Preferred Keyword Search over Encrypted Data in Cloud Computing

Zhirong Shen, Jiwu Shu†, Wei Xue
Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, China
†Corresponding author: shujw@tsinghua.edu.cn
czy10@mails.tsinghua.edu.cn, xuewei@tsinghua.edu.cn

Abstract—Cloud computing cuts down large capital outlays in facilities purchase and eliminates complex system management for users. To protect data confidentiality in cloud utilization, sensitive data are usually stored in encrypted form, making traditional search service on plaintext inapplicable. Thus, enabling keyword search over encrypted data becomes a paramount urgency. Given massive data users with various search preferences, it becomes necessary to support preferred keyword search and output the data files in the order of the user’s preference. In this paper, for the first time, we investigate the challenging problem of preferred keyword search over encrypted data (PSED). We first establish a set of privacy requirements and utilize the appearance frequency of each keyword to serve as its "weight". A preference preprocessing mechanism is then explored to ensure that the search result will faithfully respect the user’s preference and the Lagrange polynomial is introduced to express the user’s preference formula. We further represent keyword weights of each file by using vectors, convert the preference polynomial into the vector form, and securely calculate their inner products to quantitatively characterize the relevance measure between data files and a query. Finally, an extensive performance evaluation demonstrates the proposed scheme can achieve acceptable efficiency.

I. INTRODUCTION

Cloud computing centralizes a large amount of materials and offers pay-as-you-use service. However, the data hosted in the cloud may suffer from the unsolicited access from both of the cloud server and other unauthorized users. To protect confidentiality, sensitive data are hosted in encrypted form in the cloud, making it different from the traditional data service based on plaintext keyword search. The trivial solution of downloading all the data and decrypting them locally is extremely expensive. Thus, exploring an efficient search service over encrypted data becomes a paramount urgency.

On one hand, the scale of massive files in the cloud requires flexible search query to retrieve accurate search results without receiving the unneeded files. On the other hand, given the large amount of users in cloud environment, different users may find different things relevant when searching because of different preferences [14], indicating the necessity of preferred search support to cope with users’ various preferences. Thus, exploring a flexible search service with preferred search support over encrypted data is extremely meaningful in cloud environment.

During last several years, searchable encryption (SE) [2]–[11] has been evolved in pursuit of search over encrypted data under different applications. For schemes [2], [8], [10] that realize the flexible search, they only support “Boolean keyword search” and pay limited attention to the relevance between files and a query. For schemes [9], [11] that enable ranked keyword search, they either just support single keyword search [9] or may return inaccurate results [11]. Even more important, most of the existing works ignore the user’s preference, easily leading to the following drawbacks. First, a user who does not have any pre-understanding about the encrypted data has to endure the labor-intensive task of manually picking out their interested files. Secondly, the naive search output without preference consideration will easily cause network congestion because of the transmission for all the matching files. Meanwhile, in the branch of information retrieval (IR) and database (DB), some search schemes with preference [13]–[17] have been proposed to quantify the retrieved files, however, they cannot be directly applied in the context of encrypted cloud data retrieval due to the limited attention on security and privacy for both queries and files. In short, the absence of preferred search with privacy preserved and flexible search query support is still a typical shortage in existing SE schemes.

In this paper, we study the problem of preferred keyword search over encrypted data (PSED) for the first time. We first specify a set of privacy requirements and use the appearance frequency of each keyword to a file to act as its weight. A flexible search query (e.g., the query over multiple keyword fields) is converted into polynomial form and the Lagrange polynomial is utilized to characterize the user’s preference query. Then we convert the search polynomial and the preference polynomial into vector forms, and propose a secure inner product computation mechanism to capture the correlation of files to the query. Thorough the analysis investigating efficiency and privacy is given, and the intensive evaluations on a real-world dataset demonstrate the efficiency of the proposed solution.

II. PROBLEM FORMULATION

A. System Model and Threat Model

Figure 1 presents the system model. The data owner hosts a collection of encrypted files \( C = \{ F_1, \ldots, F_{|C|} \} \) in the cloud and allows authorized users to search through them. The data user is the entity who wishes to fetch the files according to his interest. He should generate a search query \( Q \) and a search preference \( \mathcal{P} \), and request for the search trapdoor \( T_{Q,\mathcal{P}} \). The cloud server keeps data files and responds users’ search requests. When receiving \( T_{Q,\mathcal{P}} \), the server will locate the matching files by scanning the indexes \( I \), calculate corresponding relevance scores, and return the ranked result.
Like many previous works [9], [11] of SE, the cloud server is treated as "honest but curious", meaning the server will "honestly" execute the designed protocol, but he is also "curious" to learn the search query and the preferences.

B. Design Goals

**Flexible search query with preferences.** PSED should support preferred search over multiple keyword fields, including equality, range, and subset query over each keyword field, such as conjunctive normal form (CNF) policy.

**Index privacy.** The index privacy generally means the keywords (resp. keyword weights) should be kept secret against the server to prevent it from learning the keywords in the query and the file content (resp. the characteristics) of the files.

**Trapdoor privacy.** Trapdoor privacy can be partitioned into query privacy and preference privacy. Query privacy means the server cannot guess whether two trapdoors are generated from the same query, while preference privacy denotes the server should be unaware of the user’s preference for each keyword.

**Relevance privacy.** Though the cloud server knows the ranked order according to the calculated scores, it is forbidden to sense the actual relevance of each file to a query, otherwise it can deduce the file content based on the actual relevance.

**Efficiency** The proposed scheme should introduce lightweight operations to the user/owner, and promise the search efficiency.

C. Variables and Notations

**Preference and Relevance score.** Users can specify some numbers called preferences to characterize his different levels of interests for keywords. The larger preference generally means the higher priority order. Since keywords and their frequencies are practical tools to characterize the file content and their significance, the relevance of a file to a query can be divided into many "sub-relevances" to represent the "correlation" of the file to keywords in the query. We adopt the product of the preference and the keyword weight to server as this "sub-relevance", and take the accumulated "sub-relevances" to act as the relevance score of the file to the query. Therefore, we can express the keyword weight and the keyword preference in vector form (see Section III-A2), so that their inner product can achieve this kind of effect. Compared to the "inner product similarity" [11] model which ignores the importance of keyword frequency, we argue our model is quite more practical and reasonable.

**Secure inner-product calculation.** It means a user can specify a third party to compute the inner-product of two encrypted vectors $E(\vec{p})$ and $E(\vec{q})$ without learning the actual values in $\vec{p}$ and $\vec{q}$ by using random asymmetric splitting, so that $E(\vec{p})^T \cdot E(\vec{q}) = \vec{p}^T \cdot \vec{q}$. The reader can refer [12] for more detail.

### TABLE I

<table>
<thead>
<tr>
<th>Variables</th>
<th>Notations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{C}, \mathbb{U}$</td>
<td>The file collection, the number of keyword fields</td>
</tr>
<tr>
<td>$\mathbb{W}, \mathbb{W_i}$</td>
<td>The keyword space of $\mathbb{C}$, the keyword space of $F_i$</td>
</tr>
<tr>
<td>$\mathbb{W_i, w_{i,j}}$</td>
<td>The $i$-th keyword field, the $j$-th keyword of $\mathbb{W_i}$</td>
</tr>
<tr>
<td>$\pi_i, n_i$</td>
<td>The number of keywords in $\mathbb{W_i}, \sum_{i=1}^{n_i} \pi_i$</td>
</tr>
<tr>
<td>$\mathbb{Q, P}$</td>
<td>a search query, a search preference</td>
</tr>
<tr>
<td>$h_{\mathbb{W_i, p_j}}$</td>
<td>The weight of $w_{i,j}$, the preference of $w_{i,j}$</td>
</tr>
<tr>
<td>$\pi_q, n_q$</td>
<td>The number of keywords in $\mathbb{Q}$</td>
</tr>
<tr>
<td>$\mathbb{W_i, p_j}$</td>
<td>The keyword set in $\mathbb{W_i}$ whose preferences are $p_j$</td>
</tr>
<tr>
<td>$h_{\mathbb{W_i, p_j}}$</td>
<td>The sum of weights for the keywords in $\mathbb{W_i, p_j}$</td>
</tr>
<tr>
<td>$\mathbb{Q, P}$</td>
<td>A search query vector, a preference vector</td>
</tr>
<tr>
<td>$T_{\mathbb{Q, P}}$</td>
<td>The trapdoor derived from $\mathbb{Q}$ and $\mathbb{P}$</td>
</tr>
<tr>
<td>$F_{\mathbb{Q, P}}$</td>
<td>The search output of $T_{\mathbb{Q, P}}$</td>
</tr>
<tr>
<td>$</td>
<td>W_i</td>
</tr>
</tbody>
</table>

III. THE DESIGN OF PSED

In this section, we start with the preference preprocessing. Then the transformation from the preference query (resp., the search query) to the preference vector (resp., the search vector) will be presented. We will further give the detailed design of PSED and provide an analysis on security and efficiency.

A. Algorithm description

To characterize the matching equality among different files to a query, we first define the concept of "Priority order of files to a search trapdoor".

**Definition 1** (Priority order of files to a search trapdoor). For files $F_i$ and $F_j$, $F_i$ is prior to $F_j$ to $T_{\mathbb{Q, P}}$ if either of following two conditions establishes:

1. $F_i$ matches with the query $\mathbb{Q}$ and $F_j$ is rejected;
2. Both $F_i$ and $F_j$ match with $\mathbb{Q}$. There exists a preference $p_m$, and two keyword sets $\mathbb{W_i, p_m} \subset \mathbb{W_i}$ and $\mathbb{W_j, p_m} \subset \mathbb{W_j}$, such that $h_{\mathbb{W_i, p_m}} > h_{\mathbb{W_j, p_m}}$. For other preferences, if there is $p_z \in \mathbb{P}$ such that $p_z > p_m$, then there exist two keyword sets $\mathbb{W_i, p_z} \subset \mathbb{W_i}$ and $\mathbb{W_j, p_z} \subset \mathbb{W_j}$, such that $h_{\mathbb{W_i, p_z}} = h_{\mathbb{W_j, p_z}}$. Here, we denote the preference $p_m$ in case (2) as a "critical preference value (CPV)" between $F_i$ and $F_j$.

1. Preference Preprocessing: Adding the product of every keyword weight and its preference together to serve as the relevance score may not output the results that strictly follow the user’s search priority.

For example, there are two files $F_1$ and $F_2$ containing $w_{1,1}$ and $w_{1,2}$ in the field $W_1$ respectively and the corresponding keyword weights are $h_{1,1} = 10$ and $h_{1,2} = 4$, then for a query $(w_{1,1} \vee w_{1,2})$ with a preference ($p_{1,1} = 8 \vee p_{1,2} = 10$), the relevance score of $F_1$ is $p_{1,1} \times h_{1,1} = 80$ which is larger than that of $F_2$ (i.e., $p_{1,2} \times h_{1,2} = 40$), resulting in the unexpected deviation of ranked order. Therefore, the owner needs to preprocess the search preference when deriving the trapdoor, ensuring the search results will faithfully adhere to the user’s search preference.

To preprocess the preference, besides secret keys, the owner can keep some additional secret information, i.e., $\zeta_i = Max(h_{i,j})$, which represents the maximum keyword weight of the field $W_i$ among the collection $\mathbb{C}$. The preferences of the keywords which do not appear in the query are assumed
Lemma 1 (The detailed proofs are shown in the Appendix).

(1) If \( p_i < p_j \), then \( p_i \cdot \text{pre} \) is prior to \( p_j \).

The processor preprocesses the search preferences by taking the following steps. Firstly, the preference of each keyword in \( P \) would be sorted in an ascending order and be converted into the vector form, for instance, the formula (2) can be sorted as \((p_1, \ldots, p_d)\), where \( d = \sum d_i \) and a bijective function \( \phi(\cdot): \{i,j\} \in [1,|u|] \times [1,|d_i|] \rightarrow \{t\} \in [1,|d|] \) can be established. Then, we set \( p'_1 := p_1 \). For \( p_j \), if \( p_{j-1} < p_j \), suppose \( \phi^{-1}(z) = (i_j, z_j) \); else if \( p_{j-1} = p_j \), then \( p'_j := p_j \). Finally, we resort the preprocessed preferences back to the original CNF formula and get the updated preference
\[
\mathcal{P}': := (p'_1 \vee \cdots \vee p'_{d}) \wedge \cdots \wedge (p'_{u} \vee \cdots \vee p'_{n},d) \tag{3}
\]

An example. Suppose a original preference formula is \((2 \vee 1) \wedge (4 \vee 3)\), where \((\{1\}, \{2\}) \leq (3,2=\{3\})\). The formula will be transformed into the vector \((2,1,4,3)\) firstly and then be sorted into \((1,2,3,4)\). In the stage of preprocessing, \( p'_1 := p_1 = 1 \), \( p'_2 := 1 + p_1 \cdot \zeta_1 := 4 \), \( p'_3 := 1 + \sum_{i=1}^{3} p'_i \cdot \zeta_i := 16 \), \( p'_4 := 1 + \sum_{i=1}^{3} p'_i \cdot \zeta_i := 64 \). Finally, the updated vector \((1,4,16,64)\) will be flushed back into the original formula form \((4 \vee 3) \wedge (4 \vee 1) \wedge (4 \vee 1)\) according to the function \( \phi^{-1}(\cdot) \).

After the preprocessing, the file containing the larger preference will always be ranked higher without being affected by the keyword weight, which is proved by the Theorem 1 and Lemma 1 (The detailed proofs are shown in the Appendix).

Meanwhile, the preprocessing obeys “order preserving” rule, i.e., \( p'_i < p'_j \) (resp., \( p'_i = p'_j \)) will still establish in \( \mathcal{P}' \) if \( p_i < p_j \) (resp., \( p_i = p_j \)) holds in \( \mathcal{P} \).

Theorem 1. The original priority of each file will not be affected after the preprocessing of preference formula \( \mathcal{P} \).

Lemma 1. If \( F_i \) is prior to \( F_j \) to \( \mathcal{P} \), then the relevance score of \( F_i \) is larger than that of \( F_j \) to \( \mathcal{P} \) if both of them match \( Q \).

2) Preference Transformation: After producing the preprocessed preference formula (3), we first transform it into the Lagrange polynomial, ensuring the actual preference \( p'_i \) will be activated to involve in the calculation of “sub-relevance” with \( h_{i,j} \), only when the keyword \( w_{i,j} \) meets the condition over the keyword field \( W_i \). During the transformation, the owner will invoke a hash function \( \lambda(\cdot) \) to map keyword \( w_{i,j} \) to the number \( \lambda_{i,j} \) and employ Lagrange coefficients to obtain the polynomials as \( \sum_{i=1}^{w} \psi_i(x_i) \) for the keyword field \( W_i \), where \( \psi_i(x_i) = \sum_{j=1}^{d_i} \left( \frac{1}{(\lambda_{i,j} - x_1)(\lambda_{i,j} - x_2)\cdots(\lambda_{i,j} - x_{n,i})} \right) \right) \cdot \phi_{i,j} \cdot \lambda_{i,j} \cdot (x_i - \lambda_{i,j}) \right) \)

If \( w_{i,j} \) meets the requirement over the field \( W_i \), the “sub-relevance” will be correctly calculated as \( p'_{i,j} \); otherwise, an incorrect “sub-relevance” \( \varphi_i(x_i) \) will be produced, and the cloud server will fail to learn the relevance between the unsatisfied files and the query even it stealthily computes their relevance scores to the query. Then, the owner can draw out the coefficients of \( x_i^2 \) from equation (4) to denote the preference vector
\[
\mathbf{b'} := (b_1, b_2, \ldots, b_{u}, a_{u}, \ldots, a_{u}) \tag{4}
\]

where \( b_{i,j} \) is the coefficient of \( x _i^2 \) and please notice that \( b_{i,j} := 0 \) for \( d_i \leq j \leq n_i \). Suppose the keyword of file \( F_s \) is \( W = u_s, a_{u,s} \), then the corresponding keyword weight vector of \( F_s \) can be denoted as
\[
\mathcal{W}_s := (t_1, t_2, \ldots, t_{u}, a_{u,s}) \tag{5}
\]

where \( t_{i,j} := (h_{s,u}(i) \cdot a_{i,s} + \sum_{j=1}^{n_i} h_{s,u}(i) \cdot \varphi_{i,j}(x_i)) \). Finally, we resort the owner to the number \( r_{q \mathcal{W} \cdot e} \) to match the relevance score query. To mess the real relevance scores against the cloud server, we introduce random numbers \( r_p, r_q \), and \( \mathcal{W} \), then \( \mathcal{W} := (r_p \mathcal{W}, r_q) \), and enlarge \( W \) as \( W := (\mathcal{W}, \mathcal{e}) \), so that the calculated relevance score will be \( \mathcal{W} \cdot P := r_p \mathcal{W} \cdot P + r_q \mathcal{W} \). Here, we denote \( \mathcal{W} \cdot P \) and \( \mathcal{W} \cdot P \) as the “actual relevance score” and the “calculated relevance score” respectively. The random range \( r_q e \) is used to blind \( r_p \mathcal{W} \cdot \mathcal{P} \), otherwise \( r_p \) could be acquired simply through “greatest common divisor” computation if the cloud server obtains enough calculated relevance scores.

To achieve the accurate order even when the random values \( e \) is introduced, we can narrow the range of \( e \) to \([-1/2,1/2]\) and make \( r_q \leq r_p \). Since the relevance score \( \mathcal{W} \cdot P \) is represented by the integer, so that the disturbance of \( r_q e \) is still within the minimum distances \( r_p \) and the relative ranking between the matching files will be preserved.

3) Flexible search query support: To improve the storage efficiency, the weight vector can be directly reused to support the flexible search query just by taking the following steps.

For the search query as formula (1), the owner can choose a set of random values \( \{r_{i,j}\}_{i=1} \), invoke the hash function \( \lambda(\cdot) \) to map keyword \( w_{i,j} \) to the numeral \( \lambda_{i,j} \), and transform the query into the polynomial form as follows: \( r_{1}(x_1 - \lambda_{1,1}) \ldots (x_1 - \lambda_{1,d_1}) \ldots (x_u - \lambda_{u,1}) \ldots (x_u - \lambda_{u,d_u}) \).

The coefficient \( a_{i,d} \) can be derived, where \( a_{i,d} \) is the coefficient of \( x_i^2 \) and \( a_{i,0} = r_{i}(-1)^d \prod_{j=1}^{d} \lambda_{i,j} \). Finally, the query vector is unified as
\[
\mathcal{Q} := (a_{1,n_1}, a_{1,0}, \ldots, a_{u,n_u}, a_{u,0}) \tag{6}
\]

for the file \( F_s \) that is labeled with the keywords
\[
\{W_1 = u_1, s_1(1), \ldots, W_u = u_s, s_u(1)\},
\]

The outputs will equal to 0 with overwhelming probability if the file really matches the search query. The random values
Algorithm 1: The detailed description of PSED

Setup\((n, u)\):
1. generate two invertible \((n + u + 1) \times (n + u + 1)\) matrices \(M_1, M_2\), initiate a \((n + u + 1)\)-dimension binary vector \(\vec{S}\) and outputs \(SK := \{M_1, M_2, S\}\);

BuildIndex\((\{M_1, M_2\}, \{\vec{S}\}, C)\):
1. For \(F_i \in C\)
   i) generate the keyword weight vector \(\vec{W}_{i}\);
   ii) expand \(\vec{W}_{i}\) to \(\hat{\vec{W}}_{i} := (\hat{\vec{W}}_{i,1}, \vec{e})\), create two random shares of \(\hat{\vec{W}}_{i,1}\), i.e., \(\hat{\vec{W}}_{i,1,1}\) and \(\hat{\vec{W}}_{i,1,2}\) to meet the following conditions;
   iii) For \(j = 1\) to \((n + u + 1)\)
        if \(\hat{S}[j] = 1\), then \(\vec{W}_{i,1,1}[j] + \vec{W}_{i,2,1}[j] := \vec{W}_{i}[j]\);
        else, \(\vec{W}_{i,1,1}[j] := \vec{W}_{i,1}[j]\);
   iv) run \(\hat{\vec{W}}_{i,1} = M_1^T\hat{\vec{W}}_{i,1,1}\hat{\vec{W}}_{i,2,1} = M_2^T\hat{\vec{W}}_{i,1,2}\), and set \(I_i := \{\hat{\vec{W}}_{i,1}, \hat{\vec{W}}_{i,2}\}\);
2. upload the encrypted files \(\{F_i\} \in C\) and \(I := \{I_i\}\) to the cloud server;

GenTrapdoor\((\vec{Q}, \vec{P}, \{M_1, M_2\}, \{\vec{S}\})\):
1. preprocess the preference \(\vec{P}\) to \(\vec{P}'\), transform the query \(\vec{Q}\) and the preference \(\vec{P}'\) into \(\hat{\vec{Q}}\) and \(\hat{\vec{P}}\);
2. expand \(\hat{\vec{Q}}\) to \(\hat{\vec{Q}} := (\hat{\vec{Q}}, 0)\), randomly choose \(r_p, r_q\) \((r_q \leq r_p)\), and create \(\vec{P}' := (r_p \vec{P}, r_q)\);
3. create two random shares of \(\hat{\vec{Q}}\) and \(\vec{P}'\), i.e., \(\hat{\vec{Q}}_1\) and \(\vec{Q}_2\), \(\vec{P}_1\) and \(\vec{P}_2\) respectively;
4. For \(j = 1\) to \((n + u + 1)\)
   if \(\hat{S}[j] = 0\), then \(\vec{Q}_1[i] + \vec{Q}_2[i] = \hat{\vec{Q}}[i], \hat{\vec{P}}_1[i] + \hat{\vec{P}}_2[i] = \hat{\vec{P}}[i]\);
   else, \(\vec{Q}_1[i] := \vec{Q}_2[i] := \hat{\vec{Q}}, \hat{\vec{P}}_1[i] := \hat{\vec{P}}_2[i] := \hat{\vec{P}}[i]\);
5. compute \(T_{\vec{Q}[1]} = M_1^{-1}\vec{Q}_1, T_{\vec{Q}[2]} = M_2^{-1}\vec{Q}_2, T_{\vec{P}[1]} = M_1^{-1}\vec{P}_1, T_{\vec{P}[2]} = M_2^{-1}\vec{P}_2\);
6. return search trapdoor \(T_{\vec{Q}, \vec{P}} := (T_{\vec{Q}[1]}, T_{\vec{Q}[2]}), T_{\vec{P}} = (T_{\vec{P}[1]}, T_{\vec{P}[2]})\) to the user.

SearchIndex\((I, T_{\vec{Q}, \vec{P}})\):
1. For \(F_i \in C\), compute \(\vec{W}_{i}^{T}T_{\vec{Q}[1]} + \vec{W}_{i}^{T}T_{\vec{Q}[2]}\);
   i) if the calculated result is zero, calculate the relevance score \(\vec{W}_{i}^{T}T_{\vec{P}[1]} + \vec{W}_{i}^{T}T_{\vec{P}[2]}\);
   ii) rank the satisfied files and return \(F_{\vec{Q}}, \vec{P}\);

\(\{r_i\}_{i=1}^{n}\) can mess the distribution of the query vector, so that even \(\vec{Q}\) and \(\vec{Q}'\) are derived from the same search query \(\vec{Q}\), \(\vec{W}_{i}^{T}\vec{Q}\) and \(\vec{W}_{i}^{T}\vec{Q}'\) will differentiate as long as \(F_i\) is excluded both by \(\vec{Q}\) and \(\vec{Q}'\). In addition, we can expand \(\hat{\vec{Q}}\) to \(\hat{\vec{Q}} := (\hat{\vec{Q}}, 0)\), so that \(\vec{W}_{i}^{T}\hat{\vec{Q}} = (\vec{W}_{i}^{T}\hat{\vec{Q}}, \vec{e})^T \cdot (\hat{\vec{Q}}, 0) = \vec{W}_{i}^{T}\hat{\vec{Q}}\).

B. PSED: Privacy-Preserving Scheme

As an summary of the designs above, our proposed preferred keyword search scheme is shown in Algorithm 1 as detail.

Setup. The owner initiates the secret keys, including a binary vector and two invertible matrices, where \(n = \sum_{i=1}^{u} n_i\) and \(u\) is the number of keyword fields.

BuildIndex. The owner generates the keyword weight vector \(\vec{W}_{i}\) (step 1-i), expands \(\vec{W}_{i}\) to \(\hat{\vec{W}}_{i}\) (step 1-ii), divides the vectors (step 1-iii), and encrypts them (step 1-iv).

GenTrapdoor. Given a query \(\vec{Q}\) and its preference \(\vec{P}\), the owner converts \(\vec{Q}\) into the query vector \(\hat{\vec{Q}}\) and preprocess the preference (step 1). Then the owner enlarges \(\hat{\vec{P}}\) (resp., \(\vec{Q}\)) to \(\hat{\vec{P}}\) (resp., \(\vec{Q}\)) (step 2), splits the vectors (step 3, step 4), and encrypts them with the secret matrices (step 5).

SearchIndex. The server calculates the relevance score for matching files and returns the ranked results.

C. The Analysis

1) Efficiency Analysis: The stage of BuildIndex (resp. GenTrapdoor) calls for multiplications between two \((n + u + 1) \times (n + u + 1)\) matrices and one (resp. two) \((n + u + 1)\)-dimension vector for each file (resp. each query with preference). In SearchIndex, the cloud server will only compute two inner-products between four \((n + u + 1)\)-dimension vectors for each mismatched file. For the matching files, extra two inner-products between four \((n + u + 1)\)-dimension vectors are needed. With respect to storage overhead, the owner should only keep two \((n + u + 1) \times (n + u + 1)\) secret matrices (i.e., \(M_1, M_2\)), a vector whose lengths is \((n + u + 1)\) (i.e., \(\vec{S}\)), and some secret information (i.e., \(\vec{c}_i\) \(1 \leq i \leq n\)). The user should store the trapdoor which is constituted by four \((n + u + 1)\)-dimension vectors, while the cloud server is required to keep the encrypted collection as well as the encrypted indexes \(I\).

2) Privacy Analysis: As for the index privacy, because keywords and their corresponding weights are hashed by the secret hash function and then be encrypted by the secret matrices, the cloud server will find it difficult to deduce the meaning of every item in the index; this security is guaranteed by the computation complexity of secure \(kNN\) [12]. Based on the same principle, the requested keywords and the corresponding preferences will be invisible to the cloud server, thus the query privacy can be achieved.

Because of the randomized splitting and the introduction of some random values (e.g., \(\{r_i\}_{1 \leq i \leq u}\), \(r_p\), and \(r_q\)), the produced trapdoors will be various even to the same query. This non-deterministic property will also make the cloud server have trouble mining the relationship between two trapdoors by comparing them directly. Though the cloud server can compare the matching files and ranked results to judge whether the targeted queries have internal correlation, this attack will fail if some puppet files to mess the search outputs are introduced.

3) Relevance privacy: With the protection of random values, the calculated scores of the matching file \(F_i\) will be \((r_p\vec{W}_i^{T}\hat{\vec{P}} + r_p\vec{e}_i), (\hat{\vec{Q}}, 0) = \vec{W}_i^{T}\hat{\vec{Q}}\).

For the unsatisfied files to a query, \(\vec{W}_i^{T}\hat{\vec{P}}\) will produce incorrect relevant scores, because the weight of the excluded keyword will participate in the calculation and roil the final result, increasing the hardness for the server to learn the relevance of the unsatisfied files to a query.
Fig. 2. The time cost of building index. (a) For the different amount of files in the whole collection, when $n=5000, n_Q=100$. (b) For the different number of keywords in the collection, when $n_Q=100, |C|=5000$.

Fig. 3. The time cost of trapdoor generation. (a) For the different number of keywords in the query, when $n=5000, |C|=1000$. (b) For the varying number of keywords in the whole collection, when $n_Q=100, |C|=5000$.

Fig. 4. The time cost of query search. (a) For the different number of files in the collection, when $n=5000, n_Q=100$. (b) For the different amount of keywords in the collection, when $n_Q=100, |C|=5000$. (c) For the varying number of keywords in the query, when $n=5000, |C|=1000$.

### IV. PERFORMANCE EVALUATION

We utilize the "Connectionist Bench (Nettalk Corpus) Data Set" from UCI Machine Learning Repository [1] which has 4 keyword fields to evaluate the performance of PSED. We fully realize PSED on the modern server equipped with 2.10GHz Intel @ Core 2 Duo CPU and 4GB RAM. The OS running on the server is Ubuntu (version: 11.04) and the kernel is Linux Ubuntu 2.6.38-8-generic. We choose MRSE [11] to act as the reference, where $U$ in MRSE-II is chosen to be 40.

**A. Index Building**

Figure 2(a) indicates the time to build indexes is linear with the number of files in the collection, since PSED has to generate an encrypted index for each file. Meanwhile, when $n$ varies, the time to generate indexes in PSED scales as $O(n^3)$, because it is usually required to implement $O(n^2)$ multiplications and $O(n^2)$ addition operations per index generation when $u$ is fixed. From the comparison, the time to generate an encrypted index in PSED is nearly the same with that in MRSE. Since it is a one-time cost to build indexes for the fileset, we argue that the efficiency is quite reasonable. In Table II, we compare the storage space of every index in PSED with that in MRSE and indicate they are approximately same.

**B. Trapdoor generation**

In Figure 3(a), the number of keywords in the query will not affect the performance of trapdoor generation very much. Figure 3(b) indicates the time to generate a trapdoor will be greatly affected by the total number of keywords in the collection, since it is required to encrypt the query vector and the preference vector, involving $O(n^2)$ multiplications and $O(n^2)$ addition calculations when $u$ is fixed. Meanwhile, the performance of PSED is a bit slower than that of MRSE for the following reasons. Firstly, PSED has to preprocess the preferences before generating the corresponding trapdoor, while MRSE only produces the trapdoor by encrypting the query vector. Secondly, PSED has to afford additional encryption of $\mathcal{P}$ to support the preferred search when compared to MRSE.

**C. Query**

The time in SearchIndex can be divided into the time to test whether an index matches the query and the time of relevance score calculation. We use "hit rate" to serve as the rate that an index matches the query and consider the search time under different "hit rates". We run three different performance tests as illustrated in Figure 4. In Figure 4(a), the search time is linear to the number of files in the fileset, since it is required to scan every file to test whether it matches the query conditions. Figure 4(b) indicates the search time is linear to the number of keywords $n$ among the whole collection, since $O(n)$ multiplications and $O(n)$ addition operations are performed in the steps of both matching test and relevance score calculation when $u$ is fixed. Figure 4(c) shows the number of keywords in the query will not affect the performance of query search. Meanwhile, the query time of PSED is a bit larger than that of MRSE and the gap will narrow as the hit rate drops, because it usually incurs four inner-products calculation if the query "hits" a index in PSED, while in MRSE the inner-product computation only happens twice.

**D. Complexity analysis**

From Table III, the computation complexity of PSED is close to that of MRSE. Different from MRSE which provides multi-keyword similar search and may return inaccurate search

<table>
<thead>
<tr>
<th>$n$</th>
<th>1000</th>
<th>3000</th>
<th>5000</th>
<th>8000</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRSE-Index (KB)</td>
<td>8.15</td>
<td>23.76</td>
<td>39.38</td>
<td>62.82</td>
</tr>
<tr>
<td>PSED-Index (KB)</td>
<td>7.85</td>
<td>23.48</td>
<td>39.10</td>
<td>62.54</td>
</tr>
<tr>
<td>MRSE-Trapdoor (KB)</td>
<td>15.70</td>
<td>46.96</td>
<td>78.20</td>
<td>125.08</td>
</tr>
</tbody>
</table>

Moreover, the owner can either distribute the secret matrices to authorized users or delegate the job of trapdoor generation to a Trusted Third Party to mitigate the computation burden at the own’s side. In addition, a trapdoor of PSED takes up nearly double the amount of storage space than that of MRSE as shown in Table II, since both the query vector and the preference vector should be encrypted in PSED.
TABLE III
THE COMPARISON OF COMPLEXITY BETWEEN MRSE AND PSED.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Time of BuildIndex/(Gen Trapdoor)</td>
<td>$O((n + U)u^2)$</td>
<td>$O((n + u)^2)$</td>
</tr>
<tr>
<td>Time of SearchIndex</td>
<td>$O(n + u)$</td>
<td>$O(n + u)$</td>
</tr>
<tr>
<td>Flexible query support</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Preferred keyword search</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Accurate search result</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

outputs, PSED focuses on preferred search over multiple fields, which aims to locate the accurate matching files and rank them according to the calculated scores. The performance difference between PSED and MRSE lies in the change of $U$ and $u$, where $u$ is introduced in keyword weight vector generation in PSED and $U$ is used to resist known-background attack in MRSE. Compared to MRSE, PSED is more suitable to applied in the environment with both preference consideration and high-accuracy requirement.

V. RELATED WORK

A. Searchable Encryption


B. Keyword Search with Preference

Leubner et al. [13] showed prioritization would be explained as the subspace preferences in vector space model. Koutrika et al. [14] established a preference model and presented progressive personalized answers. Chomicki et al. [15] presented the framework for formulating preferences. Kiessling et al. [16] proposed strictly partial order semantics for preferences. Georgiadis et al. [17] defined the preorders over attributes and explored the semantics of preferences expression. However, most of existing search schemes with preference are inapplicable on ciphertext.

VI. CONCLUSION

In this paper, we address the problem of preferred keyword search over encrypted data. We first establish a set of designed goals and use the occurrence of each keyword to characterize its significance to the file. Then we represent the query and index in the vector form, and employ secure inner-product computation to calculate the inner-product of the weight vector and the preference vector to quantitatively characterize the correlation of files to the query. Thorough analysis concerning privacy and efficiency is presented, and the intensive evaluation of PSED on a modern server demonstrates its suitability when deployed in real application.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (Grant No. 60925006), the State Key Program of National Natural Science of China (Grant No. 61232003), the National High Technology Research and Development Program of China (Grant No. 2013AA013201), and the research fund of Tsinghua-Tencent Joint Laboratory for Internet Innovation Technology.

REFERENCES


APPENDIX

Proof of Theorem 1. If $F_i$ is prior to $F_j$ to $T_{Q,p}$, then we will prove $F_i$ is still prior to $F_j$ to $T_{Q,p'}$. In the case of Definition1.1, the conclusion is obviously established. In the case of Definition1.2, because the keyword weights of $F_i$ and $F_j$ are unchanged, so $h_{W_i,p_m} > h_{W_j,p_m} > h_{W_j,p_z}$ still holds, where $p_m$ is the original critical preference value (CPV) and $p_z'$ is the preprocessed preference of $p_m$. According to the order-preserving rule of the preprocessing, if there exists a value $p_z' > p_m$, then we have $h_{W_i,p_z'} = h_{W_j,p_z}$, thus $p_z'$ will be the CPV of $P'$. Since $h_{W_i,p_z'} > h_{W_j,p_z'}$, then $F_i$ is still prior to $F_j$ to $T_{Q,p'}$. $\square$

Proof of Lemma 1. If both of $F_i$ and $F_j$ match $Q$, assume the CPV between $F_i$ and $F_j$ is $p_m$. Suppose $\phi^{-1}(z) = \{s_z, t_z\}$, since $\{h_{W_i,p_z}\}$ will be integers, then the difference of the relevance score between $F_i$ and $F_j$ will be $\sum_{p_z'} h_{W_i,p_z'} - \sum_{p_z'} h_{W_j,p_z'} = \sum_{p_z'} h_{W_i,p_z'} - \sum_{p_z'} h_{W_j,p_z'} = \sum_{p_z'} h_{W_i,p_z'} - \sum_{p_z'} h_{W_j,p_z'}$, since $\{h_{W_i,p_z}\}$ will be integers, then the difference of the relevance score between $F_i$ and $F_j$ will be $\sum_{p_z'} h_{W_i,p_z'} - \sum_{p_z'} h_{W_j,p_z'} = \sum_{p_z'} h_{W_i,p_z'} - \sum_{p_z'} h_{W_j,p_z'}$. $\square$