1: *Local Clustering*: For $I = \{I_1, \ldots, I_m\}$, TP uses ***Twin Support Vector Machine*** [3] to classify the index vectors $\{I_{i,1}, \ldots, I_{i,n_i}\}$ in I_i into 2 clusters (i-th and $(m+i)$-th initial index partition) and obtain the representative vectors for the i-th and the $(m+i)$-th initial index partition. **return** $2m$ initial index partitions and their representative vectors.
2: *Global Clustering*: TP uses ***Manhattan Frequency k-Means*** [10] algorithm to group all initial index partitions (representative vectors) into s final index partitions. **return** s index partitions.
3: *Keyword Dictionary Segmentation*: According to the obtained s index partitions, the keyword dictionary D is divided into s sub-dictionaries $D_1, \ldots, D_s$ correspondingly, where $D_i = \{w_1^i, \ldots, w_{N_i}^i\}$. TP obtains new binary index $\widehat{I} = \{\widehat{I}_1, \ldots, \widehat{I}_s\}$, where $I_i = \{\widehat{I}_{i,1}, \ldots, \widehat{I}_{i,M_i}\}$, $\widehat{I}_{i,j}$ is a N_i-dimensional vector. Delete "**public redundancy zero element**" of all index vectors in the same index partition, thus the length of the index vector becomes shorter than before. **return** s sub-dictionaries and new binary index $\widehat{I}$ for s index partitions.

Algorithm 3 *Secure Weighted Index Generation*

Input: Binary index $\widehat{I}$ for s index partitions.
Output: Secure weighted index $\widetilde{I}$ for s index partitions, i.e. the data type is floating point.

1: *Correlativity Matrix Generation*: Using the corpus to determine the semantic relationship between different keywords and obtain the *correlativity matrix* $S_{N_i \times N_i}$ (symmetric matrix).
2: *Weight Generation*: Based on KDO [6], construct the average keyword popularity AKP about DO. Specifically, calculate AKP_i of DO_i with equation "$AKP_i = (P_i \cdot \widehat{I}_i) \otimes \alpha_i$", where the operator $\otimes$ denotes the product of two vectors corresponding elements, $\alpha_i = (\alpha_{i,1}, \ldots, \alpha_{i,N_i})$, if $|L_i(w_t^i)| \neq 0$, $\alpha_{i,t} = \frac{1}{|L_i(w_t^i)|}$, otherwise $\alpha_{i,t} = 0$ (where $|L_i(w_t^i)|$ is the number of documents contain keyword w_t^i that in i-th sub-dictionary D_i, $t \in \{1, \ldots, N_i\}$). Calculate the raw weight information for DO_i, $W_i^{raw} = S_{N_i \times N_i} \cdot AKP_i$, where $W_i^{raw} = (W_{i,1}^{raw}, \ldots, W_{i,N_i}^{raw})$.
3: *Normalized Processing*: Obtain 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}, calculate $W_{i,t} = \frac{W_{i,t}^{raw}}{W_{max}[j]}$
4: *Weighted Index Generation*: TP obtains weighted index vector with "$\overline{I}_{i,j} = \widehat{I}_{i,j} \otimes W_i$", where $\overline{I}_{i,j}$ associated with document $F_{i,j}$ corresponds to the i-th index partition ($j \in \{1, \ldots, M_i\}$).
5: *Secure Weighted Index Generation*: TP pads U_i pseudo-keyword into $\overline{I}$ in i-th index partition to obtain V_i-dimensional secure weighted index $\widetilde{I}$ with high privacy protection strength [16, 19].

Algorithm 4 *MLSB-Tree and BIF Generation*

Input: Secure weighted index $\widetilde{I}$ for s index partitions, randomly generated query vector Q.
Output: MLSB-Tree $\tau_1, \ldots, \tau_s$ for s index partitions and BIF $\mathcal{F}$ for all indexes belong to DO.

1: **for** $i = 1$, **to** s **do**
2: **for** $j = 1$, **to** M_i **do**
3: Based on **probabilistic learning**, TP calculates "$Score_{i,j} = \sum_{k=1}^{n} I_{i,j}^T \cdot Q_{i,k}$"; Then, TP sorts $I_{i,1}, \ldots, I_{i,M_i}$ according to $Score_{i,1}, \ldots, Score_{i,M_i}$; Finally, TP follows the bottom-up strategy and generates MLSB-Tree τ_i (balanced tree) with **greedy** method [16].
4: **return** MLSB-Tree $\tau_1, \ldots, \tau_s$.
5: **return** BIF $\mathcal{F} = \{\tau_1, \ldots, \tau_s\}$.

Algorithm 5 *Encrypted MLSB-Tree and Encrypted BIF Generation*

Input: BIF $\mathcal{F} = \{\tau_1, \ldots, \tau_s\}$ and key $SK = \{SK_1, \ldots, SK_s\}$, where $SK_i = \{S_i, M_{i,1}, M_{i,2}\}$.
Output: Encrypted BIF $\widetilde{\mathcal{F}} = \{\widetilde{\tau_1}, \ldots, \widetilde{\tau_s}\}$.

for $i = 1$, **to** s **do**
  TP encrypts MLSB-Tree τ_i with the secret key SK_i to obtain encrypted MLSB-Tree $\widetilde{\tau_i}$.
  for $j = 1$, **to** n **do**
    TP "splits" vector $u_{i,j}.v$ of $u_{i,j}$ (node of τ_i) into two random vectors $u_{i,j}.v_1$, $u_{i,j}.v_2$.
    if $S_i[t] = 0$ **then**
      $u_{i,j}.v_1[t] = u_{i,j}.v_2[t] = u_{i,j}.v[t]$.
    else
      if $S_i[t] = 1$ **then**
        $u_i.v_1[t]$ is a random value $\in (0, 1)$, $u_{i,j}.v_2[t] = u_{i,j}.v[t] - u_{i,j}.v_1[t]$.
    TP encrypts $u_{i,j}.v$ with reversible matrices $M_{i,1}$ and $M_{i,2}$ to obtain "$\widetilde{u_{i,j}.v} = \{\widetilde{u_{i,j}.v_1}, \widetilde{u_{i,j}.v_2}\} = \{M_{i,1}^T u_{i,j}.v_1, M_{i,2}^T u_{i,j}.v_2\}$", where $u_{i,j}.v_1$ and $u_{i,j}.v_2$ are V_i-length vectors
  return Encrypted MLSB-Tree $\widetilde{\tau_i}$.
return Encrypted Encrypted BIF $\widetilde{\mathcal{F}} = \{\widetilde{\tau_1}, \ldots, \widetilde{\tau_s}\}$.

Algorithm 6 *Trapdoor Generation and GDFS*$(T, \widetilde{\mathcal{F}}, t, k)$

Trapdoor Generation
Input: Query vector $Q = \{Q_1, \ldots, Q_s\}$.
Output: Trapdoor $T = \{T_1, \ldots, T_s\}$.

for $i = 1$, **to** s **do**
  TP "splits" query vector Q_i into two random vectors $Q_{i,1}$ and $Q_{i,2}$.
  if $S_i[t] = 0$ **then**
    $Q_{i,1}[t]$ is a random value $\in (0, 1)$, $Q_{i,2}[t] = Q_i[t] - Q_{i,1}[t]$.
  else
    if $S_i[t] = 1$ **then**
      $Q_{i,1}[t] = Q_{i,2}[t] = Q_i[t]$, where $t \in \{1, 2, \ldots, V_i\}$.
  TP encrypts $Q_{i,1}$ and $Q_{i,2}$ with reversible matrices $M_{i,1}^{-1}$ and $M_{i,2}^{-1}$ to obtain trapdoor "$T_i = \{\widetilde{Q}_{i,1}, \widetilde{Q}_{i,2}\} = \{M_{i,1}^{-1} Q_{i,1}, M_{i,2}^{-1} Q_{i,2}\}$".
return $T = \{T_1, \ldots, T_s\}$.

GDFS$(T, \widetilde{\mathcal{F}}, t, k)$
Input: Query$(T, \widetilde{\mathcal{F}}, t, k)$.
Output: *top-k* documents.

for $i = 1$, **to** s **do**
  if τ_i is the specified index tree **then**
    if $u_{i,j}$ is a non-leaf node **then**
      if $Score(\widetilde{u_{i,j}.v}, T_i) > \lceil \frac{k}{t} \rceil$-th score **then**
        GDFS($u_{i,j}$.high-child)
        GDFS($u_{i,j}$.low-child)
      else
        return
    else
      if $Score(\widetilde{u_{i,j}.v}, T_i) > \lceil \frac{k}{t} \rceil$-th score **then**
        Update $\lceil \frac{k}{t} \rceil$-th score for i-th index tree τ_i and the ranked search result list for $\widetilde{\mathcal{F}}$.
return The final *top-k* documents for $\widetilde{\mathcal{F}}$.

5. ***Encrypted MLSB-Tree and Encrypted BIF Generation.*** TP encrypts $\mathcal{F}$ using algorithm 5 and sends encrypted $\widetilde{\mathcal{F}}$ to CS. τ_i and $\widetilde{\tau_i}$ are isomorphic (i.e.$\tau_i \cong \widetilde{\tau_i}$) [16]. Thus, the search capability of tree is still well maintained.
6. ***Trapdoor Generation.*** Based on query request from DU, TP generates $Q = \{Q_1, \ldots, Q_s\}$ and $T = \{T_1, \ldots, T_s\}$ using algorithm 6, and sends T to CS.
7. ***Search Process.*** **(1)** *Query Preparation*: DU send query request and attribute identifications to TP. If validating queries are valid, TP generates trapdoors and initiates search queries to CS. If access control passes, CS performs searches and returns *top-k* documents to DU. Otherwise CS refuses to query. **(2)** *Calculate Matching Score for Query on MLBS-Tree* τ_i:

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

(3) *Search Algorithm for BIF*: the greedy depth-first search (GDFS) algorithm for BIF as shown in algorithm 6.

2.4 Security Improvement and Analysis

Adversarial Learning. Padding random noise into the data [1, 8, 19] is a popular method to improve security. However, pseudo-keyword padding that follows different probability distributions will reduce query precision to varying degrees [1, 8]. Therefore, it is necessary to optimize the probability distribution that controls pseudo-keyword padding. To address this, *adversarial learning* [4] for optimal pseudo-keyword padding is proposed. As shown in Fig. 2. ***Searcher Network S(ε)*** **:** The search result (SR) is generated by taking the random noise ε ($\varepsilon \sim p(\varepsilon)$, $p(\varepsilon)$ is the object probability distribution) as an input and performing a search, and supplies SR to the discriminator network $D(x)$. ***Discriminator Network D(x)*****:** The input has an accurate search result (ASR) or SR and attempts to predict whether the current input is an ASR or a SR. One of the inputs x is obtained from the real search result set distribution $p_{data}(x)$, and then one or two are solved. Classify problems 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, $S(\varepsilon)$ generates SR, and $D(x)$ (considered as adversary) considers the probability that $S(\varepsilon)$ produces ASR is 0.5, i.e. it is difficult to distinguish between padding and without-padding, thus it can achieve effective security [19].

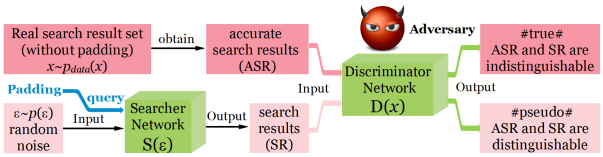


Fig. 2. Searching Adversarial Networks

Similar to GAN [4], 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 minimax game with value function $V(S, D)$:

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

Security Analysis. *Index confidentiality and query confidentiality:* ASPE approach [15] is widely used to generate secure index/query in privacy-preserving keyword search schemes [1, 6, 8, 13, 16] and its security has been proven. Since the index/query vector is randomly generated and search queries return only the

secure inner product [1] computation results (non-zero) of encrypted index and trapdoor, thus CS is difficult to accurately evaluate the keywords including in the query and matching *top-k* documents. Moreover, confidentiality is further enhanced as the optimal pseudo-keyword padding is difficult to distinguish and the transformation matrices are harder to figure out [15].

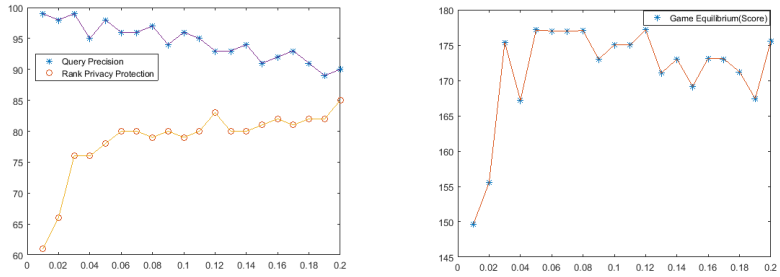
Query unlinkability: By introducing the random value ε (padding pseudo-keyword), the same search requests will generate different query vectors and receive different relevance score distributions [1, 16]. The optimal game equilibrium for precision and privacy is obtain by *adversarial learning*, which further improves query unlinkability. Meanwhile, SAN are designed to protect access pattern [19], which makes it difficult for CS to judge whether the retrieved ranked search results come from the same request.

Keyword privacy: According to the security analysis in [16], for i-th index partition, aiming to maximize the randomness of the relevance score distribution, it is necessary to obtain as many different $\sum \varepsilon_{\nu_i}$ as possible (where $\nu_i \in \{j|Q_i[j+N_i] = \alpha_i, j = 1, \ldots, U_i\}$; in [16], $\alpha_i = 1$). Assuming each index vector has at least 2^{ω_i} different $\sum \varepsilon_{\nu_i}$ choices, the probability of two $\sum \varepsilon_{\nu_i}$ share the same value is less than $\frac{1}{2^{\omega_i}}$. If we set each $\varepsilon_j \sim U(\mu'_i - \delta_i, \mu'_i + \delta_i)$ (*Uniform distribution*), according to the central limit theorem, $\sum \varepsilon_{\nu_i} \sim N(\mu_i, \sigma_i^2)$ (*Normal distribution*), where $\mu_i = \omega_i \mu'_i$, $\sigma^2 = \frac{\omega_i \delta_i^2}{3}$. Therefore, it can set $\mu_i = 0$ and balance precision and privacy by adjusting the variance σ_i^2 in real-world application. In fact, when $\alpha_i \in [0, 1]$ (floating point number), SAN can achieve stronger privacy protection.

3 Experimental Evaluation

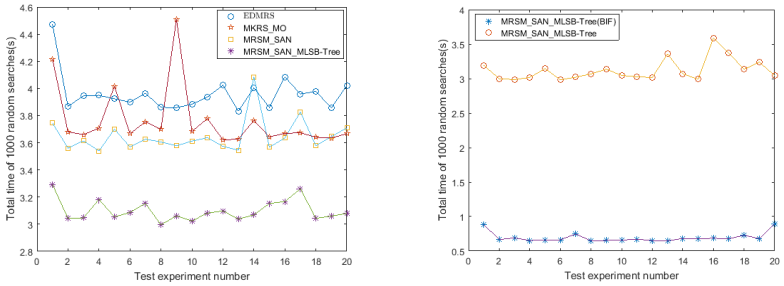
We implemented the proposed scheme using Python in Windows 10 operation system with Intel Core i5 Processor 2.40GHz and evaluated its performance on a real-world data set (academic conference publications provided by IEEE xplore https://ieeexplore.ieee.org/, including 20,000 papers and 80,000 different keywords, 400 academic conferences were randomly selected as data owners DO). All experimental results represent the average of 1000 trials.

Optimal Pseudo-keyword Padding. The parameters controlling the probability distribution (using SAN to find or approximate) are adjusted to find the optimal game equilibrium for *query precision* P_k (denoted as x) and *rank privacy protection* P'_k (denoted as y) (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 [1]). 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 shown in Fig. 3, we find the optimal game equilibrium ($\max f(x, y) = 177.5$) at $\sigma_1 = 0.05$, $\sigma_2 = 0.08$, $\sigma_3 = 0.12$. The corresponding query precision are: 98%, 97%, 93%. The corresponding rank privacy protection are: 78%,79%,84%. Therefore, we can choose the best value of σ to achieve optimal pseudo-keyword padding to satisfy query precision requirement and maximize rank privacy protection.



(a) (b)

Fig. 3. 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 $\sigma \in [0.01, 0.2]$.)



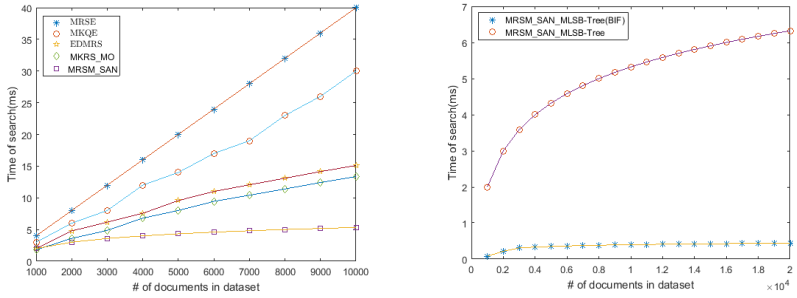
(a) (b)

Fig. 4. Time cost of query for 1000 random searches in 500 sizes of data set. **(a)** Comparison of tree-based search efficiency. Since the query is random, the search time fluctuates, which causes the curves in the graph to have intersections; **(b)** Comparison of MLSB-Tree and BIF search efficiency.

Search Efficiency of MLSB-Tree. Search efficiency is mainly described by query speed, and our experimental objects are index trees that are structured with different strategy: EDMRS [16] (ordinary balanced binary tree), MKRS_MO [6] (grouped balanced binary tree), MRSM_SAN(globally grouped balanced binary tree) and MRSM_SAN_MLSB-Tree. We first randomly generate 1000 query vectors, then perform search on each index tree respectively, finally take the results of 20 repeated experiments for analysis. As shown in Fig. 4(a), the query speed and query stability based on MLSB-Tree are better than other index trees. Compared with EDMRS and MKRS_MO, query speed increased by 21.72% and 17.69%. In terms of stability, MLSB-Tree is better than other index trees. (variance of search time(s): 0.0515 [6], 0.0193 [16], **0.0061**[MLSB-Tree])

Search Efficiency of BIF. As shown in Fig. 4(b), query 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-dimensional 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: $\eta = s\frac{\mathcal{O}(\log N)}{\mathcal{O}(\log N)-\mathcal{O}(\log s)}$,where s denotes the number of index partitions after fast index clustering, and N denotes the total number of documents. 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.



(a) (b)

Fig. 5. Time cost of query for the same query keywords (10 keywords) in different sizes of data set.{**(a)** Experimental results 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. It is worth noting that this is just the search performance based on MLSB-Tree; **(b)** Comparison of MLSB-Tree and BIF. Based on experimental analysis, it concludes that when data volume grows exponentially data features become more sparse, if all index vectors only rely on an index tree to complete the search task, the computational complexity will be getting farther away from $\mathcal{O}(\log N)$. Sparseness of data features makes the similarity between index vectors is mostly close to zero or even equal to zero, which brings trouble to the pairing of index vectors. Moreover, 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 *balanced binary tree* [6, 16]. We construct MLSB-Tree with *maximum likelihood method* and *probabilistic learning*. Interestinglythe closer the number of random searches is to infinity, the higher the search efficiency of obtained index tree, this makes the computational complexity of search can converge to $\mathcal{O}(\log N)$.}

Comparison of Search Efficiency (Larger Data Set). The efficiency of MRSM_SAN (without BIF) and related works [1, 6, 8, 16] are show as Fig. 5(a), and the efficiency of MRSM_SAN(without BIF) and MRSM_SAN(with BIF) are show as Fig. 5(b). It is more notable that the maintenance cost of scheme based

on BIF is much lower than the cost of scheme only based on a balanced index tree. When adding a new document to CS, we need to insert a new index vector (the node of the tree) in the index tree accordingly. If it is only based on an index tree, search complexity (search the location where the new index is inserted into the index tree) and update complexity (update the parent node corresponding to the new index) are both at least $\mathcal{O}(\log N)$ [9], the total cost is $2\mathcal{O}(\log N)$ (where N denotes the total number of documents). But BIF is very different, because we group all index vectors into s different index partitions and reduce their dimension. We assume that the number of index vectors in each index partition is equal, thus we need to spend the same update operation for each partition, which makes the cost is only $\frac{2}{s}\mathcal{O}(\log \frac{N}{s})$ and enables flexible dynamic system maintenance. Moreover, the increase in efficiency is positively correlated with the increase in data volume and data sparsity. For communication, compared with traditional SSE schemes [6,13,16], when the old index tree in the cloud is overwritten by the new index tree uploaded by DO/TP, our scheme only needs to update the specified index tree instead of the entire index forest. For storage, tree-based overhead is $(2^{\log N} - \frac{1}{2})/(2^{\log \frac{N}{s}} - \frac{1}{2}) \approx 2^{\log s}$ times forest-based.

4 Discussion

This paper proposes secure and efficient MRSM_SAN, and conducts in-depth security analysis and experimental evaluation. Creatively using *adversarial learning* to find *optimal game equilibrium* for query precision and privacy protection strength and combining traditional SSE with *uncertain control theory*, which opens a door for *intelligent SSE*. In addition, we propose MLSB-Tree, which generated by a sufficient amount of random searches and brings the computational complexity close to $\mathcal{O}(\log N)$. It means that using *probabilistic learning* to optimize the query result is effective in an *uncertain system* (owner's data and user's queries are uncertain). Last but not least, we implement flexible dynamic system maintenance with BIF, which not only reduces the overhead of dynamic maintenance and makes full use of distributed computing, but also improves the search efficiency and achieves fine-grained search. This is beneficial to improve the availability, flexibility and efficiency of dynamic SSE system.

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. 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)

2. Chen, K., Lin, Z., Wan, J., Xu, L., Xu, C.: Multi-client secure encrypted search using searching adversarial networks. IACR Cryptology ePrint Archive **2019**, 900 (2019)
3. Chen, S., Cao, J., Huang, Z., Shen, C.: Entropy-based fuzzy twin bounded support vector machine for binary classification. IEEE Access **7**, 86555–86569 (2019)
4. 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)
5. 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)
6. 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)
7. Knuth, D.E.: The art of computer programming, Volume III, 2nd Edition. Addison-Wesley (1998)
8. Li, R., Xu, Z., Kang, W., Yow, K., Xu, C.: Efficient multi-keyword ranked query over encrypted data in cloud computing. FGCS **30**, 179–190 (2014)
9. Poh, G.S., Chin, J., Yau, W., Choo, K.R., Mohamad, M.S.: Searchable symmetric encryption: Designs and challenges. ACM Comput. Surv. **50**(3), 40:1–40:37 (2017)
10. Salem, S.B., Naouali, S., Chtourou, Z.: A fast and effective partitional clustering algorithm for large categorical datasets using a k-means based approach. Computers & Electrical Engineering **68**, 463–483 (2018)
11. Salton, G., Wong, A., Yang, C.: A vector space model for automatic indexing. Commun. ACM **18**(11), 613–620 (1975)
12. 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)
13. 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)
14. 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)
15. 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)
16. 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)
17. Xu, L., Sun, S., Yuan, X., Liu, J.K., Zuo, C., Xu, C.: Enabling authorized encrypted search for multi-authority medical databases. IEEE TETC (2019). https://doi.org/10.1109/TETC.2019.2905572
18. Xu, L., Xu, C., Liu, J.K., Zuo, C., Zhang, P.: Building a dynamic searchable encrypted medical database for multi-client. Inf. Sci. (2019). https://doi.org/10.1016/j.ins.2019.05.056
19. Xu, L., Yuan, X., Wang, C., Wang, Q., Xu, C.: Hardening database padding for searchable encryption. In: IEEE INFOCOM 2019. pp. 2503–2511. IEEE (2019)
20. 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)