Journal of Computer and System Sciences ••• (••••) •••–•••

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

A query privacy-enhanced and secure search scheme over encrypted data in cloud computing

Hui Yin [a,b], Zheng Qin [a,*], Lu Ou [a], Keqin Li [c]

[a] College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan, 410082, China
[b] Department of Mathematics and Computer Science, Changsha University, Changsha, Hunan, 410022, China
[c] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Article history:
Received 17 December 2015
Received in revised form 4 December 2016
Accepted 16 December 2016
Available online xxxx

Keywords:
Cloud computing
Secure index
Secure search
Privacy protection

ABSTRACT

With the emerging of the cloud computing, secure search over encrypted cloud data has become a hot research spot. Previous schemes achieve weaker query privacy-preserving ability due to the limitations of query trapdoor generation mechanisms. In these schemes, a data owner usually knows fully well the query contents of data users and a data user can also easily analyze query contents of another data user. In some application scenarios, the data user may be unwilling to leak their query privacy to anyone else except himself. We propose a privacy-enhanced search scheme by allowing the data user to generate random query trapdoor every time. We leverage Bloom filter and bilinear pairing operation to construct secure index for each data file, which enables the cloud to perform search without obtaining any useful information. We prove that our scheme is secure and extensive experiments demonstrate the correctness and practicality of the proposed scheme.

© 2017 Elsevier Inc. All rights reserved.

1. Introduction

With the rapid development of cloud computing, more and more organizations and individual users are beginning to outsource their private data to cloud for enjoying IT cost savings, quick deployment, excellent computation performance, and on-demand high quality services. But, the cloud, as a semi-trusted entity [1], is not fully trusted by its customers usually due to many reasons [2]. Thus, cloud customers are usually reluctant to outsource their sensitive data to cloud in the form of plaintext. An effective solution is to encrypt data before outsourcing.

However, encryption makes effective data retrieval and utilization a very challenging task. Song et al. first introduced a practical technique that allows users to securely search over encrypted data through keywords in [3]. Later, many searchable encryption schemes have been proposed based on symmetric key and public-key setting [4–9] to strengthen security and improve query efficiency. Recently, with the growing popularity of cloud computing, how to securely and efficiently search over encrypted cloud data becomes a research focus. Some approaches have been proposed in [10–18] based on traditional searchable encryption schemes which aim to protect data users' access privacies and query privacies with better query efficiency for cloud computing.

Although existing secure query techniques allow the cloud server to perform effective search over encrypted data without knowing any useful information of data files and user query contents, most of these schemes that are designed based on

* Corresponding author.
E-mail address: zqin@hnu.edu.cn (Z. Qin).

http://dx.doi.org/10.1016/j.jcss.2016.12.003
0022-0000/© 2017 Elsevier Inc. All rights reserved.

Please cite this article in press as: H. Yin et al., A query privacy-enhanced and secure search scheme over encrypted data in cloud computing, J. Comput. Syst. Sci. (2017), http://dx.doi.org/10.1016/j.jcss.2016.12.003

the symmetric encryption setting will straightway leak user query privacies to some internal entities, besides cloud, due to the limitations of the mechanism of query trapdoors generation. More precisely, to obtain the query trapdoors, authorized data users have to ask the data owner for encrypted query keyword(s) or keyword encryption key(s), which not only either requires the data owner to always keep online or brings heavy key management overhead, but also causes the data owner to know fully well the query contents of data users. In fact, in many application scenarios, data users are reluctant to disclose their query contents to any other entities, including data owners, for example, a suspected patient may be not willing to public his query contents to anyone else when he searches electronic medical record data for knowing pathogeneses and symptoms.

On the other hand, obviously, different data users have the same query trapdoor forms when they use identical query keywords to query because all authorized data users have the same query trapdoors returned by data owners or adopt the same keys to encrypt their query keywords. Thus, it is inevitable that a data user explicitly knows other data users' query contents by comparing query trapdoor literally. Moreover, we cannot exclude the possibilities that cloud attempts to collude with a compromised data user to expose query privacies of other data users in an open cloud environment.

To release the participation of the data owner and eliminate query key management cost in the process of query, some searchable encryption schemes based on the public-key setting have been proposed such as [4,19], which allow the data user to generate query trapdoor using own private key. However, these schemes still cannot achieve strong privacy protect if they are applied directly in the cloud environment, because secure indexes are encrypted using public key and, in the process of matching between secure indexes and encrypted query keywords, the cloud is able to obtain query contents easily by dictionary attack. Moreover, every time the data user uses the same private key to encrypt query keywords, thus the cloud can obtain data user's query keywords by analyzing the previous query results and submitted encrypted query keywords.

1.1. Our contributions

In this paper, we propose a query privacy-enhanced secure search scheme over encrypted cloud data based on secure index technique by letting the data user generate query trapdoor using randomly chosen secret keys every time. In our scheme, the query contents of a data user cannot be obtained or inferred by any other entities, including the cloud server, the data owner, and the other data users, except the user data himself. We mainly make three key contributions as follows. First, we present an efficient secure search scheme with strong query privacy protection. Our scheme allows a data user to generate random query trapdoor every time by randomizing query keyword encryption key while enables the cloud server to correctly query over encrypted secure index. Second, security analysis and proof show that our scheme is secure and query privacy-enhanced. Lastly, we implement our scheme, evaluate and compare performances on a real data set with the representative searchable encryption schemes SSE [5] and secure KNN [9].

The rest of our paper is organized as follows. We first review related work in Section 2. In Section 3, we define our system model, threat model, security definition, and several necessary techniques. We define and construct our secure scheme in detail and analyze its correctness in Section 4. In Section 5, we analyze the complexity and security of our proposed scheme. We prove that our scheme is strong privacy protective and secure in Section 6 and evaluate our scheme through practical experiments in Section 7. Lastly, we conclude this paper in Section 8.

2. Related work

2.1. Conventional searchable encryption

Song et al. first introduced a practical technique [3] that allows a data owner to use an unconventional encryption method to encrypt each word of a document and a server to perform secure search by going through the whole encrypted document using a specified encrypted keyword. To improve the efficiency and system availability, in [6], Goh et al. made use of Bloom filters and pseudo-random functions to construct secure searchable index for each encrypted data file and defined search semantic security model against adaptive chosen keyword attack. The construction allows data owners to dynamically update new files without requiring rebuilding existing indexes, but the query privacy may be revealed if keywords have been searched before. To further improve security and search efficiency, in [5], Curtmola et al. adopted the inverted index [20] technique and hash table to design a novel searchable encryption scheme named SSE (Searchable Symmetric Encryption) and formally presented new and stronger security definitions. But, this scheme requires predefining a global keywords dictionary which incurs indexes rebuilding when updating data files. Other schemes based symmetric-key such as [7,8] had also been proposed to improve searchable encryption techniques. In [4] Boneh et al. first constructed a searchable encryption scheme under public-key setting. To improve user query experiences and enrich search functionalities, multi-keywords conjunctive and disjunctive search schemes over encrypted data were proposed in [19,21,22].

2.2. Secure search in cloud computing

The data outsourcing service paradigm promotes the further study on secure privacy-preserving search for cloud computing. Based on SSE [5], Wang et al. [13] first used keyword relevance score to implement top-k secure search over encrypted

Please cite this article in press as: H. Yin et al., A query privacy-enhanced and secure search scheme over encrypted data in cloud computing, J. Comput. Syst. Sci. (2017), http://dx.doi.org/10.1016/j.jcss.2016.12.003

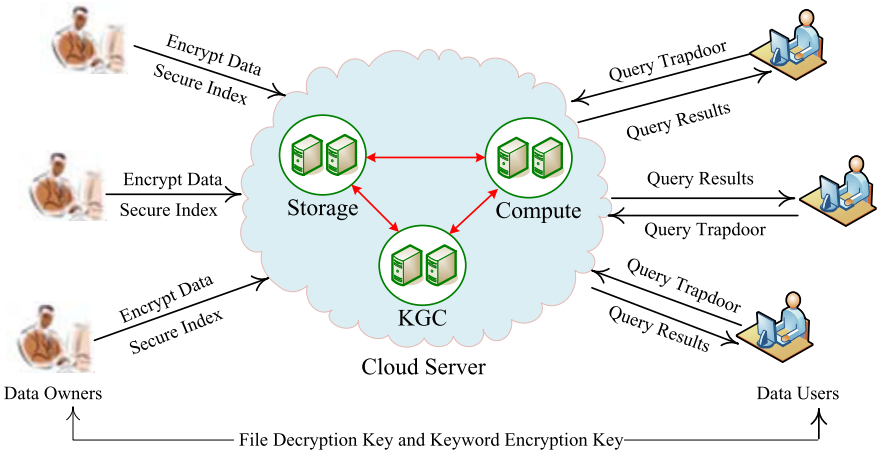

Fig. 1. A system model of secure search over encrypted cloud data.

cloud data by improving the order-preserving symmetric encryption technique [23]. Cao et al. [10] used secure KNN computation scheme [9] to construct a multi-keyword ranked search scheme, and in [14] Sun et al. further developed the scheme and made relevance ranking more accurate by introducing cosine measure based similarity scores for text search. One year later, Sun et al. [15] adopted secure KNN computation again to propose an efficient and query results verifiable multi-keyword secure query scheme by constructing secure index tree and using signature technique. To tolerate minor typos, Li et al. [11] proposed fuzzy keyword search over encrypted cloud data. To further improve the user experience, Wang et al. adopted Bloom filters and secure KNN computation to propose a multi-keyword fuzzy query scheme for cloud computing in [16]. A secure and dynamic secure search scheme in [17] and a personalized secure search scheme in [18] were proposed, in which the KNN is skillfully employed. However, in these schemes, due to limitations of query trapdoors generation, the query keywords of a data user are very easily leaked to other entities such as the data owner or the other data user. In this paper, we emphasize the problem of strong privacy protection of user query without considering ranked query and fuzzy query mechanism to be adapted to those application scenarios that need to strictly protect user query contents.

3. Background information

To clarify our proposed problems, in this section, we present our system model, threat model, security definition, and several supporting techniques used to implement our scheme.

3.1. System model

We adopt a multi-owner and multi-user system architecture of search over encrypted cloud data, which includes cloud, multiple data owners, and multiple data users, as illustrated in Fig. 1. The cloud, including storage centers, computing centers, and key generation centers (KGC), abides by cloud computing protocols to provide services for his clienteles. Data owners use a semantically secure symmetric encryption scheme to encrypt data files and construct secure searchable data indexes, then store them to the data center. Authorized users obtain data decryption key and keyword encryption key from data owners through secret communication channels. The data user generates random query trapdoor for query keywords every time by using the shared keyword encryption key and randomly chosen random numbers. Upon receiving search trapdoor, the computing centers perform search and return query results to the data user.

3.2. Threat model and security definition

3.2.1. Threat model

The semi-trusted threat model has been widely adopted in the secure search work under the cloud environment [10, 12–15,17,18], in which the cloud is considered as an “honest-but-curious” entity. That is, the cloud promises that it always correctly obeys data escrow and functionality protocols, but tries to infer some useful information about data files and query requests because of “curiosities”.

In this paper, the user query need to achieve strong privacy protection, i.e., anyone else other than the query user should not be able to gain the query contents. To evaluate the strength of query privacy protection of data users, we assume that the cloud can collude with another entity to obtain a data user’s query contents, such an entity includes a data owner or a data user. In other word, though data owners and data users are all trusted, they are prohibited from knowing a data user’s query contents.

Please cite this article in press as: H. Yin et al., A query privacy-enhanced and secure search scheme over encrypted data in cloud computing, J. Comput. Syst. Sci. (2017), http://dx.doi.org/10.1016/j.jcss.2016.12.003

3.2.2. Security definition

Our key idea is to design nondeterministic query trapdoor generation mechanism for query to achieve strong privacy protection and build secure searchable index to implement secure query over encrypted cloud data, which involves two security requirements. First, the cloud cannot obtain any plaintext information of data from searchable index. Second, the cloud server, the data owner, and the data users cannot gain query contents of a data user. So, we define the semantic security of searchable index against keyword attack and strong privacy protection of query trapdoor as follows.

Definition 1. Semantic security of searchable index against keyword attack:

The constructed searchable index is semantic secure if any probabilistic polynomial-time adversary $\mathcal{A}$ (i.e., the cloud) chooses two different data files $F1$ and $F2$, lets a simulator $\mathcal{S}$ simulate the data owner to build indexes for $F1$ and $F2$, but $\mathcal{A}$ cannot distinguish which index is for which data file. That is, any adversary $\mathcal{A}$ cannot deduce and obtain any useful plaintext information of data files from their indexes. Please refer to the art [6] for more detailed definition and description about the semantic security of searchable index against keyword attack.

Definition 2. Strong privacy protection of query trapdoor:

For any probabilistic polynomial-time adversary $\mathcal{A}$, who chooses two different keywords $W1$ and $W2$ with the same length, lets a simulator $\mathcal{S}$ simulate the data user to generate query trapdoors for $W1$ and $W2$, but $\mathcal{A}$ cannot distinguish which trapdoor is for which keyword. In other words, if $\mathcal{A}$ cannot determine which keyword is encrypted in a query with probability non-negligible different from 1/2, then we define the generated trapdoor to be strong privacy protective. There is a only restriction that the adversary $\mathcal{A}$ cannot perform search algorithm over encrypted indexes when distinguishing $W1$ and $W2$.

3.3. Basic techniques

In this section, we briefly introduce several important techniques. They are certificateless public key cryptography, Bloom filter, bilinear map, Decisional Diffie–Hellman problem, and Bilinear Diffie–Hellman problem.

3.3.1. Certificateless public key cryptography (CL-PKC)

The certificateless public key cryptography [24] is developed on the foundation of the identity-based cryptography, which eliminates the inherent key escrow problem of the identity-based cryptography (i.e., KGC always knows user secret key) and does not require certificates to guarantee the authenticity of public keys. Yet, CL-PKC still requires to involve a semi-trusted third party KGC to generate the user's partial public key and the partial private key, but the KGC knows noting about user secret information which is necessary to achieve integrated the public/private key.

3.3.2. Bloom filter

A Bloom filter [25] is a space-efficient probabilistic data structure which is used to test whether an element is a member of a set. An empty Bloom filter is a bit array of m bits, where all bits are set to 0 initially. Given a set $W = \{w_1, w_2, ..., w_n\}$ of n elements, in order to insert an element $w \in W$ into a Bloom filter, it needs to feed the element w to k independent hash functions from $\mathcal{H} = \{h_i | h_i : \{0,1\}^* \to [0, m-1], 1 \le i \le k\}$ to get k positions, and sets all these positions to 1. It confirms whether an element q is in W or not, by means of checking whether all the positions corresponding to $h_i(q)$ equal to 1, $1 \le i \le k$. If all are 1, the element q is in W, or resulting in a false positive. If any one of these positions is 0, it is obvious that q is not in W. We can control false positive rate by adjusting Bloom filter parameters. If n elements are inserted into an m-bits Bloom filter which contains k independent hash functions, the false positive rate is $fp = (1 - (1 - \frac{1}{m})^{kn})^k$, where $\frac{1}{m} \to 0$, $fp \approx (1 - e^{-\frac{kn}{m}})^k$. So, given m and n, when $k = \frac{m}{n} \ln 2$, the minimum false positive rate is $2^{-k} \approx 0.6185^{\frac{m}{n}}$.

3.3.3. Bilinear pairing map

In this paper, we use $\mathbb{G}_1$ and $\mathbb{G}_2$ to denote two cyclic multiplicative groups of large prime order q. A bilinear pairing map [26] $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ satisfies the following properties:

- Computable: For any $Q, Z \in \mathbb{G}_1$, there is a polynomial time algorithm to compute $e(Q, Z) \in \mathbb{G}_2$.
- Bilinear: For all $x, y \in \mathbb{Z}_q^*$ and $Q, Z \in \mathbb{G}_1$, the equality $e(Q^x, Z^y) = e(Q, Z)^{xy}$ holds.
- Non-degenerate: If g, h are generators of $\mathbb{G}_1$, then $e(g, h)$ is a generator of $\mathbb{G}_2$.

3.3.4. Decisional Diffie–Hellman (DDH) assumption

Let g represent a generator of $\mathbb{G}_1$. We pick up three elements a, b, c uniformly at random from $\mathbb{Z}_q^*$. The DDH [27] hardness assumption is as follows: for any efficient polynomial time algorithm $\mathcal{A}$, the probability that $\mathcal{A}$ correctly distinguishes (g^a, g^b, g^{ab}) and (g^a, g^b, g^c) is negligible, i.e., there exists a negligible function ε such that

$$|Pr[\mathcal{A}(g^a, g^b, g^{ab}) = 1] - Pr[\mathcal{A}(g^a, g^b, g^c) = 1]| \le \varepsilon(l)$$

where l is a large security parameter, which determines the size of order q.

Please cite this article in press as: H. Yin et al., A query privacy-enhanced and secure search scheme over encrypted data in cloud computing, J. Comput. Syst. Sci. (2017), http://dx.doi.org/10.1016/j.jcss.2016.12.003

Table 1
Notations description.

Notation	Description
DO	a data owner
U	an authorized data user
$\mathcal{F}$	the files collection, denoted as n data files $\mathcal{F} = \{F_1, ..., F_n\}$
$\mathcal{C}$	the ciphertext collection of $\mathcal{F}$, denoted as $\mathcal{C} = \{C_1, ..., C_n\}$, $C_i = Enc(sk, F_i)$, $0 \le i \le n$ Enc denotes a symmetric encryption scheme with the key sk
W_i	the keyword set of a file F_i, denoted as $W_i = \{w_1, w_2, w_3...\}$ extracted from F_i, $0 \le i \le n$
$\mathcal{I}$	the secure searchable index collection of $\mathcal{F}$, denoted as $\mathcal{I} = \{\mathcal{I}_1, ..., \mathcal{I}_n\}$ $\mathcal{I}_i$ is the secure index of corresponding data file F_i, $0 \le i \le n$
$\mathcal{T}_u(w)$	U's query trapdoor, w denotes a query keyword
U_{id}	U's unique identifier
FID_i	F_i's unique identifier $F_i \in \mathcal{F}$, $0 \le i \le n$

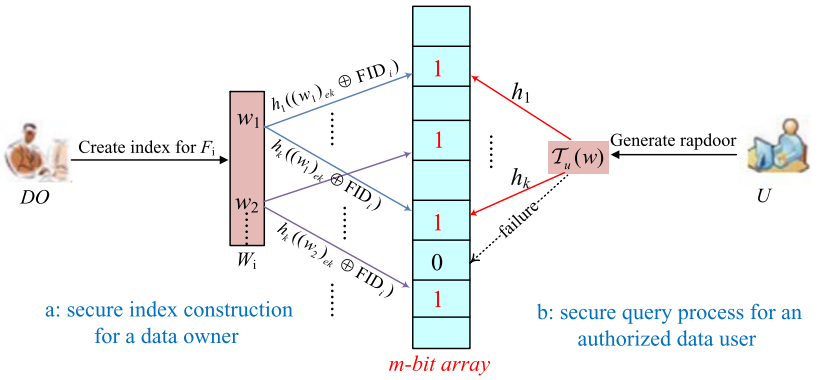

Fig. 2. An overview of our proposed secure search scheme over encrypted cloud data.

3.3.5. Bilinear Diffie–Hellman (BDH) assumption

Let g represent a generator of $\mathbb{G}_1$. We pick up three elements a, b, c uniformly at random from $\mathbb{Z}_q^*$. The BDH [6] hardness assumption is as follows: given (g, g^a, g^b, g^c), for any efficient polynomial time algorithm $\mathcal{A}$, the probability that $\mathcal{A}$ correctly distinguishes (g^a, g^b, g^c) and $(g^a, g^b, e(g,g)^{abc})$ is negligible, i.e., there exists a negligible function ε such that

$$|Pr[\mathcal{A}(g^a, g^b, g^c) = 1] - Pr[\mathcal{A}(e(g,g)^{abc}) = 1]| \le \varepsilon(l)$$

where l is a large security parameter, which determines the size of order q.

4. A privacy-enhanced secure search scheme over encrypted cloud data

4.1. Problem definition

For ease reading, we first describe some notations used in our scheme in Table 1.

According to the system model described in Section 3.1, DO encrypts $\mathcal{F}$ to be $\mathcal{C}$ and creates secure index $\mathcal{I}_i$ for each data file F_i based on the corresponding set W_i to form collection $\mathcal{I}$. $\mathcal{C}$ and $\mathcal{I}$ are outsourced to the cloud server together. To obtain desired data files, U submits query trapdoor to the cloud server, who is responsible for searching over the secure index set $\mathcal{I}$ without obtaining any plaintext information about data files and query keywords. In addition, we consider that the cloud is a typical semi-trusted third party entity (honest-but-curious), which can play a role of KGC to assist U to generate his private key and public key used for authentication.

We give the basic idea of our secure search scheme as illustrated in Fig. 2. The a part denotes the secure index construction process, where $(w_i)_{ek}$ represents the encryption of keyword w_i under key ek, $\{h_1, ..., h_k\}$ denotes k independent hash functions with the same range $\{0, m-1\}$, and '$\oplus$' represents an exclusive-OR operation. The b part denotes the secure search over the m-bit array for an authorized data user, where the solid arrow denotes a successful query and the dotted arrow denotes that F_i is not a query result of keyword w.

Definition 3. A secure search scheme over encrypted cloud data consists of the following six probabilistic polynomial time algorithms.

- **Setup**(1^l): On input a sufficiently large security parameter l, the cloud as KGC runs it to output a system master key and a public parameter tuple.

- **Keygen**($U_{id}, 1^{l'}$): On input U's unique identifier and a sufficiently large security parameter l', the algorithm generates U's public/private key pair and his a signature δ_u.
- **Authen**($pub_u, \{\delta_u, U_{id}\}$): Given U_{id} and δ_u of U, DO runs the algorithm using U's public key pub_u to check whether U is a legal user and determine whether authorize to U.
- **BuildIndex**(W_i, FID_i, ek): DO runs it to build a searchable secure index for each $F_i \in \mathcal{F}$ under the keyword encryption key ek. The algorithm outputs the secure index $\mathcal{I}_i$ of F_i.
- **Trapdoor**(w, ek): U invokes the algorithm to encrypt query keyword w using the encryption key ek and some randomly chosen numbers, outputs $\mathcal{T}_u(w)$. It is a probabilistic encryption algorithm.
- **Query**($\mathcal{T}_u(w), FID_i, \mathcal{I}_i$): Finally, the cloud accepts $\mathcal{T}_u(w)$ and performs search over F_i through its FID_i and encrypted index $\mathcal{I}_i$.

4.2. Scheme construction

We now formally give construction of our scheme. The concrete implementation of each polynomial time algorithm is described as follows.

Setup(1^l): Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two bilinear cyclic multiplicative groups of prime order q, and a bilinear pairing map $e: \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$. Let g denote a random generator of $\mathbb{G}_1$. On input a sufficiently large security parameter l, the system outputs a system master key $mk \in \mathbb{Z}_q^*$ and a keyword encryption key ek and sends ek to all data owners through private communication channels. The system calculates $g_1 = g^{mk}$ as well as chooses three hash functions $H_1: \{0,1\}^* \to \mathbb{Z}_q^*$, $H_2: \{0,1\}^* \to \{0,1\}^s$, $H_3: \{0,1\}^* \to \mathbb{G}_1$, and a hash function family $\mathcal{H} = \{bfh_i | bfh_i: \{0,1\}^* \to [0, m-1], 1 \le i \le k\}$, where s in H_2 is a security strength coefficient. Finally, the KGC opens a system parameter tuple $mpk = \{\mathbb{G}_1, \mathbb{G}_2, g, g_1, H_1, H_2, H_3, \mathcal{H}\}$.

Keygen($U_{id}, 1^{l'}$): This is an interaction between U and KGC. To avoid an illegal user to pretend U to generate his key, U inputs a security parameter l' and the algorithm mainly performs the following two procedures.

- Generation of public/private key: U chooses a large integer $uk \in \mathbb{Z}_q^*$ at random as U's secret value and computes $R_u = g^{uk}$, and transmits $U_{info} = (U_{id}, R_u)$ to KGC through secure communication channels. After receiving the u_{info}, KGC calculates $Q_u = H_1(U_{id})$ and outputs U's partial public key $partial\_pub_u = R_u^{\frac{1}{mk+Q_u}}$. KGC transmits $partial\_pub_u$ to U, and U uses his U_{id} to compute $Q_u' = H_1(U_{id})$ again and checks whether the following two equalities hold, $e(R_u, g) = e(g^{uk}, g)$ and $e(partial\_pub_u^{\frac{1}{uk}}, g_1 g^{Q_u'}) = e(g, g)$. If they hold, which means the returned user information U_{info} is not tampered by the cloud server, U uses his secret value uk to determine his public key $pub_u = (X_u = R_u, Y_u = partial\_pub_u^{uk})$ and private key $pri_u = uk$. If any one of them does not hold, U outputs $\perp$ and aborts partial private/public key.
- Signature: After U generates his public/private key successfully, he randomly chooses a large integer $r_u \in \mathbb{Z}_q^*$ and uses his private key uk to compute a signature of knowledge $\delta_u = \{\delta_1, \delta_2\}$, where $\delta_1 = g^{H_1(U_{id}||r_u)} \in \mathbb{G}_1$ and $\delta_2 = [H_1(U_{id}||r_u) + (uk \times H_1(\delta_1)) \bmod q] \in \mathbb{Z}_q$.

Finally, U will send $\{\delta_u, U_{id}\}$ to the data owner.

Authen($pub_u, \{\delta_u, U_{id}\}$): After receiving $\{\delta_u, U_{id}\}$, DO uses U's public key to verify δ_1 by checking $e(\delta_1, g) = e(g^{\delta_2}, g)/e(X_u^{H_1(\delta_1)}, g)$. If the equality holds, U passes identity authentication and accepts authorization and obtains the data file decryption key sk and keyword encryption key ek. If not, U outputs $\perp$ and DO aborts authorization. Note that how data owners authorize actions to data users and perform fine-grained data access control in the cloud computing environment is another research field and is out of the scope of this paper, and Yu et al. proposed a feasible and practical solution for the theme, please refer to [28] for more details on this. The correctness of the user authentication can be demonstrated as follows.

$$\begin{aligned} e(g^{\delta_2}, g)/e(X_u^{H_1(\delta_1)}, g) &= e(g^{H_1(U_{id}||r_u)+(uk\times H_1(\delta_1))}, g)/e(R_u^{H_1(\delta_1)}, g) \\ &= \frac{e(g^{H_1(U_{id}||r_u)}, g) \times e(g^{(uk\times H_1(\delta_1))}, g)}{e(R_u^{H_1(\delta_1)}, g)} \\ &= \frac{e(g^{H_1(U_{id}||r_u)}, g) \times e(g^{(uk\times H_1(\delta_1))}, g)}{e(g^{uk\times H_1(\delta_1)}, g)} \\ &= e(g^{(H_1(U_{id}||r_u)}, g) \\ &= e(\delta_1, g) \end{aligned}$$

BuildIndex(W_i, FID_i, ek): DO constructs a searchable secure index for each $F_i \in \mathcal{F}$ before he stores data files to the cloud server. The procedure is described as follows:

- DO creates an m-bit array $\mathcal{I}_i$ for the data file F_i and all bits are initially set to 0.

Please cite this article in press as: H. Yin et al., A query privacy-enhanced and secure search scheme over encrypted data in cloud computing, J. Comput. Syst. Sci. (2017), http://dx.doi.org/10.1016/j.jcss.2016.12.003

- For each unique keyword $w \in W_i$ of file F_i
  1. Encrypt w: encrypt w to be $H_3(w)^{ek}$ and calculate $e(H_3(w)^{ek}, g_1)$;
  2. Associate w with F_i: given F_i's identifier FID_i, compute $\mathcal{E}(FID_i) = H_2(FID_i)$ and $\mathcal{C}_w = H_2(e(H_3(w)^{ek}, g_1)) \oplus \mathcal{E}(FID_i)$, where '$\oplus$' represents an exclusive-OR operation;
  3. Create a secure index for F_i: take advantage of the hash functions family $\mathcal{H}$, compute $bfh_1(\mathcal{C}_w)$, $bfh_2(\mathcal{C}_w)$, ..., $bfh_k(\mathcal{C}_w)$, insert them into $\mathcal{I}_i$ by setting these positions to 1.
- Finally, $\mathcal{I}_i$ is regarded as a secure searchable index of F_i.

Trapdoor(w, ek): Given the query keyword w, like previous works, U normally adopts owner-authorized key ek to obtain trapdoor as $H_3(w)^{ek}$. However, the trapdoor is deterministic and cannot achieve strong privacy protection since the same query keyword has the same ciphertext every time. Re-encrypting the value $H_3(w)^{ek}$ by using random key can make the trapdoor harder to break. So, the algorithm randomly chooses an element r from $\mathbb{Z}_q^*$ and re-encrypt $H_3(w)^{ek}$ to be $H_3(w)^{ek} \cdot g^r$. To enable the cloud to effectively perform query over the encrypted data, the algorithm further chooses another random element r' and generates there secret values to be g^r, $g^{r'}$, $e(g, g_1)^{r'}$, which are used in the **Query** algorithm. **Trapdoor** outputs the query trapdoor for w as $\mathcal{T}_u(w) = (H_3(w)^{ek} \cdot g^r, g^r, g_1^{r'}, e(g, g_1)^{r'})$. We can easily see that the algorithm will generate totally different query trapdoor even for the same query keyword due to the random elements r, r'.

Query$(\mathcal{T}_u(w), FID_i, BF_i)$: To obtain query results of trapdoor $\mathcal{T}_u$, given the encoded identifier $\mathcal{E}(FID_i) = H_2(FID_i)$ and the index $\mathcal{I}_i$ of F_i, the cloud server checks whether the following equality holds:

$$\prod_{n=1}^{k} e(g, (g_1^{r'})^{bfh_n(H_2(\frac{e(H_3(w)^{ek} \cdot g^r, g_1)}{e(g^r, g_1)}) \oplus \mathcal{E}(FID_i))/k}) = e(g, g_1)^{r'}$$

we use $bfh_n(H_2(e(H_3(w)^{ek} \cdot g^r, g_1)/e(g^r, g_1)) \oplus \mathcal{E}(FID_i))$ to denote a value (0 or 1) of a position in the array $\mathcal{I}_i$. If the equality holds, the encrypted data file F_i is regarded as a correct search result of $\mathcal{T}_u$. The correctness proof of the **Query** algorithm will be given in next subsection.

In the above algorithms, *Setup* is a system parameters initialization of the whole scheme. We refer to the idea of certificateless public key cryptography to construct U's public/private key pair and simplify user authentication through algorithms *Keygen* and *Authen*, which effectively solve the private key distribution, key escrow, and authentication problems in the cloud environment. In the algorithm *BulidIndex*, we create a secure index for each data file independently using the Bloom filter, which allows a data owner to dynamically update his data files (add or delete data files to or from the cloud) without influencing the indexes of other data files and is more suitable for the cloud environment in contrast to previous solutions, such as [10,11,13–15] that need to rebuild indexes when updating data. A data user generates trapdoor by the algorithm *Trapdoor* and the cloud performs search according to secure indexes and query trapdoor by *Query*.

4.3. Scheme correctness

In this subsection, we give the correctness verification of **Query** algorithm according to the properties of the bilinear map. Specifically, when U wants search some encrypted data files containing the keyword w, he invokes *Trapdoor* to generate a query trapdoor $\mathcal{T}_u(w)$ and then submits it to the cloud, who receives $\mathcal{T}_u(w)$, given the file index BF_i and $\mathcal{E}(ID_i)$ of F_i, and checks whether the following equality holds.

$$\prod_{n=1}^{k} e(g, (g_1^{r'})^{bfh_n(H_2(\frac{e(H_3(w)^{ek} \cdot g^r, g_1)}{e(g^r, g_1)}) \oplus \mathcal{E}(FID_i))/k}) = e(g, g_1)^{r'}$$

If the equality holds, then F_i is a correct query result of w. The correctness of query can be verified by the following derivation and description. Because,

$$\prod_{n=1}^{k} e(g, (g_1^{r'})^{bfh_n(H_2(\frac{e(H_3(w)^{ek} \cdot g^r, g_1)}{e(g^r, g_1)}) \oplus \mathcal{E}(FID_i))/k})$$

$$= e(g, g_1^{r'})^{\frac{1}{k}\sum_{n=1}^{k} bfh_n(H_2(\frac{e(H_3(w)^{ek} \cdot g^r, g_1)}{e(g^r, g_1)}) \oplus \mathcal{E}(FID_i))}$$

For the exponent part, we can reduce:

$$\frac{1}{k}\sum_{n=1}^{k} bfh_n\Big(H_2\Big(\frac{e(H_3(w)^{ek} \cdot g^r, g_1)}{e(g^r, g_1)}\Big) \oplus \mathcal{E}(FID_i)\Big)$$

$$= \frac{1}{k}\sum_{n=1}^{k} bfh_n\Big(H_2\Big(\frac{e(H_3(w)^{ek}, g_1)e(g^r, g)}{e(g^r, g_1)}\Big) \oplus \mathcal{E}(FID_i)\Big)$$

Please cite this article in press as: H. Yin et al., A query privacy-enhanced and secure search scheme over encrypted data in cloud computing, J. Comput. Syst. Sci. (2017), http://dx.doi.org/10.1016/j.jcss.2016.12.003

$$= \frac{1}{k}\sum_{n=1}^{k} bfh_n(H_2(e(H_3(w)^{ek}, g_1)) \oplus \mathcal{E}(FID_i))$$

So,

$$\prod_{n=1}^{k} e(g, (g_1^{r'})^{bfh_n(H_2(\frac{e(H_3(w)^{ek}\cdot g^r, g_1)}{e(g^r, g_1)})\oplus\mathcal{E}(FID_i))/k})$$
$$= e(g, g_1^{r'})^{\frac{1}{k}\sum_{n=1}^{k} bfh_n(H_2(e(H_3(w)^{ek}, g_1))\oplus\mathcal{E}(FID_i))}$$

On the other hand, recall that DO creates the secure searchable index for each file F_i with the keyword set W_i, according to the following way, for each keyword $w' \in W_i$ of F_i, DO computes:

$$\mathcal{C}_{w'} = H_2(e(H_3(w')^{ek}, g_1) \oplus \mathcal{E}(FID_i)$$

and inserts $\mathcal{C}_{w'}$ to an m-bits Bloom filter $\mathcal{I}_i$ using $\mathcal{H}$. Thus, for the query keyword w, if there exists a keyword w' in W_i that satisfies $w' = w$, then the corresponding k positions of $bfh_i(H_2(e(H_3(w)^{ek}, g_1))) \oplus \mathcal{E}(FID_i))$ are 1 in the array $\mathcal{I}_i$ for all $i = 1, 2, ..., k$. That is,

$$\frac{1}{k}\sum_{n=1}^{k} bfh_n\Big(H_2\Big(\frac{e(H_3(w)^{ek}\cdot g^r, g_1)}{e(g^r, g_1)}\Big) \oplus \mathcal{E}(FID_i)\Big)$$
$$= \frac{1}{k}\sum_{n=1}^{k} bfh_n(H_2(e(H_3(w)^{ek}, g_1) \oplus \mathcal{E}(FID_i))$$
$$= \frac{1}{k}\sum_{n=1}^{k} bfh_n(H_2(e(H_3(w')^{ek}, g_1) \oplus \mathcal{E}(FID_i))$$
$$= 1$$

i.e.,

$$\prod_{n=1}^{k} e(g, (g_1^{r'})^{bfh_n(H_2(\frac{e(H_3(w)^{ek}\cdot g^r, g_1)}{e(g^r, g_1)})\oplus\mathcal{E}(FID_i))/k}) = e(g, g_1)^{r'}$$

4.4. About false positive

It is well known that a Bloom filter can introduce a false positive when a query keyword $w \notin W_i$ is submitted by U but the file F_i is regarded as a correct result returning back to the U. We know the false positive rate incurred by the ith file index is $fp_i = (1 - e^{\frac{-k|W_i|}{m}})^k$, where $|W_i|$ is the number of keywords of the file F_i. Let sn denote the current number of outsourced data files and R denote a query result set. We suppose that, in a query using a keyword w, the query result set R does not include any data files (i.e., $|R| = 0$ and no false positive occurrences for all sn file indexes in this query) the probability can be denoted as follows:

$$Pr_{correct} = \prod_{i=1}^{sn}(1 - fp_i) = \prod_{i=1}^{sn}(1 - (1 - e^{\frac{-k|W_i|}{m}})^k)$$

Theoretically, we can calculate an appropriate hash functions number k to maximize the probability $Pr_{correct}$. But in cloud computing environments, the total size of data set sn is unknown and incremental because data files are dynamically updated, which make the calculation for k become difficult. To solve this problem, we estimate an upper bound $|W|_{max}$ on the number of descriptive keywords for each data file. If file F_i's descriptive keywords $|W_i| < |W|_{max}$, we add additional $|W|_{max} - |W_i|$ random elements in W_i. Eventually we set $k = \frac{m}{|W|_{max}} \ln 2$ to minimize false positives incurred by per-index at the price of more index construction overhead and the probability of a false positive is:

$$fp = 2^{-k} = 2^{-\frac{m}{|W|_{max}} \ln 2}$$

For any query result set R, if there are no false positives in R, the probability is:

$$pr_{correct} = (1 - fp)^{sn-|R|}$$
$$= (1 - 2^{-\frac{m}{|W|_{max}} \ln 2})^{sn-|R|}$$

Please cite this article in press as: H. Yin et al., A query privacy-enhanced and secure search scheme over encrypted data in cloud computing, J. Comput. Syst. Sci. (2017), http://dx.doi.org/10.1016/j.jcss.2016.12.003

Table 2
Complexity analysis.

Algorithm	Computation cost	Output size	Complexity
Keygen	$4T_{H_1} + 6E_{\mathbb{G}_1} + M_{\mathbb{Z}_q^*} + 2A_{\mathbb{Z}_q^*} + 4P$	$3\lvert\mathbb{G}_1\rvert + 2\lvert q\rvert$	$\mathcal{O}(1)$
Authen	$T_{H_1} + 2E_{\mathbb{G}_1} + M_{\mathbb{G}_2} + 3P$	–	$\mathcal{O}(1)$
BuildIndex	$\lvert W\rvert(T_{H_3} + E_{\mathbb{G}_1} + 2T_{H_2} + P + T_{\oplus} + kT_{bfh})$	m	$\mathcal{O}(\lvert W\rvert)$
Trapdoor	$T_{H_3} + M_{\mathbb{G}_1} + 3E_{\mathbb{G}_1} + E_{\mathbb{G}_2}$	$3\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_2\rvert$	$\mathcal{O}(1)$
Query	$P + T_{H_2} + T_{\oplus} + kT_{bfh}$	–	$\mathcal{O}(1)$

On the other hand, if there exist some false positives in R, the mathematical expectation of false positives in R can be calculated as:

$$\begin{aligned} E &= fp + 2 \times fp^2 + 3 \times fp^3 + \cdots + |R| \times fp^{|R|} \\ &= \frac{fp \times (1 - fp^{|R|})}{(1 - fp)^2} - \frac{|R| \times fp^{|R|+1}}{1 - fp} \\ &= \frac{fp \times (1 + fp^{|R|+1}) - fp^{|R|+1} \times (1 + |R|)}{(1 - fp)^2} \end{aligned}$$

where $fp = 2^{-\frac{m}{|W|_{max}} \ln 2}$.

5. Complexity and security analysis

5.1. Complexity analysis

In this subsection, we theoretically analyze the complexity of the each algorithm in our proposed secure search scheme. For ease of reading, the computation cost, output size, and computation complexity are described in Table 2. The notations T_{H_1}, T_{H_2}, T_{H_3}, T_{bfh} denote the computation cost of hash function H_1, H_2, H_3, and bfh_i $(1 \le i \le k)$, respectively. $T_{\oplus}$ denotes the run time of an XOR operation. P denotes a pairing operation, $E_{\mathbb{G}_1}$ and $E_{\mathbb{G}_2}$ denote an exponentiation operation in $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, $M_{\mathbb{G}_1}$ and $M_{\mathbb{G}_2}$ denote a multiplication in $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, $M_{\mathbb{Z}_q^*}$ denotes a multiplication in $\mathbb{Z}_q^*$, and $A_{\mathbb{Z}_q^*}$ denotes an addition in $\mathbb{Z}_q^*$. W is the keyword set of a data file F and m is the bit length of Bloom filter.

5.2. Security analysis

In this subsection, we analyze the security of the scheme from three aspects, i.e., query privacy, data files, and the searchable index.

Query Privacy: For implementing strong privacy protection, we require that anyone else cannot obtain the query contents other than the query user himself, including the cloud server, data owners, and other data users. Recall that a data user u generates the query trapdoor for a query keyword w as $\mathcal{T}_u(w) = (H_3(w)^{ek} \cdot g^r, g^r, g_1^{r'}, e(g, g_1)^{r'})$, where r, r' are two random elements chosen by the u for this query. If other entities wish to learn the actual value of the trapdoor or distinguish two trapdoors, they have to solve the discrete logarithm problem in $\mathbb{Z}_q^*$ with large prime q, however, which is hard further according to the DDH and BDH assumptions. Therefore, our proposed trapdoor construction achieves strong privacy protection for every time query due to the randomly chosen elements r, r' as long as the discrete logarithm problem is hard.

Data Files: Recall that DO uses the semantically secure symmetric encryption scheme $Enc(sk, \cdot)$ to encrypt files before outsourcing. The semantic security of $Enc(sk, \cdot)$ guarantees data files confidentiality.

Searchable Index: Give a keyword set W_i of data file F_i, for each keyword $w \in W_i$, DO first encrypts the w to be $H_3(w)^{ek}$ and calculates $e(H_3(w)^{ek}, g_1)$, i.e., $e(H_3(w), g_1)^{ek}$. Obviously, the cloud cannot obtain the random value in $\mathbb{G}_2$ as long as the ek is kept secretly under the BDH assumption. Thus, in $\mathcal{I}_i$, the cloud cannot determine the k positions of inserting the keyword w by using hash functions $bfh_1, \ldots, bfh_k$. In addition, an adversary may be able to reveal the approximate number of keywords of each data file by the amount of 1 s in each Bloom filter. To hide the information on how much each data file has keywords, we have estimated an appropriate upper bound $|W|_{max}$ on the number of keywords and pad some random elements for each file so that the number of keywords of each data file equals to $|W|_{max}$. Detailed proof will be given in next section.

6. Security proof

We will adopt the same way used in [5] to prove that our scheme is semantically secure and strong privacy protective under the BDH and DDH assumptions.

Please cite this article in press as: H. Yin et al., A query privacy-enhanced and secure search scheme over encrypted data in cloud computing, J. Comput. Syst. Sci. (2017), http://dx.doi.org/10.1016/j.jcss.2016.12.003

We first briefly introduce some necessary notions used for our security proof, referring to [5] to get more detailed information about these notions.

History H: Given an outsourced data file set $\mathcal{F}$ and a query vector $W_q = \{w_1, w_2, ..., w_q\}$, we use $H = (\mathcal{F}, W_q)$ to denote a history. For any keyword $w \in W_q$, we can search one or more data files in $\mathcal{F}$ via w.

View V: The view is the encrypted version of the History. It includes an encrypted data files set $\mathcal{C} = \{Enc(sk, F_1), ..., Enc(sk, F_n)\}$, $F_i \in \mathcal{F}$, an encrypted query keyword set $\mathcal{T} = \{\mathcal{T}(w_1), ...\mathcal{T}(w_q)\}$, and a built secure index set $\mathcal{I}$. Formally, we denote the view of a history as $V(H) = (\mathcal{C}, \mathcal{I}, \mathcal{T})$. Intuitively, the cloud can only see the view V of the history H.

Trace T: Given a history H, a trace represents a query result set induced by H and denoted as $T(H) = \{(|C_i|, C_i), w_j \in C_i \wedge w_j \in W_q, 1 \le i \le n, 1 \le j \le q\}$, where $C_i = Enc(sk, F_i)$ is a query result of the keyword $w_j \in W_q$. The cloud knows nothing beyond the ciphertext result C_i and its length $|C_i|$ in a query.

Theorem 1. *If DDH and BDH problem assumptions hold for any polynomial time adversary, our constructed searchable indexes achieve semantic security against choose keyword attack security and the query contents of a data user achieve strong privacy protection.*

Proof. According to [5] and the definition of IND-CKA in [6], there exists a polynomial size simulator $\mathcal{S}$ which can simulate a view $V^*(H)$ based on the trace $T(H)$. We claim that a searchable encryption scheme is semantically secure if all polynomial-size adversaries cannot distinguish $V(H)$ from $V^*(H)$. Given the *trace* $T(H) = \{(|C_i|, C_i), 1 \le i \le n\}$, the $\mathcal{S}$ does the following operations to simulate $V^*(H)$.

Simulating $\mathcal{C}$: $\mathcal{S}$ selects a random $C_i^* \in \{0,1\}^{|C_i|}$ and a random file identifier $FID_i^* \in \{0,1\}^s$, $1 \le i \le n$. $\mathcal{S}$ outputs $\mathcal{C}^* = \{(C_i^*, FID_i^*), 1 \le i \le n\}$.

Simulating $\mathcal{T}$: $\mathcal{S}$ first generates an empty set $\mathcal{T}^*$. For each $w_j \in W_q$, $\mathcal{S}$ chooses four random values $w_{j1}^*, \alpha, \beta \in \mathbb{G}_1$ and $\gamma \in \mathbb{G}_2$ that satisfy $e(g, \beta) = \gamma$. Let $\mathcal{T}(w_j) = (w_{j1}^* \cdot \alpha, \alpha, \beta, \gamma)$, $\mathcal{S}$ puts $\mathcal{T}(w_j)$ to $\mathcal{T}^*$ and outputs $\mathcal{T}^* = \{\mathcal{T}(w_j), 1 \le j \le q\}$.

Simulation $\mathcal{I}$: $\mathcal{S}$ first generates a bit array $\mathcal{I}_i^*$ with the length of m, all bits are set 0 initially. Then, for each $C_i^* \in \mathcal{C}^*$, if $w_j \in W_q$ and C_i is a query result of the keyword w_j, $\mathcal{S}$ takes the first random value w_{j1}^* from $\mathcal{T}(w_j) \in \mathcal{T}^*$ and computes $\mathcal{C}_{w_j}^* = H_2(e(w_{j1}^*, g_1)) \oplus H_2(ID_i^*)$. Finally, $\mathcal{S}$ exploits the hash functions family $\mathcal{H}$, computes $bfh_1(\mathcal{C}_{w_j}^*)$, $bfh_2(\mathcal{C}_{w_j}^*)$, ..., $bfh_k(\mathcal{C}_{w_j}^*)$, and inserts them into the $\mathcal{I}_i^*$. $\mathcal{S}$ outputs $\mathcal{I}^* = \{\mathcal{I}_i^*, 1 \le i \le n\}$.

Obviously, the view $V^*(H)$ simulated by the $\mathcal{S}$ has the same trace as the $V(H)$ that the cloud has, and we can easily verify the conclusion by searching on $\mathcal{I}^*$ via query trapdoor $\mathcal{T}(w_j) \in \mathcal{T}^*$ for each j, $1 \le j \le q$. We claim that no polynomial-size adversary can distinguish the view $V(H)$ from $V^*(H) = \{\mathcal{C}^*, \mathcal{T}^*, \mathcal{I}^*\}$.

Distinguish ($\mathcal{C}$ and $\mathcal{C}^*$): We adopt the semantically secure symmetric encryption scheme $Enc(sk, \cdot)$ to encrypt data files in $\mathcal{F}$. The semantic security of $Enc(sk, \cdot)$ guarantees that no polynomial-size adversary can distinguish between $\mathcal{C}$ and $\mathcal{C}^*$.

Distinguish ($\mathcal{T}$ and $\mathcal{T}^*$): $\mathcal{S}$ chooses four random values $\mathcal{T}(w_j) = (w_j^* \cdot \alpha, \alpha, \beta, \gamma)$ to simulate the query trapdoor for each keyword $w_j \in W_q$ submitted by U, where $w_j^* \cdot \alpha, \alpha, \beta \in \mathbb{G}_1$ and $\gamma \in \mathbb{G}_2$. In practice, we know that U uses the keyword encryption key ek and two randomly chosen elements r, r' to generate the trapdoor of w_j for this query as $(H_3(w_j)^{ek} \cdot g^r, g^r, g_1^{r'}, e(g, g_1)^{r'})$, where $(H_3(w_j)^{ek} \cdot g^r, g^r, g_1^{r'} \in \mathbb{G}_1$ and $e(g, g_1)^{r'} \in \mathbb{G}_2$. As long as the random elements r, r' are kept confidentially for this query, anyone else except U cannot correctly distinguish the tuple $(H_3(w_j)^{ek} \cdot g^r, g^r, g_1^{r'}, e(g, g_1)^{r'})$ from $(w_{j1}^* \cdot \alpha, \alpha, \beta, \gamma)$, therefore no polynomial-size adversary can distinguish between $\mathcal{T}$ and $\mathcal{T}^*$. This also further demonstrates that the cloud server cannot collude with a data owner or a data user to obtain U's query keywords and our scheme achieves strong privacy protection if the DDH and BDH assumptions hold.

Distinguish ($\mathcal{I}$ and $\mathcal{I}^*$): For a data file F_i, for each keyword w_j, $\mathcal{S}$ randomly chooses an element w_j^*, calculates $e(w_j^*, g_1) \in \mathbb{G}_2$, and inserts it into the BF_i^* using k hash functions. In practice, DO encrypts w_j as $H_3(w_j)^{ek}$ and calculates $e(H_3(w_j)^{ek}, g_1) = e(H_3(w_j), g_1)^{ek} \in \mathbb{G}_2$ and inserts it into the $\mathcal{I}_i$. We say that the cloud cannot distinguish between $e(w_j^*, g_1)$ and $e(H_3(w_j), g_1)^{ek}$ as long as ek is kept secretly under the BDH assumption. Consequentially, no polynomial-size adversary can correctly distinguish between $\mathcal{I}$ and $\mathcal{I}^*$. □

7. Experimental evaluation

7.1. Evaluation setup

To evaluate the efficiency and performance of our proposed scheme more realistically, we choose a real data set RFC (Request For Comments Database) [29] as our experimental data. So far, this database contains 7389 plain text files and the total size achieves 378 MB. We select 5000 data files to build experimental subset and extract 100 keywords for each RFC file as its keyword set by reserving technical keywords and removing Stop Words (e.g., 'the', 'is', 'at', 'want', 'should', and so on). We should fill some dummy elements for data files if the number of the extracted keywords is less than 150. We set

Please cite this article in press as: H. Yin et al., A query privacy-enhanced and secure search scheme over encrypted data in cloud computing, J. Comput. Syst. Sci. (2017), http://dx.doi.org/10.1016/j.jcss.2016.12.003

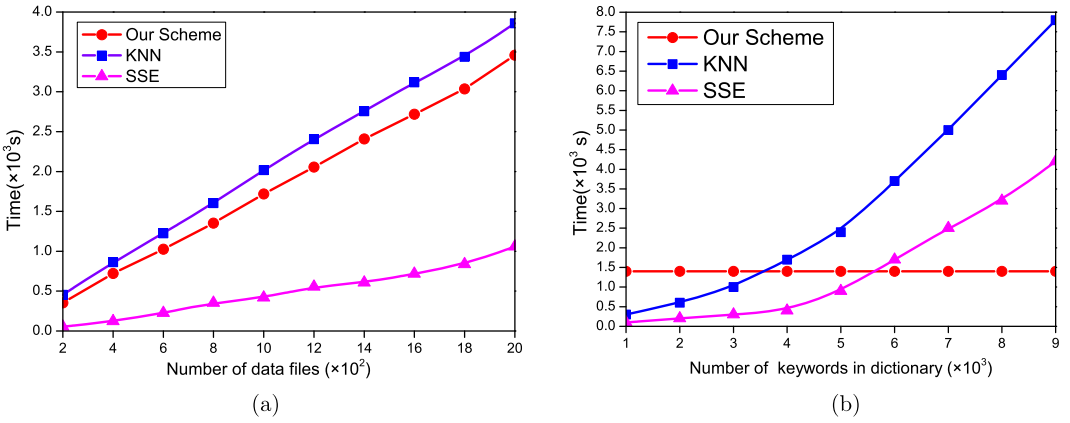

Fig. 3. (a) The running time of secure index construction with different number of data files and the same size of keywords dictionary (4000). (b) The running time of secure index construction with different size of keywords dictionary and the fixed number of data files (1000).

$m = 1000$ and $n = 100$, to minimize the false positive induced by Bloom filter, and set $k = \ln 2 \times (1000/100) = 0.693 \times 10 = 7$ and the false positive is around $0.6185^{m/n} = 0.6185^{10} = 0.819\%$.

The software and hardware configurations are as follows. The client side is a Windows 7 desktop system with Intel Core 2 Duo CPU 2.26 GHz, 3 GB memory, and 320 GHz hard driver. The server side is a virtual machine with Core 2 Duo CPU 4×2.394 GHz, 8 GB memory on the Dell blade server M610, and the Linux Centos5.8 OS.

7.2. Implementation details

We adopt Java language relying on JPBC, which is Java library of the Pairing-Based Cryptography Library [30], to implement bilinear pairing operations. In the experiment, we choose the type A elliptic curve group which is a symmetric bilinear setting, has 160-bit prime order and can achieve 1024-bit security level. You can download JPBC library and read the detailed guide for it in [31]. In addition, we use the AES symmetry encryption scheme to instantiate $Enc(sk, \cdot)$ used to encrypt data files and use the message digest algorithm SHA to implement H_2 hashing $\{0,1\}^*$ to a string of length $s = 160$. For $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$, we invoke the API function *getZr().newRandomElement()* to achieve an equal functionality. To map an arbitrary keyword $w \in \{0,1\}^*$ to elliptic curve group for implementing $H_3 : \{0,1\}^* \rightarrow \mathbb{G}_1$, we first make use of the MD5 algorithm to compute a 128-bit hash value $hv(w)$ and revoke the API function *getG().newElementFromHash(hv(w),0,hv(w).length)* to get an elliptic curve parameter for w.

7.3. Evaluation results

To evaluate the performance of our scheme and compare it with several developed searchable encryption technologies, we implement the secure KNN computation scheme [9] and the SSE scheme [5], which have been deeply exploited to develop secure search schemes over encrypted cloud data in [10,11,13–18]. The SSE is a single keywords secure search scheme based on encrypted inverted indexes and secure KNN is a multiple keywords secure search scheme based on the encrypted vector space model. Though they are preeminent searchable encryptions, both of them cannot achieve strong privacy protection of query keywords due to the limitations of generating query trapdoors for data users. In addition, a pre-defined keywords dictionary is necessary to implement SSE and KNN.

7.3.1. Index construction

Fig. 3(a) shows that the time cost of secure index construction for the three schemes increases linearly with the number of data files when fixing the size of the keywords dictionary as 4000. We can see that the KNN and Our Scheme spend much more time on index construction than SSE. This is because the multiplication operations between vectors and matrixes in KNN and the pairing operations in Our Scheme spend more time than pseudo-random functions and symmetrical encryption in SSE. When the number of data files reaches 2000, the time costed on index construction in KNN and Our Scheme is about 4000 s and 3500 s, respectively. Fig. 3(b) shows the time cost for the three schemes when varying the size of the keywords dictionary and fixing the number of data files as 1000. We can observe that the size of the keywords dictionary has no influence on index construction for Our Scheme, while the KNN increases from 300 s to 7800 s and SSE increases from 110 s to 4300 s when the size of the dictionary increases from 1000 to 9000. Although the index construction consumes relatively much time at the data owner side, it is noteworthy that the process is a one-time operation.

7.3.2. Trapdoor generation

Fig. 4(a) shows the time cost of trapdoor generation for these schemes with the same size of keywords dictionary as 4000 and different number of query keywords varied from 2 to 16. We can observe that Our Scheme and SSE all increase linearly

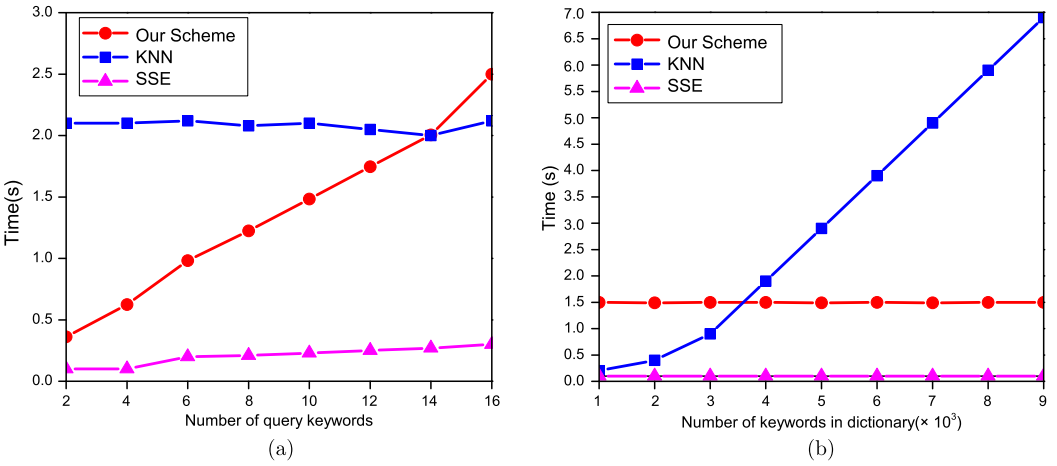

Fig. 4. (a) The running time of trapdoor generation with different number of query keywords and the same size of keywords dictionary (4000). (b) The running time of trapdoor generation with different size of keywords dictionary and the fixed number of query keywords (10).

with an increasing number of query keywords, while Our Scheme consumes more time to generation query trapdoor for data users due to the time-consuming pairing and exponentiation operations on group. When the number of query keywords reaches 16, the consumed time on trapdoor generation in Our Scheme is about 2.6 s. It is acceptable for data users. In addition, when the number of query keywords is less than 14, KNN needs to spend more time to generate query trapdoor compared to Our Scheme, while it is insensitive to the number of query keywords for trapdoor generation when fixing the size of the keywords dictionary. Fig. 4(b) shows the time cost of trapdoor generation for these schemes when varying the size of the keywords dictionary and fixing the number of query keywords as 10. We can see that Our Scheme and SSE are not affected by the size of the keywords dictionary since Our Scheme and SSE on trapdoor generation are independent of the pre-defined keywords dictionary, while KNN suffer from an approximatively quadratic growth with the size of keyword dictionary increases. Among them, the SSE consumes the least time to generate query trapdoor for data users.

7.3.3. Search efficiency

Fig. 5 shows that Our Scheme needs to spend more time for searching at the cloud server side compared with KNN and SSE. This is because that our search algorithm involves the pairing operations that need more computation time than symmetrical encryption used in KNN and SSE. However, Our Scheme can achieve effective strong privacy protection for user's query contents by employing the pairing operation very flexibly that KNN and SSE cannot implement due to the symmetrical encryption. Fig. 5(a) shows that the search time of the three schemes increases linearly with an increasing number of data files and Fig. 5(b) shows that the KNN is insensitive to the number of query keywords in the query process while the number of query keywords has a direct influence on search efficiency for Our Scheme and SSE. Fig. 5(c) further shows that Our Scheme needs not the pre-defined keywords dictionary that is necessary for implementing KNN and SSE.

7.3.4. Search precision

The two most frequent and basic measures for information retrieval effectiveness are *precision* and *recall*. We use them to evaluate our search accuracy. The *Precision* is the fraction of retrieved files that are relevant:

$$Precision = \frac{\#(relevant\ items\ retrieved)}{\#(retrieved\ items)}$$

The *Recall* is the fraction of relevant files that are retrieved:

$$Recall = \frac{\#(relevant\ items\ retrieved)}{\#(relevant\ items)}$$

Because Bloom filters do not introduce false negatives, so the *recall* always keeps 100%. With increased number of query keywords, the false positives introduced by the Bloom filter increase and the #(*retrieved items*) may contain some irrelevant results. From the Fig. 6(a), we can see that, when the number of hash functions is equal to 1, the secure indexes of data files lead to an extremely low *precision* and the query precision gradually increases with an increasing number of hash functions, and approximatively achieves 100% when the number of hash functions is set to the optimal value $k = 7$. Fig. 6(b) shows that, when fixing the size of the data file set and varying the number of the query keywords from 2 to 14, the query precision slightly decreases with the increasing number of the query keywords while *recall* still always keeps 100%. We can see that *precision* is not less than 97.7% when the number of query keywords achieves 14, which is acceptable for data users in practice.

Please cite this article in press as: H. Yin et al., A query privacy-enhanced and secure search scheme over encrypted data in cloud computing, J. Comput. Syst. Sci. (2017), http://dx.doi.org/10.1016/j.jcss.2016.12.003

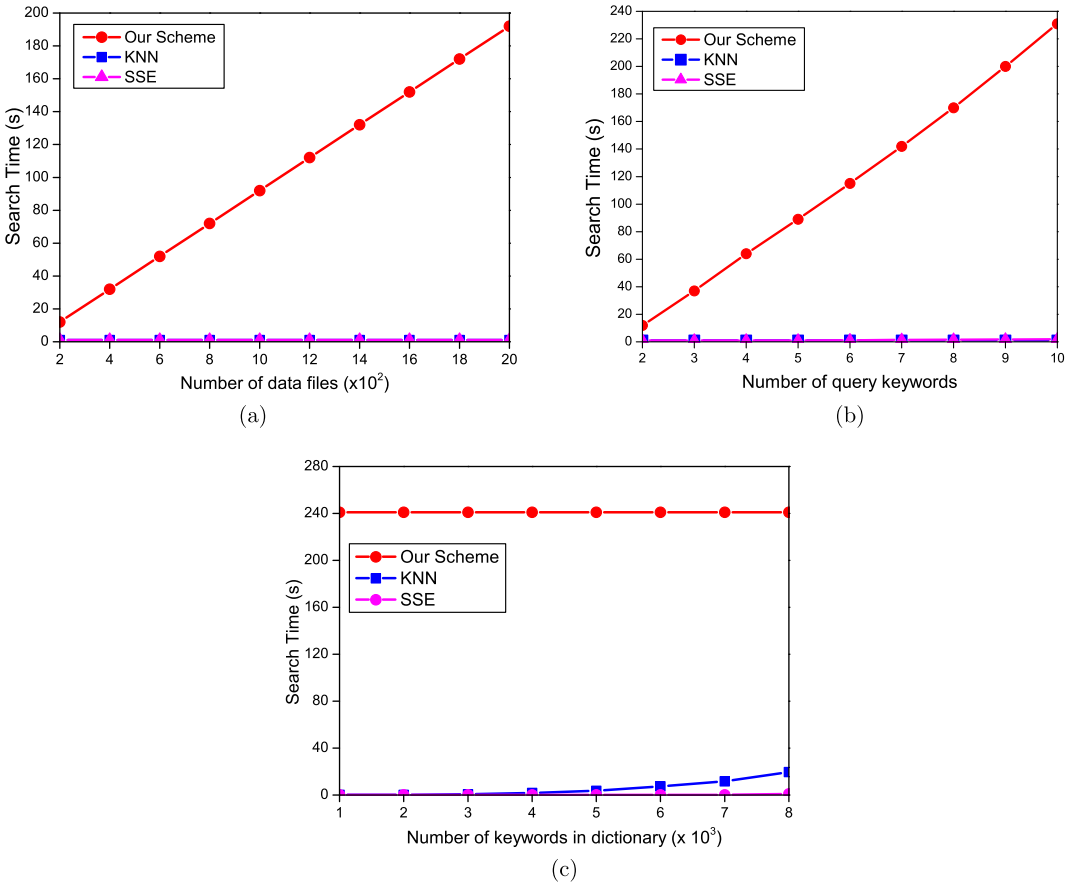

Fig. 5. (a) The search time for different size of data files and with the same size of keywords dictionary (4000), and query keywords (10). (b) The search time for different size of query keywords with the same size of data files (2000) and the same dictionary size (4000). (c) The search time for different size of dictionary with the same size of data files (2000), and query keywords (10).

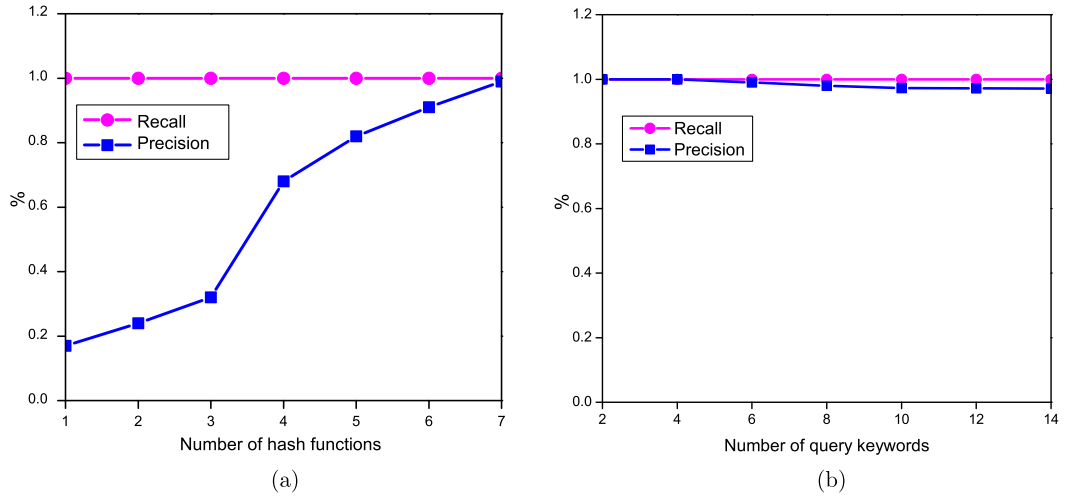

Fig. 6. (a) Evaluation of retrieval effectiveness for different number of hash functions in Bloom filter with the fixed size of query keywords (10) and data files (2000). (b) Evaluation of retrieval effectiveness for different numbers of query keywords with the fixed file size (2000) and the hash functions ($k = 7$) in Bloom filter.

8. Conclusion

In this paper, we propose a query privacy-enhanced and secure keyword query scheme for the cloud computing. Our scheme can correctly and efficiently perform search over encrypted cloud data with strong privacy protection under a

Please cite this article in press as: H. Yin et al., A query privacy-enhanced and secure search scheme over encrypted data in cloud computing, J. Comput. Syst. Sci. (2017), http://dx.doi.org/10.1016/j.jcss.2016.12.003

stronger threat model. We theoretically prove that our scheme is secure and query privacy-enhanced based on DDH and BDH hardness assumptions. Finally, the extensive experiments demonstrate the validity and practicality of our proposed scheme.

Acknowledgments

The research is supported in part by the National Natural Science Foundation of China under Grant Nos. 61472131, 61272546; Science and Technology Key Projects of Hunan Province (2015TP1004, 2016JC2012); Open Research Fund of Hunan Provincial Key Laboratory of Network Investigational Technology (2016WLZC011); Open Research Fund of Key Laboratory of Network Crime Investigation of Hunan Provincial Colleges (2015HNWLFZ056); Opening Project of Shanghai Key Laboratory of Integrated Administration Technologies for Information Security (AGK201609).

References

[1] R. Canetti, U. Feige, O. Goldreich, M. Naor, Adaptively secure multi-part computation, in: ACM 28th STOC, 1996, pp. 639–648.
[2] T. Mather, S. Kumaraswamy, S. Latif, Cloud Security and Privacy: an Enterprise Perspective on Risks and Compliance, OReilly Media, 2009.
[3] D. Song, D. Wagner, A. Perrig, Practical techniques for searches on encrypted data, in: IEEE Symposium on Security and Privacy, vol. 8, 2000, pp. 44–55.
[4] D. Boneh, G. Crescenzo, R. Ostrovsky, G. Persiano, Public-key encryption with keyword search, in: C. Cachin (Ed.), Eurocrypt, in: Lect. Notes Comput. Sci., Springer-Verlag, 2004.
[5] R. Curtmola, J. Garay, S. Kamara, R. Ostrovsky, Searchable symmetric encryption: improved definitions and efficient constructions, in: ACM CCS, vol. 19, 2006, pp. 79–88.
[6] E.-J. Goh, Secure Indexes, Tech. rep., IACR ePrint Cryptography Archive, 2003, http://eprint.iacr.org/2003/216.
[7] Y.-C. Chang, M. Mitzenmacher, Privacy preserving keyword searches on remote encrypted data, in: ACNS, Springer, 2005.
[8] B. Waters, D. Balfanz, G. Durfee, D. Smetters, Building an encrypted and searchable audit log, in: ISOC NDSS, 2004.
[9] W. Wong, D. Cheung, B. Kao, N. Mamoulis, Secure KNN computation on encrypted databases, in: ACM 35th SIGMOD International Conference on Management of Data, 2009, pp. 139–152.
[10] N. Cao, C. Wang, M. Li, K. Ren, W. Lou, Privacy-preserving multi-keyword ranked search over encrypted cloud data, in: IEEE INFOCOM, 2011, pp. 829–837.
[11] J. Li, Fuzzy keyword search over encrypted data in cloud computing, in: IEEE INFOCOM Mini-Conference, 2010.
[12] M. Li, S. Yu, N. Cao, W. Lou, Authorized private keyword search over encrypted data in cloud computing, in: IEEE ICDCS, 2011, pp. 383–392.
[13] C. Wang, N. Cao, J. Li, K. Ren, W. Lou, Secure ranked keyword search over encrypted cloud data, in: IEEE ICDCS, 2010, pp. 253–262.
[14] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, T. Hou, H. Li, Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking, in: ACM ASIACCS, 2013.
[15] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, T. Hou, H. Li, Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking, IEEE Trans. Parallel Distrib. Syst. 25 (11) (2013) 71–82.
[16] B. Wang, S. Yu, W. Lou, Y. Hou, Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud, in: IEEE INFOCOM, 2014, pp. 2112–2120.
[17] Z. Xia, X. Wang, X. Sun, Q. Wang, A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data, IEEE Trans. Parallel Distrib. Syst. 27 (2) (2015) 340–352.
[18] Z. Fu, K. Ren, J. Shu, X. Sun, F. Huang, Enabling personalized search over encrypted outsourced data with efficiency improvement, IEEE Trans. Parallel Distrib. Syst. 27 (9) (2016) 2546–2559.
[19] P. Golle, J. Staddon, B. Waters, Secure conjunctive keyword search over encrypted data, in: ACNS, vol. 3089, Springer, 2004, pp. 31–45.
[20] A. Singhal, Modern information retrieval: a brief overview, IEEE Data Eng. Bull. 24 (4) (2001) 35–43.
[21] L. Ballard, S. Kamara, F. Monrose, Achieving efficient conjunctive keyword searches over encrypted data, in: IEEE ICICS, 2005.
[22] D. Boneh, B. Waters, Conjunctive, subset, and range queries on encrypted data, in: TCC, Springer, 2007, pp. 535–554.
[23] A. Boldyreva, N. Chenette, Y. Lee, A. O'Neill, Order-preserving symmetric encryption, in: EUROCRYPT, in: Lect. Notes Comput. Sci., vol. 5479, Springer, 2009.
[24] S. Al-Riyami, K. Paterson, Certificateless public key cryptography, in: ASIACRYPT, in: Lect. Notes Comput. Sci., vol. 2894, Springer-Verlag, 2003, pp. 452–473.
[25] B. Bloom, Space/time trade-offs in hash coding with allowable errors, Commun. ACM 13 (7) (1970) 422–426.
[26] D. Boneh, M. Franklin, Identity-based encryption from the Weil pairing, in: J. Kilian (Ed.), CRYPTO, in: Lect. Notes Comput. Sci., vol. 2139, Springer, 2001, pp. 213–229.
[27] D. Boneh, The decision Diffie–Hellman problem, in: 3rd Algorithmic Number Theory Symposium, in: Lect. Notes Comput. Sci., vol. 1423, Springer-Verlag, 1998, pp. 48–63.
[28] S. Yu, C. Wang, K. Ren, W. Lou, Achieving secure, scalable, and fine-grained data access control in cloud computing, in: IEEE INFOCOM, 2010.
[29] Rfc: request for comments database, http://www.ietf.org/rfc.html.
[30] http://crypto.stanford.edu/pbc/.
[31] http://gas.dia.unisa.it/projects/jpbc/index.html.