Front. Comput. Sci., 2018, 12(3): 593–607
https://doi.org/10.1007/s11704-016-6244-5

RESEARCH ARTICLE

Preferred search over encrypted data

Zhirong SHEN, Jiwu SHU (✉), Wei XUE

Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

Abstract Cloud computing provides elastic data storage and processing services. Although existing research has proposed preferred search on the plaintext files and encrypted search, no method has been proposed that integrates the two techniques to efficiently conduct preferred and privacy-preserving search over large datasets in the cloud.

In this paper, we propose a scheme for preferred search over encrypted data (PSED) that can take users' search preferences into the search over encrypted data. In the search process, we ensure the confidentiality of not only keywords but also quantified preferences associated with them. PSED constructs its encrypted search index using Lagrange coefficients and employs secure inner-product calculation for both search and relevance measurement. The dynamic and scalable property of cloud computing is also considered in PSED. A series of experiments have been conducted to demonstrate the efficiency of the proposed scheme when deploying it in real-world scenarios.

Keywords preferred search, encrypted data, secure inner-product calculation

1 Introduction

Recent years have witnessed the rapid development of cloud computing. As a new computing paradigm, it centralizes a large amount of computing and storage resources and offers pay-as-you-use services to customers with varying resource demands [1,2]. Owing to the flexibility, cost saving, and convenience, users are increasingly interested in migrating their data to the cloud for storage and processing. However, data in the cloud have the risk of unauthorized access from both inside and outside of the data center providing the cloud service, probably leaking the data owners' private or personal information [3–5]. To protect data confidentiality, encryption is adopted before uploading data to the cloud [6,7]. Whereas privacy is preserved with data encryption, some data processing operations should still be permitted without having the data decrypted in advance. Among these operations, data search is the most commonly used and important one allowing users to identify their information of interest from the entire dataset and obtain them from the cloud. Furthermore, users submit search requests to the cloud by providing one or a number of keywords. As different users may have different education experiences, professions, and interests, even for the same set of keywords in requests submitted by different users, different keywords may carry different preferences, or their relative importance among all keywords in a request [8]. Therefore, it is important to enable search over encrypted data in the cloud and support preferred search (PS) in which a search request includes keywords and their respective preferences.

Whereas both preferred search (PS) and searchable encryption (SE) are highly demanding techniques, they have received great attention and have been well developed in recent years as two independent research topics. In the most common practice of SE today, a search query is first encrypted as a *trapdoor* and then sent to the cloud server. To determine whether a file in the server matches the query, the server takes the trapdoor and the file's encrypted index as inputs and carries out a series of pre-designed operations in the protocol. The cloud server finally returns all the matching files without knowing the keywords in the trapdoor or in the encrypted

Received April 28, 2016; accepted November 11, 2016

E-mail: zhirong.shen2601@gmail.com; {shujw, xuewei}@tsinghua.edu.cn

indices. In parallel with increasing use of encrypted search, the PS technique has also been developed to cope with users' individual interests or preferences. Preferences can be represented in various forms, such as numerical values and strict partial orders, and are usually associated with individual keywords in a query for the server to measure the relevance (e.g., the relevance scores in our paper) between the query with explicitly specified preference and a set of files. Assuming that the relevance scores accurately reflect users' search interests, the server only needs to return the top-k files ranked by the scores, saving both the network bandwidth for transmitting the search results and the users' time in identifying files matching their true interests.

Though both encrypted search and PS are in high demand, and techniques on both topics are well developed, the issue of how to provide the two services at the same time, or PS over encrypted cloud data, has received limited attention. The direct combination of existing ranked schemes (e.g., [9]) and search schemes (e.g., [10]) over encrypted data will not only increase unnecessary storage overhead but also cause the loss of privacy for the unmatched files. In fact, the realization of PS over encrypted cloud data still has a number of challenges to address.

First, implementation of PS on encrypted data requires the calculation of a relevance score between the trapdoor of a query and an encrypted index. In the calculation of the relevance score, efforts must be made to prevent unauthorized parties from learning a keyword weight and uniqueness in a data file and search preference specified in a query.

Second, in the calculation of the relevance score, the score itself should be concealed. However, the score is used for generating the file list to return as the search result. Fully concealing the scores in the process of score ranking and file selection may compromise the search precision. We need to balance the confidentiality and the search precision.

Third, cloud computing is well known for its scalable and dynamic properties; therefore, special attention should be paid to making the scheme compatible with these properties.

In this paper, we propose a scheme for preferred search over encrypted cloud data, called PSED, that supports the ranking of the matching files according to their relevance to the users' preferences. To characterize both the frequency and uniqueness of each keyword in an index, PSED first assigns each keyword with a weight evaluated by the term frequency (TF) × inverse document frequency (IDF) model. PSED then expresses the search query and the user's preference in vector form and employs secure inner-product calculation [11] to perform search and relevance score calculation without leaking the index information (including keywords and keyword weights) or the query information (including keywords and their preferences).

Our contributions can be summarized as follows.

1) We establish a system framework of PS over encrypted data in the cloud scene and specify the requirements in terms of efficiency and privacy.

2) We use Lagrange coefficients to construct indices that can support search over multiple keyword fields and enable correct relevance calculation. We express the query and user's preference in vector form and adopt secure inner-product calculation to securely perform search and relevance score calculation. Moreover, PSED is also compatible with the scalable and dynamic property of cloud computing.

3) We have conducted a thorough analysis on efficiency and privacy protection provided by PSED, and carried out extensive experiments with a real-world dataset to demonstrate the applicability of PSED in real-world scenarios.

The remainder of this paper is organized as follows. In Section 2, we first present the system formulation. Then, we describe the detailed design of PSED in Section 3. Section 4 provides intensive performance evaluations. Finally, we discuss the related works in Section 5 and conclude our work in Section 6.

2 Problem formulation

2.1 System model

We mainly consider a secure sharing service of cloud data among three parties as illustrated in Fig. 1.

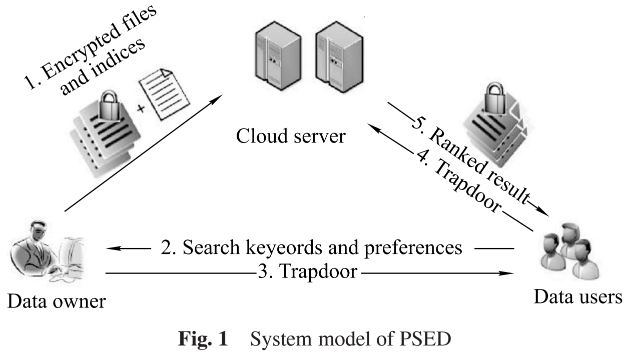


Fig. 1 System model of PSED

The *data owner* hosts a collection of encrypted data files $C = \{F_1, \ldots, F_{|C|}\}$ in the cloud and allows authorized users to search through them. To maintain the search capability for C, every file will be encrypted and tied with an encrypted index, which includes representative keywords and labels a weight for every keyword to characterize its significance (Step 1 in Fig. 1).

The *data user* wishes to retrieve the files according to his/her interest. To accomplish this procedure, he/she should generate interested keywords to constitute a search query Q, label his/her preference on each keyword to construct a search preference P, and send both P and Q to the owner (Step 2). To conceal Q and P against the cloud server, the owner will encrypt them by using his/her secret keys and send back a search trapdoor $T_{Q,P}$ (Step 3). The data user then submits $T_{Q,P}$ to the cloud for file retrieval (Step 4).

The *cloud server* is in charge of keeping the encrypted data files along with the associated encrypted indices. When receiving $T_{Q,P}$, it will pick out the files whose indices match the search query Q (i.e., all the keywords of the file are requested in Q), calculate corresponding relevance scores, and return the files ranked in decreasing orders of relevance scores (Step 5). To reduce the unnecessary communication, the user can issue a custom value k along with $T_{Q,P}$ to request for the matching files whose relevance scores are ranked in the top-k. Finally, the user can obtain the corresponding decryption keys distributed by the existing access control mechanisms [12] in the cloud scenario, which is devoted to decreasing the key management burden on the data owner.

In fact, some proxy servers can be introduced to undertake the trapdoor generation for users' queries. This methodology has been applied in some works of SE, such as [13] and [14]. The data owner can choose to tightly grasp the capability of trapdoor generation or delegate it to other parties, just according to his/her preference on security or efficiency.

Similarly to the approaches presented in [9, 15–17], PSED temporally requires the data owner to take charge of trapdoor generation. PSED can also be deployed in the scenario with multiple proxy servers, and this work is orthogonal to the main purpose of this paper.

2.2 Design goals

To realize PS over cloud data in real-world scenarios, our work should achieve the following security and performance goals.

• **Multi-field search query with preferences** The system should support the search query with preference over multiple keyword fields, including equality, range, and subset query over each keyword field, similar to the *conjunctive normal form* (CNF) policy. This type of search query is frequently experienced in real-world applications [13].

For example, a data user can issue a search query such as `(Topic="meeting")`$\wedge$`(Time`$\in$`[9:00am, 11:00am])` with the preference ($p_{meeting} = 1, p_{[9:00am,11:00am]} = 2$), where "`Topic`" and "`Time`" are two keyword fields and "`meeting`" and $p_{meeting}$ are the keyword and the corresponding preference, respectively. We also call the keyword field "`Topic`" (respectively, "`Time`") the first (respectively, second) keyword field in the query.

• **Index privacy** A primary goal is to protect the keywords and the corresponding weights against the cloud server. Otherwise, the cloud server may guess the file content and deduce the characteristic of the file if some keywords and their weights are revealed. In this work, index privacy indicates that it can resist the known-plaintext attack.

• **Trapdoor privacy** Another security goal is to protect trapdoor privacy, which is classified into query privacy and preference privacy. In this paper, trapdoor privacy indicates that it can resist the known-plaintext attack.

• **Relevance privacy** Given that the cloud server might accumulate the relevance scores of the matching files to a trapdoor and conduct the statistical analysis to estimate the differences of keyword weights among these files, the real relevance between the matching files and a trapdoor should be concealed. The cloud server is only allowed to know rank orders of the matching files to a trapdoor without sensing their real relevance scores. For the same reason, the real relevance between the mismatching files and a trapdoor should also be concealed.

• **Efficiency** The scheme should introduce lightweight overhead to both users and the owner, and guarantee the search efficiency.

2.3 Notation

2.3.1 Preference and relevance score

A user's preference is represented by numerical values in this paper. A keyword with a larger numerical value usually indicates that it receives a higher preference. We then use the term *relevance score* to quantitatively characterize the relevance of the matching files to the user's preference.

2.3.2 Secure inner-product calculation

To calculate relevance scores without revealing the weights and the user's preferences, we adopt secure inner-production calculation in which a user can specify a semi-trusted party to compute the inner-product of two encrypted vectors $E(\vec{p})$ and $E(\vec{q})$ without learning values in $\vec{p}$ and $\vec{q}$, so that $E(\vec{p})^{\mathrm{T}} \cdot E(\vec{q}) = \vec{p}^{\mathrm{T}} \cdot \vec{q}$. Wong et al. [11] studied the secure k-nearest neighbor (kNN) computation on encrypted databases and introduced a random asymmetric splitting method. We refer the interested reader to [11] for more background information. The analysis given in [11] shows the search space would be sufficiently large when the length of employed binary vector for encryption in [11] (i.e., the $\vec{S}$ in the following Algorithm: Design of PSED) is larger than 80 and the security of 1024-bit RSA (Rivest-Shamir-Adleman) keys is roughly equivalent as 80-bit symmetric keys as the general consensus indicates. In this work, the secure inner-product calculation is employed as the *black box* to generate the encrypted index and trapdoor.

3 Design of PSED

In this section, we first introduce the TF×IDF model to evaluate keyword weights. Then, to utilize the secure inner-product calculation, the preference query (respectively, the search query) should be transformed into the preference vector (respectively, the query vector). Finally, the detailed design of PSED is presented and an analysis on security and efficiency is given. To help readers gain a better understanding of the method in this paper, we list the frequently used symbols and descriptions in Table 1.

Table 1 Frequently used symbols and descriptions

Symbol	Description
C	File collection
$\lvert C\rvert$	Number of files in C
E_i	ith keyword field
$w_{i,j}$	Keyword value
u	Number of keyword fields
n	Total number of keywords
n_i	Number of keywords over the ith keyword field
Q, P	Search query, search preference
$h_{i,j}$	Weight of $w_{i,j}$
$p_{i,j}$	Preference of $w_{i,j}$
n_Q	Number of keywords in Q, i.e., $\sum_{i=1}^{u} d_i$
$\vec{Q}, \vec{P}$	Search query vector, preference vector
$T_{Q,P}$	Trapdoor derived from Q and P

3.1 Algorithm description

3.1.1 TF×IDF model

We first select the TF×IDF model to measure the significance of every keyword in a file. It is widely used to retrieve relevant data files based on TF (Term Frequency, the occurrence frequency of a term in a file) and IDF (Inverse Document Frequency, the universality of a keyword across all the data files). TF×IDF model follows the rule that the significance (called the *weight* in this paper) of a keyword to a file should increase with its occurrence frequency in this file, but decrease with the appearance frequency among other files [18]. From [19] (Chapter 4), the weight of the keyword $w_{i,j}$ in the file F can be evaluated by the equation:

$$h_{i,j} := \frac{1}{L} \cdot (1 + \ln A_{i,j}) \cdot \ln\left(1 + \frac{|C|}{|\mathcal{F}_{i,j}|}\right). \tag{1}$$

In this equation, $|C|$ denotes the number of data files in the collection C, $|\mathcal{F}_{i,j}|$ is the number of files that contain the keyword $w_{i,j}$, L is the length of the file F, and $A_{i,j}$ represents the appearance frequency of $w_{i,j}$ in file F. Figure 2 illustrates an index of a mail dataset in plaintext.

Keyword field	Topic	Date	Location	Label
Keyword	Meeting	7/1/2013	Beijing	Urgent
Keyword weight	2.12	3.56	1.36	6.45

Fig. 2 Index of mail datasets in plaintext

After introducing the weight measurement, we show how to express the keywords, the user's query, and the user's preference in the vector form so that we can adopt secure inner-product calculation to securely perform search and relevance score calculation.

Without loss of generality, a general search query over multiple keyword fields can be expressed as $Q := (E_1 = w_{1,1} \vee \cdots \vee w_{1,d_1}) \wedge \cdots \wedge (E_u = w_{u,1} \vee \cdots \vee w_{u,d_u})$, where E_i $(1 \leqslant i \leqslant u)$ is the ith keyword field, u is the number of keyword fields, $w_{i,j}$ is the jth requested keyword over E_i, and d_i is the number of requested keywords in Q over E_i. Assume that the assigned preference of keyword $w_{i,j}$ in Q is $p_{i,j}$, then the search preference of Q can be expressed as $P := (p_{1,1} \vee \cdots \vee p_{1,d_1}) \wedge \cdots \wedge (p_{u,1} \vee \cdots \vee p_{u,d_u})$.

Therefore, for a search trapdoor $T_{Q,P}$, suppose a file F_s associates with keywords $W_s := (E_1 = w_{1,s_1}, \ldots, E_u = w_{u,s_u})$, where w_{i,s_i} is the keyword of F_s over the ith keyword field E_i $(1 \leqslant i \leqslant u)$ and each keyword w_{i,s_i} is assigned with a weight h_{i,s_i}. We call "W_s matches with Q" if all the keywords in W_s are requested in Q. Therefore, if W_s matches with Q, then P should have the corresponding preference p_{i,s_i} for each keyword w_{i,s_i}. As referred in previous works [**?**, 20, 22], the relevance score between the file and the preference can be measured by their product as follows:

$$\mathrm{R}(T_{Q,P}, F_s) := \sum_{i=1}^{u} h_{i,s_i} \cdot p_{i,s_i}. \tag{2}$$

Moreover, we also denote $h_{i,s_i} \cdot p_{i,s_i}$ as the *sub-relevance* of $T_{Q,P}$ and F_s on the ith keyword field, as it denotes the contribution of the ith keyword field in the relevance score calculation.

3.1.2 Preference transformation

As the preference formula is usually expressed by conjunctive normal formula, an intractable problem is *how to ensure that the preference $p_{i,j}$ will correctly join the multiplication with the corresponding weight $h_{i,j}$, only when $w_{i,j}$ is a requested keyword over the ith keyword field in the query*.

We give a simple example to show this problem. Suppose a query $Q_1 = (E_1 = w_{1,1} \vee w_{1,2}) \wedge (E_2 = w_{2,1} \vee w_{2,2})$ and its preference formula $P_1 = (p_{1,1} \vee p_{1,2}) \wedge (p_{2,1} \vee p_{2,2})$. For a file with keywords $W_1 = (E_1 = w_{1,1}, E_2 = w_{2,2})$ and weights $H_1 = (h_{1,1}, h_{2,2})$, then the sub-relevance on the first field should be calculated as $p_{1,1}h_{1,1}$ rather than $p_{1,2}h_{1,1}$. Finally, the relevance score should be $p_{1,1}h_{1,1} + p_{2,2}h_{2,2}$.

To this end, we utilize the Lagrange polynomial. When given a query $Q := (E_1 = w_{1,1} \vee \cdots \vee w_{1,d_1}) \wedge \cdots \wedge (E_u = w_{u,1} \vee \cdots \vee w_{u,d_u})$ and its corresponding preference formula $P := (p_{1,1} \vee \cdots \vee p_{1,d_1}) \wedge \cdots \wedge (p_{u,1} \vee \cdots \vee p_{u,d_u})$, the owner will employ Lagrange coefficients to construct a polynomial $\varphi_i(x_i)$ $(1 \leqslant i \leqslant u)$ for the requested keywords (i.e., $w_{i,1} \vee \cdots \vee w_{i,d_i}$) over the ith keyword field and then obtain the formula $\varphi(x_1, \ldots, x_u)$.

$$\varphi(x_1, \ldots, x_u) = \sum_{i=1}^{u} \varphi_i(x_i), \tag{3}$$

where

$$\varphi_i(x_i) := \sum_{j=1}^{d_i} \frac{\prod_{k=1}^{j-1}(x_i - w_{i,k}) \prod_{k=j+1}^{d_i}(x_i - w_{i,k})}{\prod_{k=1}^{j-1}(w_{i,j} - w_{i,k}) \prod_{k=j+1}^{d_i}(w_{i,j} - w_{i,k})} p_{i,j}.$$

$\varphi_i(x_i)$[1)] satisfies the following condition: if $w_{i,j}$ is a requested keyword over the ith keyword field in the query Q, then $\varphi_i(w_{i,j}) = p_{i,j}$.

Example We also take the query $Q_1 = (E_1 = w_{1,1} \vee w_{1,2}) \wedge (E_2 = w_{2,1} \vee w_{2,2})$ and its preference formula $P_1 = (p_{1,1} \vee p_{1,2}) \wedge (p_{2,1} \vee p_{2,2})$ as an example. When receiving this query, the owner then converts it into a Lagrange polynomial $\varphi(x_1, x_2) = \varphi_1(x_1) + \varphi_2(x_2)$, where

$$\varphi_1(x_1) = \frac{x_1 - w_{1,2}}{w_{1,1} - w_{1,2}} \cdot p_{1,1} + \frac{x_1 - w_{1,1}}{w_{1,2} - w_{1,1}} \cdot p_{1,2},$$

and

$$\varphi_2(x_2) = \frac{x_2 - w_{2,2}}{w_{2,1} - w_{2,2}} \cdot p_{2,1} + \frac{x_2 - w_{2,1}}{w_{2,2} - w_{2,1}} \cdot p_{2,2}.$$

After the transformation, if the keywords of an index is $W_1 = (E_1 = w_{1,1}, E_2 = w_{2,2})$, then we have $\varphi(w_{1,1}, w_{2,2}) = \varphi_1(w_{1,1}) + \varphi_2(w_{2,2}) = p_{1,1} + p_{2,2}$. We can observe that the introduction of Lagrange polynomial can ensure that only the requested keyword in the query can *extract the corresponding preference into next relevance score calculations*.

To calculate relevance scores by utilizing secure inner-product calculation, the owner needs to express the preference formula and keyword weights in the vector form through polynomial convention. We call the converted vectors the *preference vector* and *keyword weight vector*. Specifically, the owner converts the polynomial in Eq. (3) by extracting the coefficient of x_i^j from $\varphi_i(x_i)$. Then, the preference vector will be

$$\vec{P} := (b_{1,n_1}, \ldots, b_{1,0}, \ldots, b_{u,n_u}, \ldots, b_{u,0})^{\mathrm{T}}, \tag{4}$$

where $b_{i,j}$ is the coefficient of x_i^j in $\varphi_i(x_i)$ and n_i is the number of keywords over the ith keyword field in the dataset. Note that $b_{i,j} := 0$ for $d_i \leqslant j \leqslant n_i$, where d_i is the number of requested keywords over the ith keyword field in Q.

Suppose the keywords of F_s are $W_s = (E_1 = w_{1,s_1}, \ldots, E_u = w_{u,s_u})$, the keyword weight vector of F_s can be denoted as

$$\vec{W}_s := (t_{1,n_1}, \ldots, t_{1,0}, \ldots, t_{u,n_u}, \ldots, t_{u,0})^{\mathrm{T}}, \tag{5}$$

where $t_{i,j} := h_{i,s_i} \cdot w_{i,s_i}^j$ and h_{i,s_i} is the weight of w_{i,s_i}, so that the real relevance score of F_s to $T_{Q,P}$ is

$$\mathrm{R}(T_{Q,P}, F_s) := \vec{W}_s^{\mathrm{T}} \vec{P} := \sum_{i=1}^{u} h_{i,s_i} \cdot \varphi_i(w_{i,s_i}). \tag{6}$$

The output will be $\sum_{i=1}^{u} h_{i,s_i} \cdot p_{i,s_i}$, which is consistent with the relevance score calculation in Eq. (2) if F_s matches the search query (i.e., w_{i,s_i} is a requested keyword over the ith keyword field for $1 \leqslant i \leqslant u$).

Given the relevance privacy, the real relevance scores should be concealed against the cloud server. To this end, we then introduce random values both in the preference vector and in the weight vector, such that the real relevance score will be concealed. Specifically, the data owner first introduces random values α, β, and ε_s (note that these values will not be disclosed to users), and expands $\vec{P}$ and $\vec{W}_s$, such that

$$\hat{P} := (\alpha\vec{P}, \beta), \quad \hat{W}_s := (\vec{W}_s, \varepsilon_s). \tag{7}$$

After the vector expansion, the disturbed relevance score

[1)] In $\varphi_i(x_i)$, a keyword $w_{i,j}$ can be expressed by a real number transformed by using a hash function, which maps strings to real numbers will be

$$\begin{aligned}\mathrm{DR}(T_{Q,P}, F_s) &:= \hat{W}_s^{\mathrm{T}}\hat{P} \\ &:= \alpha \cdot \vec{W}_s^{\mathrm{T}}\vec{P} + \beta\varepsilon_s \\ &:= \alpha \cdot \mathrm{R}(T_{Q,P}, F_s) + \beta\varepsilon_s. \qquad (8)\end{aligned}$$

Here, we call $\mathrm{R}(T_{Q,P}, F_s)$ in Eq. (6) and $\mathrm{DR}(T_{Q,P}, F_s)$ in Eq. (8) the *real relevance score* and the *disturbed relevance score*, respectively. The random value $\beta\varepsilon_s$ is used to blind $\alpha\vec{W}_s^{\mathrm{T}}\vec{P}$, otherwise α can be acquired simply through greatest common divisor computation if the server obtains enough disturbed relevance scores, and then the real relevance score will be leaked.

However, the introduction of $\beta\varepsilon_s$ may degrade the rank precision, as precision and privacy are two important metrics that are in opposition to each other. Larger $\beta\varepsilon_s$ will enhance relevance privacy but incur a lower precision, whereas smaller $\beta\varepsilon_s$ will increase the rank precision but cause a higher probability of privacy leakage. Therefore, these two metrics should be carefully balanced depending on the applications and the user's concerns.

3.1.3 Multi-field search query support

To support the multi-field search query over encrypted data, the straightforward way of introducing an existing SE scheme such as HPE [23] or PE [10] will cause considerable storage cost to keep this extra searchable index. In fact, the weight vector can be reused to support the search operation by just taking the following steps, thus saving a considerable amount of storage space.

For the search query $Q := (E_1 = w_{1,1} \vee \cdots \vee w_{1,d_1}) \wedge \cdots \wedge (E_u = w_{u,1} \vee \cdots \vee w_{u,d_u})$, the owner first chooses a set of random non-integer values $\{r_i\}_{i=1}^u$ that is used to conceal the distribution when performing a search as described in the following, and will not be shared with users. After that, the owner then transforms the query into the polynomial form as follows:

$$r_1 \sum_{i=1}^{d_1}(x_1 - w_{1,i}) + \cdots + r_u \sum_{i=1}^{d_u}(x_u - w_{u,i}). \qquad (9)$$

A vector $(a_{1,d_1}, \ldots, a_{1,0}, \ldots, a_{u,d_u}, \ldots, a_{u,0})$ can be derived from the polynomial, where $a_{i,j}$ is the coefficient of x_i^j and $a_{i,0} = r_i(-1)^{d_i}\prod_{j=1}^{d_i} w_{i,j}$. Finally, the query vector can be unified as

$$\hat{Q} := (a_{1,n_1}, \ldots, a_{1,0}, \ldots, a_{u,n_u}, \ldots, a_{u,0}, 0)^{\mathrm{T}}, \qquad (10)$$

where n_i is the number of keywords on the ith keyword field and $n_i \geqslant d_i$. It is easy to observe that $a_{i,j} := 0$ for $(d_i + 1) \leqslant j \leqslant n_i$.

To perform the matching test for the file F_s labeling with keywords $W_s = (E_1 = w_{1,s_1}, \ldots, E_u = w_{u,s_u})$, the cloud server will calculate

$$\begin{aligned}\hat{W}_s^{\mathrm{T}}\hat{Q} := h_{1,s_1}r_1 &\sum_{i=1}^{d_u}(w_{1,s_1} - w_{1,i}) + \cdots \\ &+ h_{u,s_u}r_u \sum_{i=1}^{d_u}(w_{u,s_u} - w_{u,i}). \qquad (11)\end{aligned}$$

The outputs will equal zero if the keywords of F_s really match the search query Q. We can observe that the introduction of random values $\{r_i\}_{i=1}^u$ can mess the distribution of the query vector. Suppose there are two search queries named Q and Q', and the corresponding query vectors are $\hat{Q}$ and $\hat{Q}'$, $\hat{W}_s^{\mathrm{T}}\hat{Q}$ and $\hat{W}_s^{\mathrm{T}}\hat{Q}'$ will almost certainly be different as long as F_s is excluded both by Q and Q'.

3.2 Design of PSED

To enforce the confidentiality of trapdoors and indices during the search, PSED makes use of secure inner-product calculation [11]. As a summary of the designs above, the detailed procedures of PSED are shown in the following algorithm, which includes four procedures.

• Setup The owner initiates the secret keys, including a binary vector $\vec{S}$ of length $(n+u+1)$, and two invertible matrices M_1 and M_2 of size $(n+u+1)\times(n+u+1)$, where $n = \sum_{i=1}^u n_i$, and u is the number of keyword fields. Here, M_1 and M_2 are used for query encryption and preference encryption.

• BuildIndex The owner first generates the keyword weight vector $\hat{W}_s$ and divides it into two vectors (step 1.(i)). This division is used for encryption as referred in [11]. These two vectors are finally encrypted by the matrices M_1 and M_2 (step 1.(iii)).

• GenTrapdoor When receiving a query Q and its preference P, the owner first converts Q and P into the vectors $\hat{Q}$ and $\hat{P}$, respectively, as in Eqs. (10) and (7) (step 1). Finally, it randomly splits the vectors (step 2), and encrypts them with the inverse of secret matrices M_1 and M_2 (steps 3 and 4). The split and encryption also follow the method in [11].

• SearchIndex When receiving $T_{Q,P}$, the cloud server goes through every index. It first computes $\widetilde{W}_{s,1}^{\mathrm{T}} \cdot T_{Q1} + \widetilde{W}_{s,2}^{\mathrm{T}} \cdot T_{Q2}$, the value of which actually equals $\hat{W}_s^{\mathrm{T}}\hat{Q}$ according to the property of secure inner-product calculation [11]. If the result is zero, then the file matches the query. The disturbed relevance score will then be calculated by running $\widetilde{W}_{s,1}^{\mathrm{T}} \cdot T_{P1} + \widetilde{W}_{s,2}^{\mathrm{T}} \cdot T_{P2}$, which equals $\hat{W}_s^{\mathrm{T}}\hat{P}$. Finally, the server returns the ranked results.

Algorithm Design of PSED

```
Setup($n, u, m$): generate $SK := \{M_1, M_2, \vec{S}\}$;

BuildIndex($SK$,$C$):
1. for $F_s \in C$ do
    i) generate $\hat{W}_s$, and create $\hat{W}_{s,1}$ and $\hat{W}_{s,2}$;
    ii) for $i = 1$ to $(n + u + 1)$ do
        if $\vec{S}[i] = 1$ then
            generate $\hat{W}_{s,1}[i]$ and $\hat{W}_{s,2}[i]$, so that
            $\hat{W}_{s,1}[i] + \hat{W}_{s,2}[i] = \hat{W}_s[i]$;
        else if $\vec{S}[i] = 0$ then
            $\hat{W}_{s,1}[i] := \hat{W}_{s,2}[i] := \hat{W}_s[i]$;
    iii) run $\widetilde{W}_{s,1} := M_1^{\mathrm{T}}\hat{W}_{s,1}$, $\widetilde{W}_{s,2} := M_2^{\mathrm{T}}\hat{W}_{s,2}$;
    iv) output $I_s := \{\widetilde{W}_{s,1}, \widetilde{W}_{s,2}\}$;

GenTrapdoor($Q$, $P$, $SK$):
1. convert $P$ and $Q$ into $\hat{P}$ and $\hat{Q}$;
2. for i=1 to $(n + u + 1)$ do
    if $\vec{S}[i] = 0$ then
        generate $\hat{Q}_1[i]$, $\hat{Q}_2[i]$, $\hat{P}_1[i]$, and $\hat{P}_2[i]$ so that
        $\hat{Q}_1[i] + \hat{Q}_2[i] = \hat{Q}[i]$ and $\hat{P}_1[i] + \hat{P}_2[i] = \hat{P}[i]$;
    else if $\vec{S}[i] = 1$ then
        $\hat{Q}_1[i] := \hat{Q}_2[i] := \hat{Q}[i]$; $\hat{P}_1[i] := \hat{P}_2[i] := \hat{P}[i]$;
3. compute $T_{Q1} = M_1^{-1}\hat{Q}_1$, $T_{Q2} = M_2^{-1}\hat{Q}_2$;
4. compute $T_{P1} = M_1^{-1}\hat{P}_1$, $T_{P2} = M_2^{-1}\hat{P}_2$;
5. return $T_{Q,P} := \{(T_{Q1}, T_{Q2}), (T_{P1}, T_{P2})\}$.

SearchIndex($C$,$T_{Q,P}$)
1. for $F_s \in C$ do
    compute $\widetilde{W}_{s,1}^{\mathrm{T}} \cdot T_{Q1} + \widetilde{W}_{s,2}^{\mathrm{T}} \cdot T_{Q2}$;
    if the calculated result is an integer then
        calculate $\widetilde{W}_{s,1}^{\mathrm{T}} \cdot T_{P1} + \widetilde{W}_{s,2}^{\mathrm{T}} \cdot T_{P2}$;
2. rank the satisfied files and return the top-$k$ matching files;
```

3.3 Analysis of PSED

3.3.1 Efficiency analysis

In the step of BuildIndex, two multiplications between a $(n+u+1)\times(n+u+1)$ matrix and a $(n+u+1)$-dimensional vector are required for each file. When generating a trapdoor, it needs four multiplications between a $(n+u+1)\times(n+u+1)$ matrix and a $(n + u + 1)$-dimensional vector. In the step of SearchIndex, the cloud server will only calculate the inner-product of two $(n + u + 1)$-dimensional vectors for each mismatching file. For every matching file, an extra inner-product calculation between two $(n + u + 1)$-dimensional vectors is 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) and a vector with the length of $(n + u + 1)$ (i.e., $\vec{S}$). The user should store the trapdoor that is constituted by four $(n+u+1)$-dimensional vectors, whereas the cloud server keeps the encrypted collection and the encrypted indices.

3.3.2 Index privacy and trapdoor privacy

As mentioned above, to enable the secure inner-product computation, the trapdoor and index in PSED are encrypted by using a random binary vector $\vec{S}$ and two invertible matrices M_1 and M_2. This encryption scheme is proposed in [11] and its security against the lever-3 attack [11] is proved. In the lever-3 attack, an attacker who is unaware of the random binary vector $\vec{S}$, may possess t plaintext vectors $\{\hat{W}_s\}_{s=1}^t$ and the corresponding encrypted vectors $\{\widetilde{W}_{s,1}, \widetilde{W}_{s,2}\}_{s=1}^t$, and try to recover other encrypted vectors. Our detailed proof is presented in the Appendix.

Meanwhile, because of the randomized splitting and the introduction of some random values (e.g., $\{r_i\}_{1\leqslant i\leqslant u}$, α, and β), the produced trapdoors will be various even to the same query. This non-deterministic property will also increase the difficulty for the cloud server in mining the relationship between two trapdoors by comparing them directly. Though the cloud server might compare the corresponding matching files and ranked results to judge whether the targeted queries have internal correlation, this attack will be useless if some puppet files are introduced to conceal the search outputs.

3.3.3 Relevance privacy

With the protection of random values, the disturbed relevance scores between F_s and P will be $\alpha \cdot \mathrm{R}(T_{Q,P}, F_s) + \beta\varepsilon_s$, which blinds the real relevance score $\mathrm{R}(T_{Q,P}, F_s)$ against the cloud server. Even the cloud server may try to collect t real relevance scores $\{\mathrm{R}(T_{Q,P}, F_s)\}_{s=1}^t$ with the corresponding disturbed relevance scores, and construct t linear equations as follows:

$$\begin{cases} \mathrm{DR}(T_{Q,P}, F_1) = \alpha \cdot \mathrm{R}(T_{Q,P}, F_1) + \beta\varepsilon_1, \\ \qquad\vdots \\ \mathrm{DR}(T_{Q,P}, F_t) = \alpha \cdot \mathrm{R}(T_{Q,P}, F_t) + \beta\varepsilon_t. \end{cases}$$

This is an attempt to obtain the randomly chosen values (e.g., α, β, $\{\varepsilon_i\}_{i=1}^t$) and recover the real relevance scores of other files. However, it will be useless because there are $(t + 2)$ variables in these t equations.

For the unsatisfied files to a query, $\mathrm{R}(T_{Q,P}, F_s)$ will also output incorrect relevance scores, because the weight of the excluded keyword will participate in the calculation and jumble the calculated relevance scores, making it more difficult for the server to learn the relevance of the unsatisfied files to a query.

3.3.4 Scalability of PSED

PSED is also compatible with the scalable and dynamic feature of cloud computing. When some new keywords are introduced, PSED can efficiently cope with this change without fully re-encrypting the whole index. Without loss of generality, suppose m_1 new keywords are added to the first keyword field, then the updated keyword weight vector $\hat{W}'_s := (\Delta\vec{W}_1, \hat{W}_s)$, where $\Delta\vec{W}_1$ is the expanded vector produced by the new keywords and its length is m_1. PSED can accordingly extend the secret binary vector $\vec{S}$ to $\vec{S}'$ and expand the matrices M_1 and M_2 to M'_1 and M'_2, respectively, where $\vec{S}' := (\Delta\vec{S}, \vec{S})$, $M'_1 := \begin{pmatrix} \Delta M_1 & 0 \\ 0 & M_1 \end{pmatrix}$ and $M'_2 := \begin{pmatrix} \Delta M_2 & 0 \\ 0 & M_2 \end{pmatrix}$ (both ΔM_1 and ΔM_2 are $m_1 \times m_1$ invertible matrices). Therefore, the encryption of $\hat{W}'_s$ can be treated as the combination of two independent encryptions of $\Delta\vec{W}_1$ (encrypted by $\Delta\vec{S}$, ΔM_1, and ΔM_2) and $\hat{W}_s$ (encrypted by $\vec{S}$, M_1, and M_2). As the encrypted index before expansion is already the encryption of $\hat{W}_s$ under the effect of $\vec{S}$, M_1, and M_2, the data owner only needs to compute the encryption of $\Delta\vec{W}_1$ and combine it with the old version of the index without fully re-encrypting $\hat{W}'_s$. This method is more efficient and scalable, which conforms to the properties of cloud computing.

4 Performance evaluation

In this section, we choose the Enron email dataset, which is a real-world email dataset consisting of several keyword fields and email contents. The Enron email dataset includes 517,431 instances, whose number is much larger than the average number of shared files per user reported in two typical cloud storage systems (i.e., 55 per Box user[2)] and 684 in [24]). Therefore, we argue that the use of Enron email dataset is reasonable.

We select $u = 4$ representative fields and a number of instances to evaluate the performance of PSED. PSED is fully run on a modern server (this setting can easily extend to multiple servers), which is equipped with a 2.10 GHz Intel Core 2 Duo CPU and 4 GB RAM. The operating system is Ubuntu (version: 11.04). We compare PSED with MRSE_II [9]. MRSE realizes multi-keyword ranked search over encrypted data and outputs similar results. In the evaluation, we use n_Q to denote the number of keywords in the query. Here, n and $|C|$ represent the number of keywords and the number of files in the collection, respectively.

[2)] Box Free Cloud Storage, Secure Content Online File Sharing

4.1 Index building

To explore the per-index encryption time, we focus on the overhead when the total number of keywords n in the collection and the collection size $|C|$ change. From Fig. 3(a), the per-index encryption time is constant when n is fixed. Therefore, the generation time for a dataset will be linear with the number of files. Meanwhile, when n varies, the per-index encryption time in PSED scales as $O(n^2)$, because it is usually required to perform $O(n^2)$ multiplications and $O(n^2)$ addition operations in the index generation when u is fixed. This comparison also indicates that the per-index encryption time in PSED is nearly the same with that in MRSE.

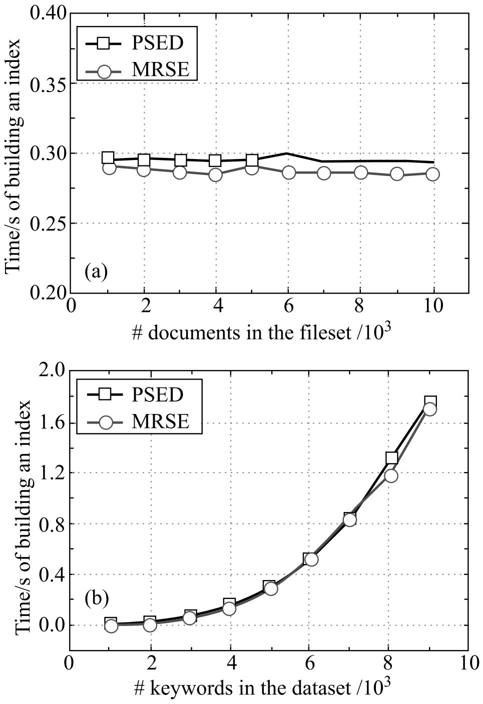


Fig. 3 Per-index encryption time. (a) The per-index encryption time under different sizes of dataset, when n=5,000 and n_Q=100, where n_Q is the number of keywords in the query; (b) The per-index encryption time under different number of keywords in the dataset, when n_Q=100 and $|C|$=5,000

As it is usually a one-time cost to build indices for the dataset, we argue that the efficiency is quite reasonable. Table 2 also lists the per-index storage overhead in PSED and MRSE. The results indicate that PSED achieves nearly the same storage efficiency on index building compared with MRSE.

4.2 Trapdoor generation

For GenTrapdoor, Fig. 4(a) shows that the number of keywords in the query n_Q will not affect the performance of trapdoor generation very much. Figure 4(b) indicates that the per-trapdoor generation time will be affected by the total number of keywords in the collection, because it is required to employ secret matrices to encrypt the query vector and the preference vector, incurring $O(n^2)$ multiplications and $O(n^2)$ addition calculations when u is fixed. Meanwhile, the performance of PSED is a bit slower than that of MRSE, this is because some additional encryption operations should be carried out in PSED, i.e., the encryption of Q for the multi-field query.

Table 2 Size of the index and trapdoor in MRSE and PSED

n	MRSE-Index(/Trapdoor)/kB	PSED-Index/kB	PSED-Trapdoor/kB
1,000	7.8	7.9	15.8
5,000	39.1	39.1	78.2
8,000	62.5	62.5	125.0

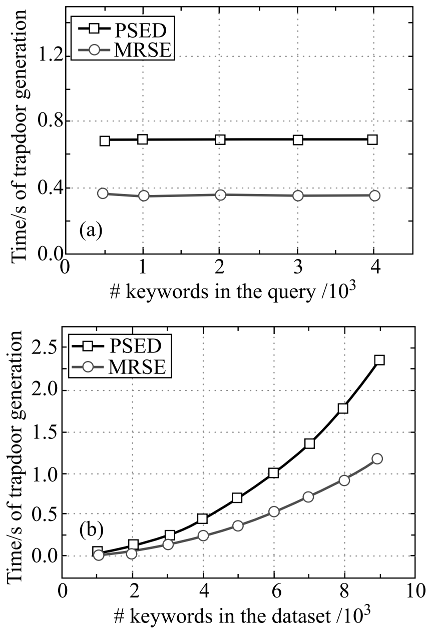


Fig. 4 Trapdoor generation. (a) Trapdoor generation for the different numbers of keywords in the query, when n=5,000; (b) trapdoor generation for the varying numbers of keywords in the whole dataset, when n_Q=100

In addition, we also compare the storage overhead to keep a trapdoor in PSED with that in MRSE in Table 2. It seems that a trapdoor of PSED takes up nearly twice the amount of storage space as that of MRSE, because a trapdoor in PSED should include the extra preference information apart from the user's query.

4.3 Search

In PSED, the cloud server first picks out the matching files to a query and then calculates their disturbed relevance scores. Thus, the actual time cost in the query stage can be decomposed into the time in matching test and the time in relevance score calculation. In this test, we use *hit rate* to denote the rate of matching files to the query and consider the query time under different hit rates. We then carry out three tests to measure the query time under different numbers of keywords in the query, and different numbers of keywords in the collection. The results are illustrated in Fig. 5.

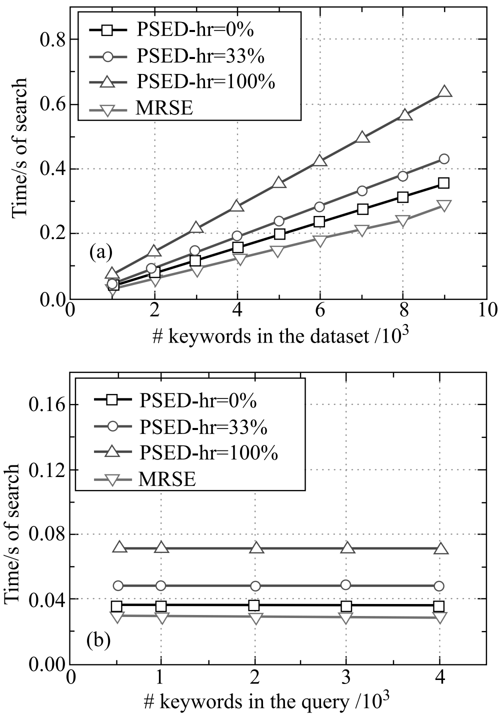


Fig. 5 Search time. (a) The search time under different numbers of keywords in the dataset, when n_Q=100 and $|C|$=5,000; (b) the search time under different numbers of keywords in the query, when $n = 5,000$ and $|C|$=1,000

Figure 5(a) shows the search time is linear to the number of keywords in the collection, because it is required to perform $O(n)$ multiplications and $O(n)$ addition operations in the matching test and the relevance score calculation, when u is fixed. Figure 5(b) indicates the number of keywords in the query will not affect the performance of query search.

Meanwhile, it requires more time if the hit rate becomes higher, as more relevance score calculations will be invoked when the number of matching files increases. We can also observe that PSED can achieve roughly the same search efficiency of MRSE when the hit rate is zero.

4.4 Comparison of match degree

In this test, we compare PSED with MRSE on the metric of match degree to measure the ratio of the matching files to the query in the returned files. We use $\mathcal{M}_i$ to denote the collection of matching files for the ith search request and use q_i to denote the match degree.

As we mentioned before, MRSE performs search to a query according to the principle of *coordinate matching* (i.e., the number of common keywords appearing in both the query and the index) to quantitatively capture their relevance and return the files ranked in the top-k. This approach will return similar search result in the ith search request when the number of matching files $|\mathcal{M}_i|$ is less than the pre-defined value k. We can model q_i in MRSE by the following equation:

$$q_i := \frac{|\mathcal{M}_i|}{k}, |\mathcal{M}_i| \leqslant k. \tag{12}$$

In contrast, the cloud server in PSED respects the rule of selecting the matching files first and then returning them according to their relevance scores. Suppose the number of matching files is $|F_{Q,P}|$, once the amount of matching files is less than k, i.e., $|F_{Q,P}| < k$, the cloud server will only return the matching files with the number of $|F_{Q,P}|$. Therefore, the number of returned files will be $Min\{k, |F_{Q,P}|\}$. Thus, $q_i :\equiv 1$ always holds in PSED.

Obviously, q_i partly reflects the useless bandwidth wastage during the ith search, and a larger q_i usually indicates less wasted bandwidth. Moreover, if all the returned files match with the query, then q_i will reach the maximum value (i.e., 1). To evaluate q_i, we generate the encrypted indices for 200 selected files and carry out two tests in this comparison. Note that we denote the fields that have requested keywords in the query as *valid keyword fields*. First, we set the pre-specified value k as 10, and change the number of valid fields and the number of files in the dataset. For each valid keyword field in the query, we choose an interested keyword over it. Second, we set the number of files in the dataset as 200 and calculate q_i when k and the number of valid fields in the query change. We repeat these two tests many times and record the averaged results in Fig. 6.

Two observations can be derived from Fig. 6(a). First, when the scale of data files and the value k are fixed, q_i will drop if the number of valid keywords in the query increases. This is because $|\mathcal{M}_i|$ in MRSE will be smaller when the requirements in the query are stricter. Second, when the number of valid fields in the query and k are fixed, q_i will increase when the number of data files becomes larger. This is because $|\mathcal{M}_i|$ will probabilistically increase when the dataset scales. As indicated in Fig. 6(b), q_i will drop when the pre-specified value k becomes larger. In contrast, because PSED only returns the matching files to the query, it can always achieve the match degree of 1.

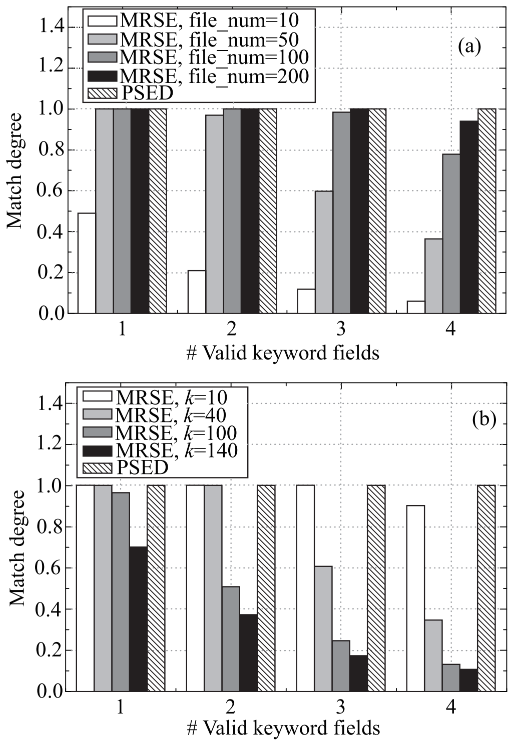


Fig. 6 Comparison of match degree. (a) The match degree under different scales of dataset, when k=10; (b) the match degree under different selections of k, when the number of files is 200

4.5 Relevance privacy and precision

In this test, we mainly focus on the relevance privacy and retrieval precision under different strengths of randomization. We select 10,000 files, and calculate the weight of the keyword over each keyword field. As the introduced randomization may cause rank perturbation for the matching files, we evaluate the precision on two metrics, i.e., search precision and rank deviation.

Suppose the search output (i.e., the returned files) in PSED produced by the real relevance scores should be $\mathcal{F}$ and the output produced by the disturbed relevance scores is $\mathcal{F}'$, it is obvious that $|\mathcal{F}| = |\mathcal{F}'| \leqslant k$ establishes where $|\mathcal{F}|$ and $|\mathcal{F}'|$ denote the size of $\mathcal{F}$ and the size of $\mathcal{F}'$, respectively. The search precision is to measure the miss rate for the files $\{F_i | F_i \in \mathcal{F}, F_i \notin \mathcal{F}'\}$ and the rank deviation is to measure the rank perturbation for the files $\{F_i | F_i \in \mathcal{F} \cap \mathcal{F}'\}$. Based on the above introduction, the search precision σ can be calculated by

$$\sigma := \frac{|\mathcal{F} \cap \mathcal{F}'|}{|\mathcal{F}|}. \tag{13}$$

Suppose the rank orders of file F_i in $\mathcal{F}$ and $\mathcal{F}'$ are o_i and o'_i, respectively, then the rank deviation δ can be evaluated by the following equation:

$$\delta := \sum_{F_i \in \mathcal{F} \cap \mathcal{F}'} |o_i - o'_i|. \tag{14}$$

As stated previously, the randomization introduced to conceal the real relevance score is $\beta\varepsilon_s$ and α. In this test, we evaluate these two metrics by varying the randomization of $\beta\varepsilon_s$ from $[0, \frac{1}{10}\alpha p_{\max}]$ to $[0, \alpha p_{\max}]$, where $p_{\max}$ is the maximum preference selected in the evaluation. The results are shown in Fig. 7. Moreover, we also partition the minimal range that covers both the real relevance scores and the disturbed relevance scores into 50 intervals when the non-linear randomization $\beta\varepsilon_s$ for file F_s is uniformly selected from the range $[0, \frac{1}{2}\alpha p_{\max}]$.

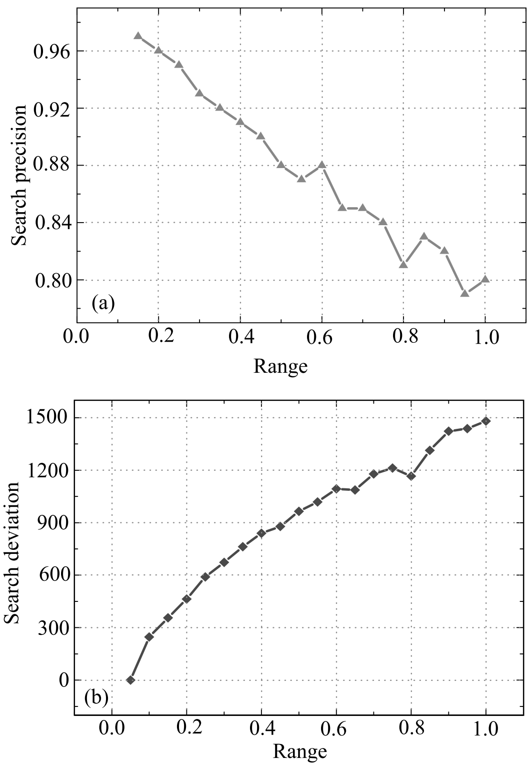


Fig. 7 Search precision and search deviation. (a) The search precision under different selected ranges, when k=100; (b) the search deviation under different selected ranges, when k=100. The range $\xi \in (0, 1.0)$ means the non-linear randomization is uniformly chosen from the range $[0, \xi\alpha p_{\max}]$

Figure 7(a) indicates that the search precision will decrease when the introduced random values become larger. This is because the increase of randomization strength will increase the probability of causing larger rank perturbation. Thus, it requires users to make a tradeoff between the search precision and the relevance privacy. Based on the same principle, the search deviation will increase with the range of randomization extension as shown in Fig. 7(b).

4.6 Summary

We further summarize the properties of PSED in Table 3, which indicates that PSED focuses on PS over multiple fields, searches the accurate matching files first and then ranks them based on the relevance.

Table 3 Summary of PSED

Metrics	PSED
Multi-field query support	Yes
Preferred keyword search	Yes
Match degree	Accurate match

We also compare PSED with other representative SE schemes in Table 4. We can see that PSED only introduces lightweight computation in search operations and provides multi-field search query with preference.

Table 4 Summary of PSED

Schemes	Index size	Search time
GSW04 [25]	$O(v)$	$O(e)G + O(e)P$
BCOP04 [16]	$O(v)$	$O(1)P + O(1)H$
SBCSP07 [26]	$O(D \cdot \log T)$	$O(\log T^D)P + O(\log T^D)G$
CWLRL [9]	$O(n + y)$	$O(n + y)$
KSW08 [23]	$O(n)$	$O(n)P$
PSED	$O(n + u)$	$O(n + u)$

Note: n: the total number of keywords in the dataset
u: the number of keyword fields
v: the number of keywords in the index
e: the number of requested keywords in the query
G: the operation in groups
H: the hash operation
D: the number of dimensions
P: the pairing operation
y: the number of dummy elements inserted

5 Related work

5.1 Preferred search

Stefanidis et al. [27] proposed PerK to implement personalizing keyword search in relational databases that took the user's preference into account. Leubner and Kiessling [28] considered two preference constructors (i.e., Pareto accumulation and prioritization) that had partial order semantics under the scenario of full-text search. Koutrika and Ioannidis [8] used a profile to store the user's preference. When a user issued an ordinary query, the system would re-generate a new query that covered the user's preference by taking the original query and user's profile as input. Chomicki [29] presented a framework to formulate complex preference queries by utilizing a simple winnow operator. Kiessling [30] proposed strictly partial order semantics for preferences and constructed a complex preference by utilizing preference constructors. Georgiadis et al. [31] defined the preorders over attributes and proposed query-rewriting algorithms to support a progressive evaluation of block sequences. However, most existing schemes on PS were mainly investigated over plaintext and remain inapplicable to encrypted data.

5.2 Searchable encryption

Song et al. [32] proposed the first practical scheme in the literature of SE, where the search time increased with the file size. Goh [33] studied secure indexing over encrypted data by employing the Bloom Filter [34], which however would introduce the possibility of false positive. Boneh et al. [16] proposed the first SE scheme based on public keys. Waters et al. [35] realized a searchable audit log in two ways, namely, a symmetric-encryption- based scheme and identity-based encryption (IBE)-based scheme. To achieve good search experience, Wang et al. [18,36] investigated secure ranked keyword search, which needed the keyword to be preprocessed by the data owner locally and only supported equality query. Wang et al. [37] investigated the problem of similarity search over encrypted data and constructed the trie-traversing search index based on edit distance. This method also supported the fuzzy search over encrypted data studied by Li et al. [38]. However, all the above works only supported single-keyword search.

To enable multi-keyword search, Golle et al. [25] developed conjunctive keyword search over encrypted data. Shi et al. [26] realized multi-dimensional range query over encrypted data. Several attempts [9, 39] have also been made on the multi-keyword ranked search over encrypted data and their schemes output similar files to the query. Cao et al. [15] followed the principle of filtering-and-verification and fulfilled the privacy-preserving query over encrypted graph. In addition, some representative works [10, 23] in the area of predicate encryption could achieve searching over encrypted data by attaching the attribute vector $\vec{v}'$ to the ciphertext and representing a search query by the predicate vector $f_{\vec{v}}$, therefore a match happened only when $(f_{\vec{v}}, \vec{v}') = 0$. Shen et al. [40] designed an interesting index based on both access policy and keywords and, thus, the cloud server can simultaneously perform access control and search over encrypted data. Shen et al. [41] further proposed to assign each keyword with a preference for enabling preferred keyword search over encrypted data in cloud computing. Li et al. [42] also tried to realize access control and keyword search over encrypted data by employing both attribute-based encryption [43] and hybrid clouds.

In addition, Fu et al. [44] proposed an efficient multi-keyword fuzzy ranked search scheme with improved accuracy. Xia et al. [45] designed a secure multi-keyword ranked search scheme that also supports dynamic update operations. Fu et al. [46] found that previous keyword-based search schemes ignore the semantic information. They then developed a semantic search scheme based on the concept hierarchy and the semantic relationship between concepts in the encrypted datasets. Fu et al. [47] designed a SE scheme that supported both multi-keyword ranked search and parallel search. However, most previous work only paid limited attention to the user's preferences.

6 Conclusions

We investigated the problem of preferred search over encrypted cloud data. We first established a set of designed goals and used the TF×IDF model for keyword weight measurement. We expressed the user's query and preference and keywords and their weights in vector form. The secure inner-product computation was then employed to perform search and measure the relevance between files and the user's preference. Thorough analysis concerning privacy and efficiency was presented, and the intensive evaluation on a modern server demonstrated its suitability.

Acknowledgements We would like to thank the anonymous reviewers for their comments and suggestions. This work was supported by the National Key R&D Program (2016YFA0602103), the National Natural Science Foundation of China (Grant Nos. 61232003, 61433008, 91530323, 61602120, and 61361120098), and the Beijing Municipal Science and Technology Commission of China (D151100000815003).

Appendixes

Appendix A Attack model

In our attack model, we assume an attacker can access the encrypted data, the encrypted vectors (e.g., query vectors and keyword weight vectors), and the encrypted results. In addition, the attacker can obtain a set of vectors in plaintext and corresponding encrypted vectors, and try to recover other encrypted vectors that he has not yet learned. This is equivalent to the known-plaintext attack.

Theorem 1 PSED is resilient to the known-plaintext attack if the attacker cannot derive the secret keys, i.e., the binary vector $\vec{S}$ and the invertible matrices M_1 and M_2 (see Setup in Algorithm: Design of PSED).

Proof We first prove the index privacy against the known-plaintext attack, and the proof of trapdoor privacy is similar. Suppose the attacker possesses t plaintext vectors $\{\hat{W}_{s,1}, \hat{W}_{s,2}\}_{1\leqslant s\leqslant t}$ after being split by the binary vector $\vec{S}$. In addition, the attacker also obtains their corresponding encrypted vectors $\{\widetilde{W}_{s,1}, \widetilde{W}_{s,2}\}_{1\leqslant s\leqslant t}$. If the attacker does not know the splitting configuration, he has to model $\hat{W}_{s,1}$ and $\hat{W}_{s,2}$ as two random $(n+u+1)$-dimensional vectors. The equations to solve the matrices are $M_1^{\mathrm{T}}\hat{W}_{s,1} = \widetilde{W}_{s,1}$ and $M_2^{\mathrm{T}}\hat{W}_{s,2} = \widetilde{W}_{s,2}$ for $1 \leqslant s \leqslant t$, where M_1 and M_2 are two $(n+u+1)\times(n+u+1)$ unknown matrices (see Setup in Algorithm 1). There are $2(n+u+1)t$ unknowns in $\{\widetilde{W}_{s,1}, \widetilde{W}_{s,2}\}_{1\leqslant s\leqslant t}$, and $2(n+u+1)^2$ unknowns in M_1 and M_2. As there are only $2(n+u+1)t$ equations, which is less than the number of unknowns, the attacker cannot have sufficient information to solve for the matrices. Hence, PSED can resist against the known-plaintext attack. □

Appendix B Leakage function

A search for a query Q leaks a file set I_Q matching the requested keyword in Q. It also leaks the disturbed relevance scores of I_Q to the preference query P (see Eq. (8)), as the files in I_Q will be sorted according to their disturbed relevance scores.

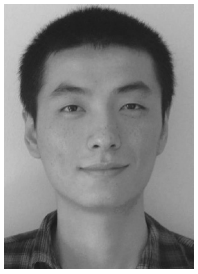
Suppose for a file set I_Q and for a preference query P, the disturbed relevance scores for I_Q and P are denoted by $DR_{I_Q,P}$. For a given pair of the search query Q and the preference query P, we define the leakage function as $\mathsf{leak}_{Q,P} = \{I_Q, DR_{I_Q,P}\}$.

References

1. Armbrust M, Fox A, Griffith R, Joseph A, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, Zaharia M. A view of cloud computing. Communications of the ACM, 2010, 53(4): 50–58
2. Ren Y J, Shen J, Wang J, Han J, Lee S Y. Mutual verifiable provable data auditing in public cloud storage. Journal of Internet Technology, 2015, 16(2): 317–323
3. Ma T H, Zhou J J, Tang M L, Tian Y, Dhelaan A, Rodhaan A, Lee S Y. Social network and tag sources based augmenting collaborative recommender system. IEICE transactions on Information and Systems, 2015, E98–D(4): 902–910
4. Shu J W, Shen Z R, Xue W. Shield: a stackable secure storage system for file sharing in public storage. Journal of Parallel and Distributed Computing, 2014, 74(9): 2872–2883
5. Shu J W, Shen Z R, Xue W, Fu Y X. Secure storage system and key technologies. In: Proceedings of Asia and South Pacific Design Automation Conference. 2013, 376–383
6. Cai K, Hong C, Zhang M, Feng D G, Lv Z Q. A secure conjunctive keywords search over encrypted cloud data against inclusion-relation attack. In: Proceedings of IEEE International Conference on Cloud Computing Technology and Science. 2013, 339–346
7. Guo H, Li Z J, Mu Y, Zhang X Y. Provably secure identity-based authenticated key agreement protocols with malicious private key generators. Information Science, 2011, 181(3): 628–647
8. Koutrika G, Ioannidis Y. Personalized queries under a generalized preference model. In: Proceedings of International Conference on Data Engineering. 2005, 841–852
9. Cao N, Wang C, Li M, Ren K, Lou W. Privacy-preserving multi-keyword ranked search over encrypted cloud data. In: Proceedings of IEEE International Conference on Computer Communications. 2011, 829–837
10. Okamoto T, Takashima W. Hierarchical predicate encryption for inner-products. In: Proceedings of International Conference on the Theory and Application of Cryptology and Information Security. 2009, 241–237
11. Wong W K, Cheung D W, Kao B, Mamoulis N. Secure knn computation on encrypted databases. In: Proceedings of ACM SIGMOD International Conference on Management of Data. 2009, 139–152
12. Yu S C, Wang C, Ren K, Lou W J. Achieving secure, scalable, and fine-grained data access control in cloud computing. In: Proceedings of IEEE International Conference on Computer Communications. 2010, 534–542
13. Li M, Yu S C, Cao N, Lou W J. Authorized private keyword search over encrypted data in cloud computing. In: Proceedings of IEEE International Conference on Distributed Computing Systems. 2011, 383–392
14. Pervez Z, Awan A, Khattak A, Lee S, Huh E. Privacy-aware searching with oblivious term matching for cloud storage. The Journal of Supercomputing, 2013, 63(2): 538–560.
15. Cao N, Yang Z Y, Wang C, Lou W J. Privacy-preserving query over encrypted graph-structured data in cloud computing. In: Proceedings of IEEE International Conference on Distributed Computing Systems. 2011, 393–402
16. Boneh D, Crescenzo G, Ostrovsky R, Persiano G. Public key encryption with keyword search. In: Proceedings of International Conference on the Theory and Applications of Cryptographic Techniques. 2004, 506–522
17. Lu Y. Privacy-preserving logarithmic-time search on encrypted data in cloud. In: Proceedings of the 19th Annual Network & Distributed System Security Symposium. 2012
18. Wang C, Cao N, Li J, Ren K, Lou W J. Secure ranked keyword search over encrypted cloud data. In: Proceedings of IEEE International Conference on Distributed Computing Systems. 2010, 253–262
19. Witten I, Moffat A, Bell T. Managing Gigabytes: Compressing and Indexing Documents and Images. San Mateo, CA: Morgan Kaufmann Publishers, 1999
20. Yan T W, Garcia-Molina H. Sift: a tool for wide-area information dissemination. In: Proceedings of USENIX Annual Technical Conference. 1995, 16–20
21. Liu W Y, Chen Z, Lin F, Zhang H J, Ma W Y. Ubiquitous media agents: a framework for managing personally accumulated multimedia files. Multimedia Systems, 2003, 9(2): 144–156
22. Good N, Schafer J, Konstan J, Borchers A, Sarway B, Herlocker J,

Biedl J. Combining collaborative filtering with personal agents for better recommendations. In: Proceedings of the 16th National Conference on Artificial Intelligence and 11th Conference on Innovative Applications of Artificial Intelligence. 1999, 439–446

23. Katz J, Sahai A, Waters B. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In: Proceedings of International Conference on the Theory and Applications of Cryptographic Techniques. 2008, 146–162
24. Yang X, Liu L, Global I. Principles, Methodologies, and Service-Oriented Approaches for Cloud Computing. Hershey, PA: Business Science Reference, 2013
25. Golle P, Staddon J, Waters B. Secure conjunctive keyword search over encrypted data. In: Proceedings of the International Conference on Applied Cryptography and Network Security. 2004, 31–45
26. Shi E, Bethencourt J, Chan H, Song D, Perrig A. Multi-dimensional range query over encrypted data. In: Proceedings of IEEE Symposium on Security and Privacy. 2007, 350–364
27. Stefanidis K, Drosou M, Pitoura E. PerK: personalized keyword search in relational databases through preferences. In: Proceedings of International Conference on Extending Database Technology. 2010, 585–596
28. Leubner A, Kiessling W. Personalized keyword search with partial-order preferences. In: Proceedings of Brazilian Symposium on Databases. 2002, 181–193
29. Chomicki J. Preference formulas in relational queries. ACM Transaction on Database Systems, 2003, 28(4): 427–466
30. Kiessling W. Foundations of preferences in database systems. In: Proceedings of International Conference on Very Large Data Bases. 2002, 311–322
31. Georgiadis P, Kapantaidakis I, Christophides V, Nguer E, Spyratos N. Efficient rewriting algorithms for preference queries. In: Proceedings of International Conference on Data Engineering. 2008, 1101–1110
32. Song D, Wagner D, Perrig A. Practical techniques for searches on encrypted data. In: Proceedings of IEEE Symposium on Security and Privacy. 2000, 44–55
33. Goh E J. Secure indexes. IACR Cryptology ePrint Archive, 2003
34. Bloom B. Spacetime trade-offs in hash coding with allowable errors. Communications of the ACM, 1970, 13(7): 422–426
35. Waters B, Balfanz D, Durfee G, Smetters D. Building an encrypted and searchable audit log. In: Proceedings of Network and Distributed System Security Symposium. 2004
36. Wang C, Cao N, Ren K, Lou W J. Enabling secure and efficient ranked keyword search over outsourced cloud data. IEEE Transactions on Parallel and Distributed Systems, 2012, 23(8): 1467–1479
37. Wang C, Ren K, Yu S C, Urs K. Achieving usable and privacy-assured similarity search over outsourced cloud data. In: Proceedings of IEEE International Conference on Distributed Computing Systems. 2012, 451–459
38. Li J, Wang Q, Wang C, Cao N, Ren K, Lou W J. Fuzzy keyword search over encrypted data in cloud computing. In: Proceedings of IEEE International Conference on Distributed Computing Systems. 2010, 441–445
39. Sun W H, Wang B, Cao N, Li M, Lou W J, Hou Y, Li H. Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. In: Proceedings of ACM Symposium on Information, Computer and Communications Security. 2013, 71–82
40. Shen Z R, Shu J W, Xue W. Keyword search with access control over encrypted data in cloud computing. In: Proceedings of International Symposium of Quality of Service. 2014, 87–92
41. Shen Z R, Shu J W, Xue W. Preferred keyword search over encrypted data in cloud computing. In: Proceedings of International Symposium of Quality of Service. 2013, 207–212
42. Li J W, Li J, Chen X F, Jia C F, Liu Z L. Efficient keyword search over encrypted data with fine-grained access control in hybrid cloud. In: Proceedings of International Conference on Network and System Security. 2012, 490–502
43. Goyal V, Pandey O, Sahai A, Waters B. Attribute-based encryption for fine-grained access control of encrypted data. In: Proceedings of ACM Conference on Computer and Communications Security. 2006, 89–98
44. Fu Z J, Wu X L, Guan C W, Sun X M, Ren K. Towards efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement. IEEE Transactions on Information Forensics and Security, 2016, 11(12): 2706–2716
45. Xia Z H, Wang X H, Sun X M, Wang Q. A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. IEEE Transactions on Parallel and Distributed Systems, 2015, 27(2): 340–352
46. Fu Z J, Ren K, Shu J G, Sun X M, Huang F X. Enabling personalized search over encrypted outsourced data with efficiency improvement. IEEE Transactions on Parallel and Distributed Systems, 2015, 27(9): 2546–2559
47. Fu Z J, Sun X M, Liu Q, Zhou L, Shu J G. Achieving efficient cloud search services: multi-keyword ranked search over encrypted cloud data supporting parallel computing. IEICE Transactions on Communications, 2015, E98–B(1): 190–200



Zhirong Shen received a bachelor's degree from the University of Electronic Science and Technology of China, China in 2010, and a PhD from Tsinghua University, China in 2016. He is now a postdoctoral fellow at the Chinese University of Hong Kong, China. His current research interests include storage reliability and storage security.

Jiwu Shu received a PhD degree in computer science from Nanjing University, China in 1998, and finished the postdoctoral position research at Tsinghua University, China in 2000. Since then, he has been teaching at Tsinghua University. His current research interests include storage security and reliability, non-volatile memory-based storage systems, and parallel and distributed computing. He is a member of the IEEE.

Wei Xue is an associate professor in Department of Computer Science and Technology and Center of Earth System Science in Tsinghua University, China. His research interests include high-performance computing, uncertainty quantification for climate system model. He is a senior member of the CCF and a member of the IEEE and ACM.