Future Generation Computer Systems 72 (2017) 208–218

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A multi-user searchable encryption scheme with keyword authorization in a cloud storage

Zuojie Deng [a,b,*], Kenli Li [c,d], Keqin Li [c,d,e], Jingli Zhou [a]

[a] School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China
[b] School of Computer and Communication, Hunan Institute of Engineering, Xiangtan, Hunan, 411104, China
[c] College of Information Science and Engineering, Hunan University, Changsha, Hunan 410082, China
[d] National Supercomputing Center in Changsha, Changsha, Hunan 410082, China
[e] Department of Computer Science, State University of New York, New Paltz, NY 12561, United States

HIGHLIGHTS

- A security model of keyword authorization search over encrypted files is defined.
- We propose a multi-user searchable encryption scheme with keyword authorization.
- To describe keyword authorization relationships, a KABtree is defined.
- We construct MSESK with asymmetric bilinear map groups of Type-3 and KABtrees.
- The performance evaluation experiments explain the feasibility of MSESKA.

ARTICLE INFO

Article history:
Received 28 October 2015
Received in revised form
7 March 2016
Accepted 18 May 2016
Available online 1 June 2016

Keywords:
Cloud storage
Encrypted data
Keyword authorization
Multi-user searchable encryption

ABSTRACT

Multi-user searchable encryption (MSE) allows a user to encrypt its files in such a way that these files can be searched by other users that have been authorized by the user. The most immediate application of MSE is to cloud storage, where it enables a user to securely outsource its files to an untrusted cloud storage provider without sacrificing the ability to share and search over it. Any practical MSE scheme should satisfy the following properties: concise indexes, sublinear search time, security of data hiding and trapdoor hiding, and the ability to efficiently authorize or revoke a user to search over a file. Unfortunately, there exists no MSE scheme to achieve all these properties at the same time. This seriously affects the practical value of MSE and prevents it from deploying in a concrete cloud storage system. To resolve this problem, we propose the first MSE scheme to satisfy all the properties outlined above. Our scheme can enable a user to authorize other users to search for a subset of keywords in encrypted form. We use asymmetric bilinear map groups of Type-3 and keyword authorization binary tree (KABtree) to construct this scheme that achieves better performance. We implement our scheme and conduct performance evaluation, demonstrating that our scheme is very efficient and ready to be deployed.

1. Introduction

1.1. Motivation

Cloud storage has become a prevalent storage scheme in recent years, where a user can store and share its files [1]. However, the cloud storage provider is not fully trusted, if a user outsources its confidential files to a remote cloud storage server in plaintext form, it may cause some horrible privacy leakage. A promising approach to protect confidential files for a user in a cloud storage is to encrypt its files by using a secure symmetric encryption algorithm, e.g. AES. However, storing files in encrypted form will make some useful file operation functions, such as search, sharing, etc., unavailable. If a user cannot share and search over its files on a remote cloud storage server, it will be reluctant to outsource its files to the cloud storage.

To resolve the searchable problem of encrypted files in a cloud storage server, we can use the searchable encryption technology

* Corresponding author at: School of Computer of Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China.
E-mail addresses: zjdeng@hotmail.com (Z. Deng), lkl@hnu.edu.cn (K. Li), lik@newpaltz.edu (K. Li), jlzhou@hust.edu.cn (J. Zhou).

http://dx.doi.org/10.1016/j.future.2016.05.017
0167-739X/

in the literature [2–9]. A user can store its files in encrypted form at an untrusted cloud server by using these searchable encryption schemes, and delegate the cloud server to search over its files by issuing a trapdoor (i.e. encrypted keyword). However, all these schemes are limited to the single-user setting where the file owner who generates these encrypted files is also the single-user performs searches on it. Therefore, their schemes cannot support encrypted file sharing. Generally speaking, there exists file sharing between a set of users in the cloud storage, and the file owner should authorize other users to search over its encrypted files. Since above schemes do not support this, we cannot apply these schemes to cloud storage directly. Curtmola et al. suggested to share the secure key for file search among some users by extending their single-user scheme directly [4]. Later, other researchers proposed some schemes for multi-user [10–15]. In all these schemes, there exists a user manager to manage the search capabilities of multiple users (e.g. enable them to search each other's files). However, there usually exists no trusted user administrator in a cloud storage, so all these multi-user schemes cannot directly be applied to cloud storage setting as well. To resolve this problem, Popa et al. proposed a searchable encryption scheme that enables keyword search on files encrypted with different keys [16]. But the granularity of authorization in their scheme is very coarse, and they did not explicitly specify how u_i can authorize u_j to search its indexes, where $i \neq j$. Subsequently, Tang extended the Popa–Zeldovich scheme, and proposed a secure and scalable multi-party searchable encryption scheme [17]. However, there are still three shortcomings in the scheme as follows:

- Firstly, since the authorization in the scheme is granted on the index level, the authorization granularity is also coarse. If u_i wants to authorize u_j to search for a subset of keywords in its indexes, then the scheme cannot complete this task.
- Secondly, the scheme only supports search authorization, but it does not support search authorization revocation explicitly.
- Thirdly, the match algorithm in the construction of the scheme has two pairing map operations, which seriously affect its performance.

1.2. Our contributions

In this work, we study the problem of how to enable a user to share its files with others and authorize them to search its files using a subset of keywords in encrypted form. We propose a multi-user searchable encryption scheme with keyword authorization in a cloud storage (MSESKA), which can be regarded as multi-user version of the symmetric searchable encryption proposed by Song et al. [2]. Briefly, our MSESKA allows every user to build an encrypted index for each of its files and store it on a cloud storage server. The index contains a list of encrypted keywords which are well organized, and some authorization information selectively authorizes other users to search for a subset of keywords in the index. Our contribution can be summarized as follows:

- Firstly, we define a formal security model of how to authorize a user to search for a word over an encrypted file in a cloud storage. In particular, our definition captures a strong noting of security, which is adaptive security against chosen-keyword attacks.
- Secondly, we propose a multi-user searchable encryption scheme with keyword authorization in a cloud storage, which supports keyword authorization revocation explicitly. Our scheme overcomes shortcomings in the Tang scheme [17].
- Thirdly, we propose a KABtree, and use it to organize the index in our scheme. If there exist n users and m keywords in a KABtree, then the construction time of the KABtree is $O(mn)$, the authorization or revocation time for a keyword of the KABtree is $O(m \log n)$, and the search time for a keyword of the KABtree is $O(r \log n)$, where r is the number of users that have been authorized to the keyword.
- Fourthly, we construct the MSESKA using asymmetric bilinear map groups of Type-3 [18] and prove that the construction is secure in the random oracle model under the BDHV and SXDH assumptions.
- Fifthly, we implement our scheme and conduct performance evaluation. The results show that our scheme is very efficient and practical.

The paper is organized as follows. In Section 2, we present the preliminary knowledge. In Section 3, we describe and give some definitions about the problem and define a multi-user searchable encryption scheme with keyword authorization in a cloud storage. The keyword authorization binary tree is presented in Section 4. In Section 5, we give a construction for MSESKA using Type-3 pairings and KABtree. In Section 6, we give the security and performance analysis for our construction. In Section 7, we implement our scheme and conduct a performance evaluation and performance evaluation results are presented here. In Section 8, we discuss related works. Finally, some conclusions are given in Section 9.

2. Preliminary

2.1. Notations

In this paper, we use the following notation. k is the security parameter. $p.p.t.$ denotes probabilistic polynomial time, $x \parallel y$ denotes the concatenation of x and y, and $y \xleftarrow{\$} A(x_1, x_2, \ldots, x_n; o_1, o_2, \ldots, o_n)$ denotes that y is the output of the algorithm A which runs with the input $x_1, x_2, \ldots, x_n$ and access to oracles $o_1, o_2, \ldots, o_n$. When X is a finite set, we use $x \in_R X$ to denote that x is chosen from X uniformly at random, and use $|X|$ to denote the size of X. We say that a function f is negligible in a parameter k, if for every polynomial $p(k)$, there exists an integer K such that for all $k > K, f(k) < \frac{1}{p(k)}$. For simplicity, we write $f(k) = negl(k)$. If $f(k)$ is negligible, then we say $1 - f(k)$ is overwhelming.

2.2. Assumptions

We use asymmetric bilinear map groups of Type-3 [18] for our construction. $Setup(k)$ is a bilinear group generator that takes a security parameter k as input, and outputs curve parameters $params = (p, G_1, G_2, G_T, e, g_1, g_2, g_T)$ where:

- G_1, G_2 and G_T are three disjoint cyclic subgroups on an elliptic curve of Type-3.
- g_1, g_2 and g_T are generators of G_1, G_2 and G_T.
- e is an efficiently-computable bilinear pairing map $e : G_1 \times G_2 \to G_T$ that satisfies two properties:
  (1) Bilinearity: for all $a, b \in Z_p$, $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$.
  (2) No-degeneracy: $e(g_1, g_2) \neq 1$.

Definition 1 (*Bilinear Diffie–Hellman Variant (BDHV) Assumption [16]*)**.** Given $T = (p, G_1, G_2, G_T, e, g_1, g_2)$, for all p.p.t. adversary A, for every sufficiently large security parameter k and $a, b, c \in_R Z_p$ and $R \in_R G_T$, A's advantage $\varepsilon_{BDHV} = |Pr[A(T, g_1^a, g_2^{\frac{1}{a}}, g_2^{\frac{b}{a}}, g_1^c, e(g_1, g_2)^{bc}) = 1] - Pr[A(T, g_1^a, g_2^{\frac{1}{a}}, g_2^{\frac{b}{a}}, g_1^c, R)]| = negl(k)$.

Definition 2 (*Symmetric EXternal Diffie–Hellman (SXDH) Assumption [19]*)**.** Given $T = (p, G_1, G_2, G_T, e, g_1, g_2)$, for all p.p.t. adversary A, for every sufficiently large security parameter k and $a, b, c, d, r_1, r_2 \in_R Z_p$, A's advantage $\varepsilon_{SXDH} = max(\varepsilon_1, \varepsilon_2) = negl(k)$, where $\varepsilon_1 = |Pr[A(T, g_1^a, g_1^b, g_1^{ab}) = 1] - Pr[A(T, g_1^a, g_1^b, g_1^{r_1}) = 1]|$, and $\varepsilon_2 = |Pr[A(T, g_2^c, g_2^d, g_2^{cd}) = 1] - Pr[A(T, g_2^c, g_2^d, g_2^{r_2}) = 1]|$.

Definition 3 (*n-Parallel Decisional Diffie–Hellman ($PDDH_n$) Assumption [20]*)**.** Given $T = (p, G_1, G_2, G_T, e, g_1, g_2)$, for all p.p.t.

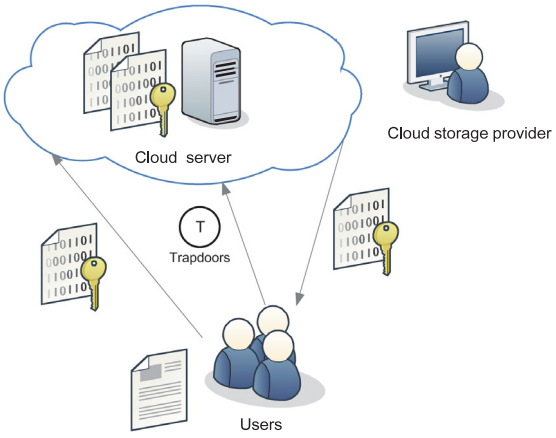

Fig. 1. The system model.

adversary A, for every sufficiently large parameter k and $a_1, a_2, a_3, \ldots, a_n \in_R Z_p$ and $R_1, R_2, \ldots, R_n \in_R G_1$, A's advantage $\varepsilon_{PDDH_n} = |Pr[A(T, g_1^{a_1}, g_1^{a_2}, \ldots, g_1^{a_n}, g_1^{a_1a_2}, g_1^{a_2a_3}, \ldots, g_1^{a_na_1}) = 1] - Pr[A(T, g_1^{a_1}, g_1^{a_2}, \ldots, g_1^{a_n}, R_1, R_2, \ldots, R_n) = 1]| = negl(k)$.

It has been proven that $PDDH_n$ assumption is equivalent to DDH assumption in G_1, namely $\varepsilon_{DDH} \leq \varepsilon_{PDDH_n} \leq n\varepsilon_{DDH} \leq n\varepsilon_{SXDH}$. In the same way, $PDDH_n$ assumption can be defined in G_2.

3. Problem and definition

3.1. The system model

The system model is described in Fig. 1. It consists of two kinds of entities: a cloud server and a set of users u_i $(1 \leq i \leq N)$. N is an integer, which is a polynomial in the security parameter k. Each entity has different responsibility respectively. Each user can encrypt its files and generate encrypted indexes for its files, and give other users access to its files by giving them the decryption keys. All encrypted indexes and encrypted files are stored on the cloud server which owned by some cloud storage provider. When a user initiates a trapdoor to the cloud server to search for a word, the cloud server searches for it over all encrypted indexes on the user's behalf. If the server finds some encrypted files that match the trapdoor, it will return these encrypted files to the user, and the user restores these encrypted files with the decryption keys that are given by the file owners. In the following sections, we use cloud storage provider and cloud server interchangeably.

3.2. Problem formalization

Since the cloud storage is entirely distributed, there is no trusted third party for a user to choose keys or to help with providing access to files. Each user generates its file keys and gives other users access to its files. It builds an encrypted index for each of its files. The index contains a list of encrypted keywords which are well organized, and some authorization information selectively authorizes other users to search for a subset of keywords in the index. All indexes and files on the cloud server are encrypted by different keys, which are owned by different users. If a user wants to search for a keyword in all the files that it can access, it should only give one search trapdoor to the cloud server. We can formalize the problem as follows:

When a user, say u_1, wants to store n files at the cloud server, u_1 generates (MPK_1, MSK_1), which is its master public/private key pair. For each of its file, u_1 first extracts $(w_0, w_1, \ldots, w_{M-1})$ which is a list of keywords, then uses $(w_0, w_1, \ldots, w_{M-1})$, MSK_1 and MPK_i $(1 \leq i \leq N)$ to generate an encrypted index, and finally outsources the index to the cloud storage server. The encrypted index contains a list of ciphertexts for $(w_0, w_1, \ldots, w_{M-1})$ and a list of authorization tokens $(au_{1,1,0}, au_{1,1,1}, \ldots, au_{1,N,M-1})$. The list of authorization tokens indicates who can search the index. To search for a specific keyword w, a user, say u_2 issues a trapdoor $Trap_w$ to the cloud server, generated based on w and MSK_2, where MSK_2 is his master private key. With $Trap_w$ from u_2, the cloud storage server first selects the encrypted indexes whose attached authorization information indicates that u_2 is authorized to search for w. Then, for each selected index, the cloud server runs a match algorithm to decide whether the index contains the same keyword w as that in $Trap_w$. If u_1 want to revoke u_2 the keyword authorization of w, it uses MSK_1, w and MPK_2 to generate $AUR_{1,2,w}$, where $AUR_{1,2,w}$ is an authorization revocation indication. According to $AUR_{1,2,w}$, the cloud storage server removes $au_{1,2,w}$ from u_1's every encrypted index.

3.3. The MSESKA scheme

In order to solve the problem that we have formalized in Section 3.2, we design a multi-user searchable encryption scheme with keyword authorization in a cloud storage.

Definition 4 (*MSESKA*)**.** A multi-user searchable encryption scheme with keyword authorization in a cloud storage is a tuple of algorithms as follows:

- Setup(k)$\rightarrow$ *params*: It takes the security parameter k as input, and outputs the public parameter *params*.
- KeyGen(*params*)$\rightarrow$ (MPK_i, MSK_i): This algorithm is run by u_i $(1 \leq i \leq N)$, it takes the system parameters *params* as input and outputs a master public/private key pair (MPK_i, MSK_i).
- AuthToken$(MSK_i, MPK_j, w_t) \rightarrow au_{i,j,t}$: This algorithm is run by u_i, it takes MSK_i, MPK_j, and w_t as input and generates an authorization token $au_{i,j,t}$ as output. The token $au_{i,j,t}$ denotes u_j can search for the word w_t over u_i's encrypted files. We use $AU_{i,j}$ to denote the authorization token set that u_j can use to search over the u_i's encrypted files, where $AU_{i,j} = (au_{i,j,0}, au_{i,j,1}, \ldots, au_{i,j,M-1})$, and use AU_i to denote the authorization token set generated by u_i, where $AU_i = (AU_{i,1}, AU_{i,2}, \ldots, AU_{i,N})$.
- $BuildIndex(MSK_i, W, AU_i, f) \rightarrow (Index_{id_f}, FK_{id_f})$: This algorithm is run by u_i, it takes MSK_i, W, AU_i and f as input and outputs an index file $Index_{id_f}$ and a file-specific secret key FK_{id_f}.f is a file owned by u_i, $W = (w_0, w_1, \ldots, w_{M-1})$ is a keyword set where no keyword should repeat, and AU_i is the authorization token set that u_i authorizes u_j to search its file f.
- $Enc(FK_{id_f}, w) \rightarrow c$: This algorithm is run by u_i, it takes FK_{id_f} and w as input and outputs c, where c is an encryption of the keyword w.
- $Trapdoor(MSK_j, w) \rightarrow Trap_w$: This algorithm is run by u_j, it outputs a trapdoor $Trap_w$ for the keyword w.
- $Match(Trap_w, Index_{id_f}) \rightarrow b$: This algorithm is run by the cloud server, it takes $trap_w$ and $Index_{id_f}$ as input and outputs a bit b.
- $AuthRevoke(MSK_i, MPK_j, w) \rightarrow AUR_{i,j,w}$: This algorithm is run by u_i, it takes MSK_i, MPK_j and w as input and outputs an authorization revocation token $AUR_{i,j,w}$.
- $Revoke(Index_{id_f}, AUR_{i,j,w}) \rightarrow Index'_{id_f}$: This algorithm is run by the server, it takes an encrypted index $Index_{id_f}$ and an authorization revocation token $AUR_{i,j,w}$ as input and outputs a new encrypted index $Index'_{id_f}$.

Definition 5 (*Soundness of MSESKA*)**.** For any polynomial n, for every sufficiently large security parameters k, for all $w_0 \neq w_1 \in_R \{0, 1\}^{n(k)}$, there exists a negligible function *negl* such that

$$Pr\left[\begin{array}{l} Setup(k) \rightarrow \text{ params};\\ KeyGen(params) \rightarrow (MPK_i, MSK_i);\\ AuthToken(MSK_i, MPK_j, w_0) \rightarrow au_{i,j,0};\\ Buildindex(MSK_i, W, AU_i, f) \rightarrow (Index_{id_f}, FK_{id_f}),\\ where\ w_0 \in W, au_{i,j,0} \in AU_i\ and\ au_{i,j,1} \notin AU_i;\\ Trapdoor(MSK_j, w_0) \rightarrow Trap_{w_0};\\ Trapdoor(MSK_j, w_1) \rightarrow Trap_{w_1};\\ Match(Trap_{w_0}, Index_{id_f}) = 1\ and\\ Match(Trap_{w_1}, Index_{id_f}) = 0 \end{array}\right] = 1 - negl(k).$$

The soundness of MSESKA says when searching for the keyword w_0, that has been authorized to search, the encryption of w_0 in some files will match, but if not having been authorized to search a different keyword w_1, then the encryptions of w_1 will not match the search.

3.4. Security model for MSESKA

Intuitively, we require two security properties for the MSESKA scheme: the ciphertext and the trapdoor should not reveal the value of the underlying plaintext, and the only information revealed to the server is whether a search trapdoor matches a ciphertext only when the server determines whether one is searching for the same word as before. We formalize these properties with Data hiding game and Trapdoor hiding game, that express these goals. In Data hiding game and Trapdoor hiding game, we assume u_1 to be the challenger that is honest, and use the adversary to play the role of untrusted cloud storage provider or other malicious users u_i $(2 \leq i \leq N)$.

3.4.1. Data hiding

Intuitively, the data hiding property say that, if u_1 has not authorized a keyword in an index to any other user, then the attacker will not learn anything about the keyword in this index unless u_1 reveals it by issuing the relevant trapdoor.

In Data hiding game, we define data hiding property formally through a standard challenger–attacker attack game. The malicious adversary tries to distinguish between ciphertexts of two values that do not matched by some trapdoor. Data hiding game has six phases: Setup, QueryToken, QueryIndex, Challenge, Adaptive step and Guest.

Data hiding game.

- Setup: First, the challenger runs the setup function of the MSESKA and gets the public parameter *params*, then runs the KeyGen function of the MSESKA and gets (MPK_1, MSK_1), and sends *params* and MPK_1 to the adversary.
- QueryToken: The adversary generates (MPK_i, MSK_i) $(2 \leq i \leq N)$ and w_t $(0 \leq t \leq M-1)$, and sends MPK_i $(2 \leq i \leq N)$ and w_t $(0 \leq t \leq M-1)$ to the challenger to query token $au_{1,i,t}$. The challenger runs the authorize function and gets $au_{1,i,t}$ and sends $au_{1,i,t}$ to the adversary.
- QueryIndex: The adversary selects a file f randomly, and sends f and AU_1 to the challenger to query its index. The challenger runs the buildindex function of the MSESKA to generate an $Index_{id_f}$, and sends $Index_{id_f}$ to the adversary.
- Challenge: The adversary chooses w_0, w_1 randomly, and provides w_0, w_1 to the challenger. The challenger chooses a random bit b and provides $MSESKA.Enc(FK_{id_f}, w_b)$ to the adversary.
- Adaptive step: The adversary makes the following queries to the challenger adaptively. the lth query can be:
  (1) "Encrypt w_l $(w_l \neq w_0, w_1)$ to f": the challenger returns $MSESKA.Enc(FK_{id_f}, w_l)$.
  (2) "Trapdoor for word w_l for user i": the challenger returns $MSESKA.Trapdoor(MSK_i, w_l)$.
- Guest: The adversary outputs b', its guess for b.

We say that if $b = b'$ then the adversary wins the Data hiding game.

Definition 6. If for any *p.p.t.* adversary, for all sufficiently large k, the probability that the adversary wins the Data hiding game satisfies (1), then we say our *MSESKA* is data hiding.

$$Pr[win_{Datahidinggame}(k)] < \frac{1}{2} + negl(k). \tag{1}$$

3.4.2. Trapdoor hiding

Intuitively, Trapdoor hiding requires that an adversary cannot learn the keyword that one searches for. Namely, if u_1 does not authorize any other user, then the attacker will not learn anything about the keywords in the u_1's trapdoors unless these trapdoors match an index that has shared with the attacker.

In Trapdoor hiding game, the malicious adversary try to play a game with challenger to distinguish a trapdoor of a designated keyword from some other trapdoors. If he can accomplish this task, then he can get some useful information from trapdoor set. Trapdoor hiding game has six phases: Setup, QueryToken, QueryIndex, Challenge, Adaptive step and Guest.

Trapdoor hiding game.

- Setup: First, the challenger runs the setup function of the MSESKA, and gets the public parameter *params*, then runs the KeyGen function of the MSESKA and gets (MPK_1, MSK_1), and sends *params* and MPK_1 to the adversary.
- QueryToken: The adversary generates (MPK_i, MSK_i) $(2 \leq i \leq N)$ and w_t $(0 \leq t \leq M-1)$, and sends MPK_i $(2 \leq i \leq N)$ and w_t $(0 \leq t \leq M-1)$ to the challenger to query token $au_{1,i,t}$. The challenger runs the authorize function and gets $au_{1,i,t}$ and sends $au_{1,i,t}$ to the adversary.
- QueryIndex: The adversary selects a file f randomly, and sends f and AU_1 to the challenger to query an index. The challenger runs the buildindex function of the MSESKA to generate an $Index_{id_f}$, and sends $Index_{id_f}$ to the adversary.
- Challenge: The adversary chooses w_0, w_1 randomly, and provides w_0, w_1 to the challenger. The challenger chooses a random bit b and provides $Trapdoor(MSK_1, w_b)$ to the adversary.
- Adaptive step: The adversary makes the following queries to the challenger adaptively. The lth query can be:
  (1) "Encrypt w_l $(w_l \neq w_0, w_1)$ to f": the challenger returns $MSESKA.Enc(FK_{id_f}, w_l)$.
  (2) "Trapdoor for word w_l for user i": the challenger returns $MSESKA.Trapdoor(MSK_i, w_l)$.
- Guest: The adversary outputs b', its guess for b.

We say that if $b = b'$ then the adversary wins Trapdoor hiding game.

Definition 7. If for any *p.p.t.* adversary, for all sufficiently large k, the probability that wins the Trapdoor hiding game satisfies (2), then we say our *MSESKA* is trapdoor hiding.

$$Pr[win_{Trapdoorhidinggame}(k)] < \frac{1}{2} + negl(k). \tag{2}$$

4. Keyword authorization binary tree

To make our scheme revoke authorization conveniently, we will use some data structure to organize keyword authorization. Let $W = (w_0, w_1, \ldots, w_{M-1})$ be a set of keywords for f_i. We

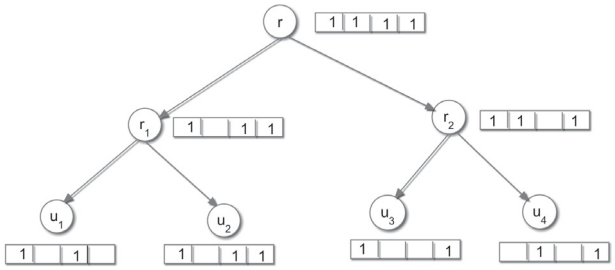

Fig. 2. An example of a KABtree.

view each individual file f_i as a bit-string of polynomial length, i.e., $f_i = \{0,1\}^{poly(k)}$. Let $U = (u_1, u_2, \ldots, u_N)$ be a sequence of users with corresponding identifiers $id_u = (1, 2, \ldots, N)$. We assume that the universe of keywords is fixed but the users can grow. We design a data structure for keyword authorization which refers to as a keyword authorization binary tree (KABtree) (see Fig. 2). The intuitive reason for the KABtree is that we want to allow both keyword-based operations (by following paths from the root to the leaves) and user-based operations (by following paths from the leaves to the root). As we will see later, this property is useful for handling authorization revocation efficiently.

4.1. Definition

Definition 8. A KABtree is a binary tree with the difference that its nodes store an *M-bit* vector *data* to indicate the keyword authorization relation, and its leaves store the user identifiers as well. Its nodes satisfy the following properties:

- For every leaf l storing identifier j, we set $data_l[i] = 1$ if and only if user u_j can search keyword w_i.
- At each internal node s of the tree T, it stores an *M-bit* vector $data_s$. The ith bit of $data_s$ accounts for keyword w_i. Specifically, if $data_s[i] = 1$, then there is at least one path from s to some leaf that stores some identifier j, such that u_j can search w_i.
- Let s be an internal node of the tree T with left child v and right child z. The vector $data_s$ of the internal node s is computed recursively according to Eq. (3).

$$data_s = data_v + data_z \tag{3}$$

where $+$ denotes the bitwise Boolean *OR* operation.

Assume $U = (u_1, \ldots, u_N)$ is a set users, $id_u = (1, \ldots, N)$ is a set of user identifiers, $W = (w_0, w_1, \ldots, w_{M-1})$ is a keyword set of keywords, and $Tau = (Tau_{1,0}; \ldots; Tau_{N,M-1})$ is a set of keyword authorization indications. To construct a KABtree, we can use the following procedure, which we denote as BuildKABtree(U, W, Tau):

(1) Build a binary tree T on top of $(1, \ldots, N)$. At each leaf l, store an *M-bit* vector $data_l$. The jth bit of $data_l$ accounts for keyword w_j, for $j = 0, 1, \ldots, M-1$. If $Tau_{i,j} = 1$ then set $data_l[j] = 1$, otherwise set $data_l[j] = 0$.
(2) At each internal node s of the tree, store an *M-bit* vector $data_s$. The ith bit of $data_s$ accounts for keyword w_i, for $i = 0, 1, \ldots, M-1$. If $data_s[i] = 1$, then there is at least one path from s to some leaf that stores some identifier j, such that user u_j can search w_i.
(3) Assume s be an internal node of T with left child v and right child z. The vector $data_s$ of the internal node s is computed recursively according to Eq. (3).

To search these authorization users for w in the KABtree T, assume that i is the position in the *M-bit* vectors to account for w at each node, we can use the following procedure, which we denote as SearchKABtree(T, i):

Check the bit at position i of node v and examine v's children if the bit at the same position is 1. When this traversal is over, return these identifiers in all the leaves that were reached.

To revoke the keyword authorization of w for the user u_j in a KABtree T, we can use the following procedure, denoted as RevokeFromKABtree(T, i, u_j), where i is the position in the *M-bit* vectors to account for w at the nodes of T:

(1) Traverse T to reach the leaf l that stores the identity of the user u_j, and set its $data_l[i] = 0$.
(2) Traverse T from the leaf l to the *root*, compute all the *M-bit* vector in these internal nodes recursively according to Eq. (3).

Lemma 1. *Assume $U = (u_1, \ldots, u_n)$ be a set of n users that are authorized to search m keywords $W = (w_0, w_1, \ldots, w_{m-1})$. Then there exists a KABtree for keyword authorization organization such that:* (1) *the space complexity of the KABtree is $O(mn)$;* (2) *the constructing time of the KABtree is $O(mn)$;* (3) *the search time for a keyword w ($w \in W$) is $O(r \log n)$, where r is the number of users that is authorized to search w;* (4) *the time to authorize or revoke a keyword authorization of a user u ($u \in U$) is $O(m \log n)$.*

Proof. The space complexity of the underlying binary tree in a KABtree is $O(n)$ and there exists a bit-vector of m bits at each node, we can get that the space complexity of the KABtree is $O(mn)$.

According to Eq. (3), we can build the KABtree on the user collection U by following a postorder traversal. As the time to compute the *OR* of two *m-bit* vectors is $O(m)$ and a postorder traversal visits $O(n)$ nodes, we can get that the time for KABtree is $O(mn)$.

According to the SearchKABtree procedure, we know that search for a keyword w, which corresponds to position i of the *m-bit* vectors, proceeds as follows: while the bit at position i of node s is 1, examine s's children. Therefore the search procedure will traverse as many paths as the users can search keyword w, namely r paths. Since the maximum height of the binary tree is maintained to be $O(\log n)$, the search time is $O(r \log n)$.

Since it uses the *m-bit* vector in each node to represent the keyword authorization relation, when we authorize or revoke the authorization of w to a user u. It only sets or unsets the bit in the *m-bit* vector that accounts for w on the leaf representing u. Since the maximum height of the binary tree is maintained to be $O(\log n)$, the traversal time from the root to the leaf l that stores the identity of the user u is $O(\log n)$ and the time spent in each node is $O(m)$, the time to authorize or revoke a keyword authorization of u is $O(m \log n)$. ■

4.2. An illustrative example

In Fig. 2, we use a KABtree to organize keyword authorization for file f. The KABtree is built on four users, namely u_1, u_2, u_3, u_4 over four keywords, namely w_1, w_2, w_3, w_4. All the nodes in the KABtree store a 4-*bit* vector *data* to indicate the keyword authorization. For example, the vector value on the leaf u_2 is 1011, which indicates that u_2 can search three keywords in F, namely w_1, w_3, w_4.

(1) **Searching**. Searching is the main operation in the KABtree. Suppose the server wants to determine which users are authorized to search for w_3. It first gets the position of w_3 in the 4-*bit* vector. In our example, this position is 3. After completing that, it checks $data_r[3]$ in root r and finds that this bit is 1. As this bit is 1, it continues to visit the children of r and examines their 4-*bit* vectors. Because $data_{r_2}[3]$ is 0, it does not examine the children of node r_2. But $data_{r_1}[3]$ is 1, so it keeps on examining the children of r_1, and finds that $data_{u_1}[3] = 1$ and $data_{u_2}[3] = 1$. Since node u_1 and u_2 are leaves, it can determine that u_1 and u_2 are users that are authorized to search for w_3, and returns u_1 and u_2.

(2) **Revoking an authorization**. Suppose the server wishes to revoke the keyword authorization of w_1 for u_3, it traverses T to reach the leaf u_3 and sets $data_{u_3}[1] = 0$. Then it computes $data_{r_2}$ and $data_r$ according to Eq. (3) in turn. As a result, it sets $data_{r_2}[1] = 0$ and $data_r[1] = 1$.

5. A construction for MSESKA

Let $H_1 : \{0,1\}^* \to G_1$ and $H_2 : \{0,1\}^* \to G_2$ be hash functions, which are modeled as random oracles. Let k be the security parameter, our construction for MSESKA is as follows:

- Setup(k)$\to$ $params$: It takes k as input and outputs $params$, where $params = (p, G_1, G_2, G_T, e, g_1, g_2, g_T)$.
- KeyGen(params)$\to$ (MPK_i, MSK_i): It takes $params$ as input and outputs a master public/private key pair (MPK_i, MSK_i), where $MPK_i = g_2^{\frac{1}{x_i}}$ and $MSK_i = x_i$ for $x_i \in_R Z_p$.
- AuthToken$(MSK_i, MPK_j, w_t) \to au_{i,j,t}$: It inputs MSK_i and MPK_j and returns $au_{i,j,t} = (MPK_i, MPK_j, i, j, H_2(g_2^{\frac{x_i}{x_j}} \parallel i \parallel j \parallel w_t).g_2^{x_i})$. It uses $AU_{i,j}$ to denote the authorization tokens that u_j can search the word w_t in the encrypted files owned by u_i where $AU_{i,j} = (au_{i,j,0}, au_{i,j,1}, \ldots, au_{i,j,M-1})$ and uses $AU_i = (AU_{i,1}, AU_{i,2}, \ldots, AU_{i,N})$ to denote the set of authorization tokens generated by u_i.
- $Buildindex(MSK_i, W, AU_i, f) \to Index_{id_f}$: It parses MSK_i as x_i, W as $(w_0, w_1, \ldots, w_{M-1})$ and AU_i as $(au_{i,1,0}, au_{i,1,1}, \ldots, au_{i,N,M-1})$ and outputs an index file $Index_{id_f}$ owned by u_i. The index file $Index_{id_f}$ is built as follows:
  (1) For the file f, select a unique index identifier $id_f \in_R \{0,1\}^k$, and generate a file-specific secret key $FK_{id_f} = k_1$ where $k_1 \in_R Z_p$.
  (2) Generate $TAG_{id_f} = (Enc(FK_{id_f}, w_0), Enc(FK_{id_f}, w_1), \ldots, Enc(FK_{id_f}, w_{M-1}))$, where TAG_{id_f} is a set of the ciphertexts.
  (3) Build a KABTree T_f as follows:
    (1) For every $au_{i,j,t} \in AU_i$, parse it as $(MPK_i, MPK_j, i, j, H_2(g_2^{\frac{x_i}{x_j}} \parallel i \parallel j \parallel w_t).g_2^{x_i})$, set $TU_j = MPK_j$ and $Tau_{j,t} = 1$. It generates $TU = (TU_1, TU_2, \ldots, TU_n)$ and $Tau = (Tau_{1,0}, Tau_{1,1}, \ldots, Tau_{n,M-1})$.
    (2) Call BuildKABtree(TU, TAG_{id_f}, Tau) to output T_f.
  (4) For every $au_{i,j,t} \in AU_i$, generate $\Theta_{MPK_j} = g_2^{\frac{k_1}{x_j}}$.
  (5) For u_i, generate $\Theta_{MPK_i} = g_2^{\frac{k_1}{x_i}}$ for himself. It can use Θ_{MPK_i} to search f or to revoke the keyword authorization of other users from f.
  (6) Set $\Delta_{id_f} = (\Theta_{MPK_1}, \ldots, \Theta_{MPK_N})$.
  (7) Return $Index_{id_f} = (id_f, TAG_{id_f}, \Delta_{id_f}, T_f)$.
- $Enc(Fk_{id_f}, w) \to c$: It parses FK_{id_f} as k_1 and returns c, where $c = e(H_1(w), g_2)^{k_1}$.
- $Trapdoor(MSK_j, w) \to Trap_w$: It parses MSK_j as x_j and returns $Trap_w = (MPK_j, H_1(w)^{x_j})$.
- $Match(Trap_w, Index_{id_f}) \to b$: It parses $Trap_w$ as (α, β) and parses $Index_{id_f}$ as $(id_f, (Enc(FK_{id_f}, w_0), \ldots, Enc(FK_{id_f}, w_{M-1})), \Delta_{id_f}, T_f)$ and proceeds as follows:
  (1) If $\Theta_\alpha \notin \Delta_{id_f}$ return 0.
  (2) If $Test(Enc(FK_{id_f}, w_t), \beta) = 1$ for some $0 \le t \le M-1$, set $pos = t$, otherwise return 0. Let $Test(Enc(Fk_{id_f}, w_t), \beta) = 1$ iff $c = e(\beta, \Theta_\alpha)$.
  (3) Call SearchKABtree(T_f, pos) to output U_{ID} which is the set of users that can search w in f.
  (4) If $\alpha \in U_{ID}$ return 1, otherwise return 0.
- $AuthRevoke(MSK_i, MPK_j, w) \to AUR_{i,j,w}$: It parses MSK_i as x_i and MPK_j as $g_2^{x_j}$ and returns $AUR_{i,j,w}$ where $AUR_{i,j,w} = (MPk_i, MPK_j, i, j, H_2(w)^{x_i}, g_2^{x_i}, H_2(g_2^{\frac{x_i}{x_j}} \parallel i \parallel j \parallel w))$.
- $Revoke(Index_{id_f}, AUR_{i,j,w}) \to Index'_{id_f}$: It takes $Index_{id_f}$ and $AUR_{i,j,w}$ as input and outputs a new encrypted index $Index'_{id_f}$. It parses $AUR_{i,j,w}$ as $(AU_{d_1}, AU_{d_2}, AU_{d_3}, AU_{d_4}, AU_{d_5}, AU_{d_6}, AU_{d_7})$ and proceeds as follows:
  (1) If $AU_{d_6} * AU_{d_7} \neq H_2(g_2^{\frac{x_i}{x_j}} \parallel i \parallel j \parallel w).g_2^{x_i}$, abort.
  (2) If $Test(Enc(FK_{id_f}, w_t), AU_{d_5}) = 1$ for some $0 \le t \le M-1$, set $pos = t$, otherwise abort. Let $Test(Enc(FK_{id_f}, w_t), AU_{d_5}) = 1$ iff $c = e(AU_{d_5}, \Theta_{AU_{d_1}})$.
  (3) Call $SearchKABtree(T_f, pos)$ to output U_{ID}, which is the set of users that can search for w over f. If $AU_{d_2} \notin U_{ID}$, abort.
  (4) Call RevokeFromKABtree(T_f, pos, AU_{d_2}) to revoke u_j the keyword authorization of w, which is authorized by u_i and generate a new index $Index'_{id_f}$.

6. Security and performance analysis

In this section, we will show that our scheme is secure and efficient. In Theorem 1, we demonstrate that our scheme is sound. We prove Theorems 2 and 3 by a sequence of hybrid games. In Remark 1, we discuss the problem about the search authorization being exposed to the cloud server, and show that it does not affect the security of our scheme. In Theorem 4, we analyze the performance of our scheme, and show that our scheme is efficient.

6.1. Security analysis

Theorem 1. *The construction of the MSESKA achieves sound under Definition 5.*

Proof. Assume f is a file owned by u_i, w_0 and w_1 are two different keywords which are randomly selected from f. Let $MSK_i = x_i$, $MPK_i = g_2^{\frac{1}{x_i}}$, $MSK_j = x_j$, $MPK_j = g_2^{\frac{1}{x_j}}$ and $FK_{id_f} = k_1$, we can get $\Theta_{MPK_j} = g_2^{\frac{k_1}{x_j}}$, $c = Enc(Fk_{id_f}, w_0) = e(H_1(w_0), g_2)^{k_1}$, $Trapdoor(MSk_j, w_0) = (MPK_j, H_1(w_0)^{x_j})$ and $Trapdoor(MSk_j, w_1) = (MPK_j, H_1(w_1)^{x_j})$.

$$\text{As } e(H_1(w_0)^{x_j}, \Theta_{MPK_j}) = e(H_1(w_0)^{x_j}, g_2^{\frac{k_1}{x_j}}) = e(H_1(w_0), g_2)^{k_1}$$

and $e(H_1(w_1)^{x_j}, \Theta_{MPK_j}) = e(H_1(w_1)^{x_j}, g_2^{\frac{k_1}{x_j}}) = e(H_1(w_1), g_2)^{k_1}$, the equality $e(H_1(w_0)^{x_j}, \Theta_{MPK_j}) = c$ hold with probability 1.

Since H_1 is a secure hash function, we can get $H_1(w_0) \neq H_1(w_1)$ holds with probability 1-$neg(k)$. As a result, the inequality $e(H_1(w_1)^{x_j}, \Theta_{MPK_j}) \neq c$ holds with probability $1 - negl(k)$.

The value of $Buildindex(MSK_i, W, AU_i, f)$ is the same as required in Definition 5. If u_i authorized u_j to search for w_0 and did not authorize u_j to search for w_1 over f, the equalities $Match(Trapdoor(MSk_j, w_0), Index_{id_f}) = 1$ and $Match(Trapdoor(MSk_j, w_1), Index_{id_f}) = 0$ hold with probability $1 - negl(k)$. This completes the proof. ■

Theorem 2. *Under the BDHV and SXDH assumptions in the random oracle model, the construction of MSESKA achieves data hiding property under Definition 6.*

Proof. We will create a sequence of hybrid games as follows:

Game 1: The challenger performs everything in the Data hiding game. Let the adversary's advantage be ε, according to the Data hiding game, we have $\varepsilon = Pr[win_{Datahidinggame}(k)] - \frac{1}{2}$.

Game 2: The challenger performs the same as in Game 1, except for the following. The challenger constructs an empty hash value list for H_1. The list is consisted of two columns, the first column is used to store keyword w and the second column is used to store its hash value $H_1(w)$. When the adversary queries $H_1(w)$ where $w \in \{w_0, w_1\}$, the challenger first checks whether w has been stored in the list. If w is not in the hash value list, the challenger chooses $\gamma \in_R G_1$ as the hash value, appends w and γ in the list and sends γ to the adversary, otherwise the challenger returns the existing value in the list to the adversary. When the adversary queries $H_1(w)$ where $w \notin \{w_0, w_1\}$, the challenger first checks whether w has been stored in the list. If so, it returns the existing hash value, otherwise it chooses $r \in_R Z_p$ and uses g_1^r as the hash value, and appends w and g_1^r in the list. The challenger sends g_1^r to the adversary. This game is identical to Game 1. Let the adversary's advantage be ε_2 in this game, and we have $\varepsilon_2 = \varepsilon_1$.

Game 3: The challenger performs the same as in Game 2, except for the following. It gives $(g_1^x, g_2^{\frac{1}{x}}, g_2^{\frac{k}{x}}, H(w_0), H_1(w_1), e(H_1(w_b), g_2)^k, e(H_1(w_{\bar{b}}), g_2)^k)$ to the adversary and asks it to guess b. With $g_1^x, g_2^{\frac{1}{x}}$ and $g_2^{\frac{k}{x}}$, the adversary can answer Buildindex oracles on its own. Let the adversary's advantage be ε_3 in Game 3. As the adversary has been given the additional information by the challenger, we have $\varepsilon_3 \geq \varepsilon_2$.

Game 4: The challenger performs the same as in Game 3, except for replacing $e(H_1(w_b), g_2)^k$ with $R_0 \in_R G_T$ and $e(H_1(w_{\bar{b}}), g_2)^k$ with $R_1 \in_R G_T$. The challenger gives $(g_1^x, g_2^{\frac{1}{x}}, g_2^{\frac{k}{x}}, H_1(w_0), H_1(w_1), R_0, R_1)$ to the adversary and asks it to guess b. Let the adversary's advantage be ε_4 in Game 4, and we have $|\varepsilon_3 - \varepsilon_4| \leq \varepsilon_{BDHV} + \varepsilon_{SXDH}$. In Game 4, the challenge is independent from b, it is clear that $\varepsilon_4 = 0$.

According to above games, we have $\varepsilon \leq \varepsilon_{BDHV} + \varepsilon_{SXDH}$. ε_{BDHV} and ε_{SXDH} are negligible under the *BDHV* and *SXDH* assumptions, and we can get ε is also negligible. Namely $Pr[win_{Datahidinggame}(k)] < \frac{1}{2} + negl(k)$. This completes the proof. ■

Theorem 3. *Under the SXDH assumption in the random oracle model, the construction of the MSESKA achieves Trapdoor hiding property under Definition 7.*

Proof. We will create a sequence of hybrid games as follows:

Game 1: The challenger faithfully performs everything in Trapdoor hiding game. Let the adversary's advantage be ε, according to the Trapdoor hiding game, we have $\varepsilon = Pr[win_{Trapdoorhidinggame}(k)] - \frac{1}{2}$.

Game 2: The challenger performs the same as in Game 1, except for the following. The challenger constructs an empty hash value list for H_1. The list is consisted of two columns, the first column is used to store keyword w and the second column is used to store its hash value $H_1(w)$. When the adversary queries $H_1(w)$ where $w \in \{w_0, w_1\}$, the challenger first checks whether w have been stored in the list. If w is not in the hash value list, the challenger chooses a value $\gamma \in_R G_1$ as the hash value, appends w and γ in the list and sends γ to the adversary, otherwise the challenger returns the existing value in the list to the adversary. When the adversary queries $H_1(w)$ where $w \notin \{w_0, w_1\}$, the challenger first checks whether w has been stored in the list. If so, it returns the existing hash value, otherwise it chooses $r \in_R Z_p$ and returns g_1^r as the hash value, and appends w and g_1^r in the list. The challenger sends g_1^r to the adversary. This game is identical to Game 1. Let the adversary's advantage be ε_2 in this game, and we have $\varepsilon_2 = \varepsilon_1$.

Game 3: The challenger executes the same as in Game 2, except for the following.

According to the Trapdoor hiding game, we know that u_1 plays the role of the challenger. Let $MPK_1 = g_2^{\frac{1}{x^*}}$ and $MSK_1 = x^*$. The challenger selects $k_1^* \in_R Z_p$ and computes $g_2^{\frac{k_1^*}{x^*}}$; it computes $H_1(w_b)^{x^*}$ and $H_1(w_{\bar{b}})^{x^*}$ when getting w_b and $w_{\bar{b}}$ from the adversary. The challenger answers any buildindex oracle query with the input(W, AU) as follows:

(1) It uses AU to generate TU and Tau.
(2) If $MPK_1 \notin TU$, the challenger rejects the query and lets the adversary answer it by itself.
(3) If $MPK_1 \in TU$ and $w_0, w_1 \notin W$, the challenger performs the following steps:
- Select a unique identifier $id_f \in_R \{0, 1\}^k$ for the index.
- Select $z \in_R Z_p$.
- Suppose $H_1(w) = g_1^r$ for every $w \in W$, then set its ciphertext $e(g_1^{x^*}, g_2^{\frac{k_1^*}{x^*}})^{rz}$. Use all the ciphertexts to generate TAG_{id_f}.
- Set $\Theta_{MPK_1} = g_2^{\frac{k_1^* z}{x^*}}$. Since all MPK_t are released publicly where $MPK_t = g_2^{\frac{1}{x_t}}$, the challenger can use k_1, MPK_t and z to generate Θ_{MPK_t} for all $MPK_t \in TU$, where $\Theta_{MPK_t} = g_2^{\frac{k_1^* z}{x_t}}$, and use all Θ_{MPK_t} to generate Δ_{id_f}.
- Use TU, TAG_{id_f} and Tau to build a KBAtree T.
- Return $Index_{id_f} = (id_f, TAG_{id_f}, \Delta_{id_f}, T)$.

(4) If $MPK_1 \in AU$ and $w_0, w_1 \in W$, do the following.
- Select a unique identifier $id \in_R \{0, 1\}^k$ for the index.
- Select $z \in_R Z_p$.
- If $w_0, w_1 \in W$, let $e(H_1(w_b)^{x^*}, g_2^{\frac{k_1^*}{x^*}})^z$ be its ciphertext. For every $w \in W$ and $w \notin \{w_0, w_1\}$, suppose $H_1(w) = g_1^r$ then set its ciphertext $e(g_1^{x^*}, g_2^{\frac{k^*}{x^*}})^{rz}$. Use all the ciphertexts to generate TAG_{id_f}.
- Set $\Theta_{MPK_1} = g_2^{\frac{k_1^* z}{x^*}}$. Since all MPK_t are released publicly where $MPK_t = g_2^{\frac{1}{x_t}}$, the challenger can use k_1, MPK_t and z to generate Θ_{MPK_t} for all $MPK_t \in TU$, where $\Theta_{MPK_t} = g_2^{\frac{k_1^* z}{x_t}}$, and use all Θ_{MPK_t} to generate Δ_{id_f}.
- Use TU, TAG_{id_f} and Tau to build a KBAtree T.
- Return $Index_{id_f} = (id_f, TAG_{id_f}, \Delta_{id_f}, T)$.

Since the adversary has not been given any additional information by the challenger in Game 3, it is indeed identical to Game 2. Let the adversary's advantage be ε_3 in Game 3, and we have $\varepsilon_3 = \varepsilon_2$.

Since AU is released publicly, it is clear that the challenger only needs $(g_1^{x^*}, H_1(w_0), H_1(w_1), H_1(w_b)^{x^*}, H_1(w_{\bar{b}})^{x^*})$ to faithfully answer any buildindex oracle query from the adversary in Game 3.

Game 4: The challenger performs the same as in Game 3, except for replacing $H_1(w_b)^{x^*}$ with $R_1 \in_R G_1$ and $H_1(w_{\bar{b}})^{x^*}$ with $R_2 \in_R G_1$. Let the adversary's advantage be ε_4 in this game. It is clear that, in Game 3 and Game 4, the adversary can be regarded as a distinguisher for extended 3-PDDH problem in G_1, and $\varepsilon_{PDDH_3} \leq 3\varepsilon_{DDH} \leq 3\varepsilon_{SXDH}$. Therefore, we have $|\varepsilon_3 - \varepsilon_4| \leq 3\varepsilon_{SXDH}$. Since the challenge is independent from b in Game 4, it is clear that $\varepsilon_4 = 0$.

According to above games, we have $\varepsilon \leq 3\varepsilon_{SXDH}$. It is clear that ε_{SXDH} is negligible under the *SXDH* assumption, and we can get ε is negligible. Namely, $Pr[win_{Trapdoorhidinggame}(k)] < \frac{1}{2} + negl(k)$. This completes the proof. ■

Remark 1. In construction of the MSESKA, the cloud server can get the keyword authorization from three kinds of entities, namely KABtree, authorization token and authorization revocation token. Given $w \in W$, its position in *m-bit* vector *data* is exposed to the cloud server due to the KABtree. But the server cannot learn the value of w. If we use the following technology, it will reduce the

ability of the server to deduce the position of w in m-bit vector $data$. When the file owner wants to build the KABtree for the file f, it first generates m dummy keywords, and randomly mixes these m dummy keywords with the m keywords which are abstracted from f, and generates a new keyword set with $2m$ keywords. Then it uses the new keyword set to generate $2m$-bit vector $data'$, and sets all bits in the leaves that account for these dummy keywords to 0, and calls BuildKABtree to generate the KABtree. At the same time, it uses the new keyword set to generate a set of the ciphertexts $TAG_{id_f} = (Enc(FK_{id_f}, w_0), Enc(FK_{id_f}, w_1), \ldots, Enc(FK_{id_f}, w_{2m-1}))$. After that, the cloud server cannot deduce the position of w in the $2m$-bit vector $data'$ in MSESKA, and it does not affect the function of MSESKA. The authorization token $au_{i,j,w} = (MPK_i, MPK_j, i, j, H_2(g_2^{\frac{x_i}{x_j}} \| i \| j \| w).g_2^{x_i})$ and the authorization revocation token $AUR_{i,j,w} = (MPk_i, MPK_j, i, j, H_2(w)^{x_i}, g_2^{x_i}, H_2(g_2^{\frac{x_i}{x_j}} \| i \| j \| w))$. In $au_{i,j,w}$ and $AUR_{i,j,w}$, w is blinded by $g_2^{\frac{x_i}{x_j}}$. Since the private key x_i is kept by u_i securely, the cloud server cannot get any information of w from $H_2(g_2^{\frac{x_i}{x_j}} \| i \| j \| w).g_2^{x_i}$, $H_2(w)^{x_i}$ and $H_2(g_2^{\frac{x_i}{x_j}} \| i \| j \| w)$. Therefore, though keyword authorization in our scheme is exposed to the cloud server, it does not affect the security of the construction of the MSESKA.

6.2. Performance analysis

Theorem 4. *In the construction of the MSESKA, assume n users that are authorized to search for m keywords over a file f, we can get that:* (1) *in the initialization phrase, the client computation complexity is* $O(mn)$*;* (2) *in the search phrase, the client computation complexity is* $O(1)$*, the server computation complexity is* $O(r \log n)$*;* (3) *in the revoking phrase, the client computation complexity is* $O(1)$*, the server computation complexity is* $O(m \log n)$*;* (4) *the client storage complexity is* $O(1)$.

Proof. In the initialization phrase, the client runs the *Authtoken* to compute mn authorization tokens and runs the *buildindex* to generate an index file. In the *buildindex*, the client generates m ciphertexts, builds a KABtree T_f and generates Δ_{id_f} which has n Θs. According to Lemma 1, we know that the time of *buildindex* is $O(mn)$, and as a result, the client computation complexity is $O(mn)$.

In the search phrase, the client runs the *trapdoor* which its time is $O(1)$ and the server runs the *match*. In *match*, it needs search Δ_{id_f} which needs n computation time, and calls the *test* function which needs m computation time, and calls the *SearchKABtree*. According to Lemma 1, we know that the time of *SearchKABtree* is $O(r \log n)$, where r is the number of authorized users. Hence, the computation cost of the *match* in the server is $O(r \log n)$.

In the revoking phrase, the client calls the *authrevoke* to generate the revocation indication, where its computation time is $O(1)$. After receiving the authorization indication from the file owner, the server runs the *test* function which needs m computation time, then it calls the *SearchKABtree* to generate the set of users that can search the keyword, and it calls the *RevokeFromKABtree*. According to Lemma 1, we know that the time of *SearchKABtree* is $O(r \log n)$ and the time of *RevokeFromKABtree* is $O(m \log n)$. Hence, in the revoking phrase, the computation complexity of the server is $O(m \log n)$.

In our construction, the client only stores its private key, therefore its space complexity is $O(1)$. This completes the proof. ■

7. Experiments

In order to evaluate the performance of the MSESKA, we have realized our construction for MSESKA using C and PBC library [21] for implementation of a type-3 curve. We have performed all experiments on an Intel ® Pentium ® Dual E2160 1.8 GHz with 4 GB RAM running Fedora 8.0 with kernel 2.6.23.1. The disk of the experiment machine is Western Digital Caviar SE hard drive that has 320 GB capacity, 7200 rpm with 8 MB cache. All experiments ran single-threaded on the machine and all data represents the mean of 10 executions.

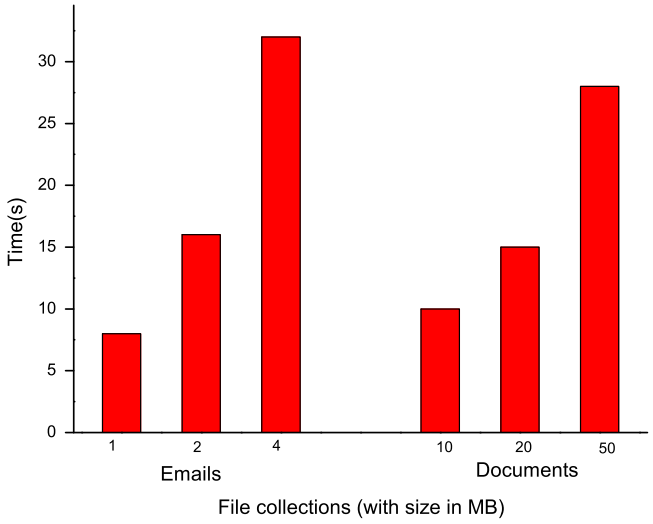

Fig. 3. Execution time for MSESKA.Buildindex.

In our experiments, we chose two sets of real-world data. The first set was selected from the Enron emails [22]; we extracted a subset of emails as file collections. The second set consisted of Microsoft documents (using the Word, PowerPoint, and Excel file types) which produced by our research group. In a similar fashion to the emails, we chose a subset of this collection as smaller file collections. To index the emails and documents, we used an indexer that employs Apache POI to extract unique words from each file. The indexer also extracts properties of the files from the Linux filesystem, such as owner of the file. To evaluate the performance of MSESKA, we ran the MSESKA algorithms specified in Section 5 on the emails and documents.

Fig. 3 shows the results of the Buildindex algorithm, which takes the most time of MSESKA. Note that the Buildindex algorithm is executed by the file owner before his files are sent to the cloud storage. Fig. 3 shows the difference between the documents and the emails. As Microsoft Office documents are a collection of rich text file, which may contain some visual components (such as images), which are not indexed, so each file in documents has few words. However, the emails are a collection of plain text files, which consist of email headers, so almost every byte of every file is part of a word that will be indexed, and each small file contains many words. From Fig. 3, we can see that the index generation in MSESKA requires significantly more time for large text collections than that for the common office documents.

In our MSESKA, the search function is performed by the Match algorithm on the server. Since search was performed for the word that was indexed for the most files, the total needed for the search depended on the prevalence of words in files. Documents had few words, even in 50 MB of content, whereas more words occur in every email. Fig. 4 gives the time needed for the server to perform a match, given a search trapdoor (we neglect the cost of generating a trapdoor, since it is a small constant in microseconds). The match costs in MSESKA is small, even for the email index, the longest search takes only about 20 ms to complete.

Fig. 5 shows the execution time for revoke an authorization for a word over files. The cost of the operation is divided into several components: MSESKA.AuthRevoke refers to client generation of

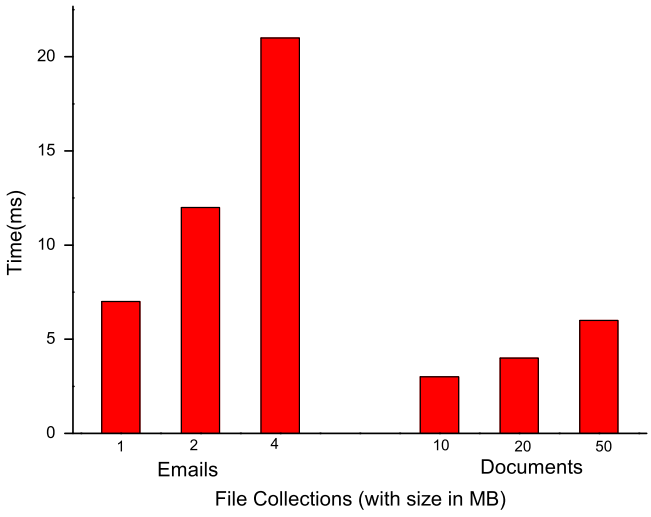

Fig. 4. Execution time for MSESKA.Match.

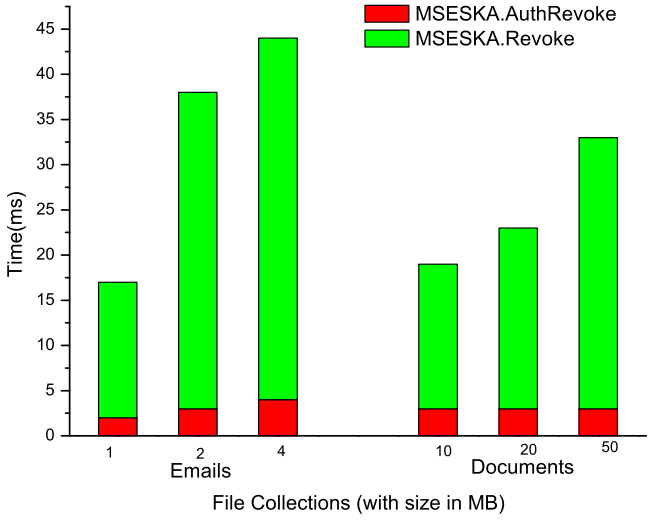

Fig. 5. Execution time for MSESKA.Revoke.

the authorization revocation token, and MSESKA.Revoke refers to the server using the revocation token to update the index. From Fig. 5, we know that the cost on the server is low, this allows the server to support many clients easily.

As there exist only two practical multi-user searchable encryption schemes that support search on files encrypted with different keys in the literature. We refer to them as MSES-Poa [16] and MSES-Tang [17] respectively, we perform the search experiments of our MSESKA against MSES-Poa and MSES-Tang. When perform a search, MSES-Poa adjusts the trapdoor first and then runs match to do the search, but MSESKA and MSES-Tang do not need to adjust the trapdoor, they run match directly. From Fig. 6, we can see that the search performance of MSESKA outperforms those of other schemes. This is because MSES-Poa needs to run adjust and match algorithm, MSES-Tang needs to perform two test operations which are pairing operation, whereas our MSESKA only performs one pairing operation and one search on a KABtree, as a result, the server can search for a word quickly using our MSESKA.

8. Related works

There exist many works on the searchable encryption in the literature. Song et al. proposed the first searchable encryption

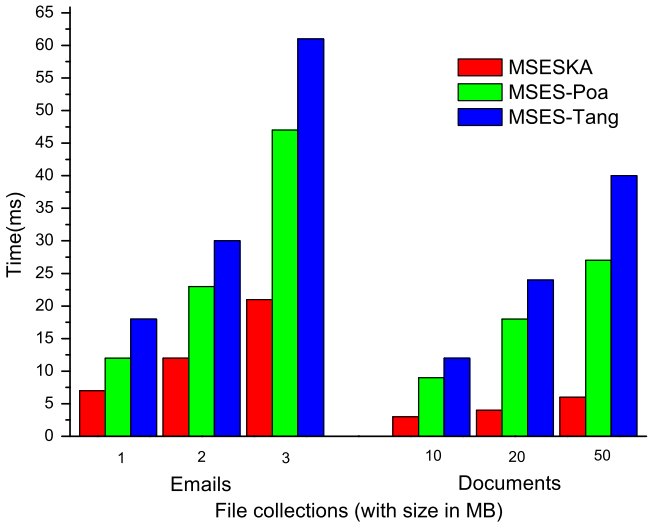

Fig. 6. Search performance comparison.

scheme in the symmetric setting [2]. Subsequently, many researchers have proposed some other schemes with strong security properties or some other properties for different application fields. Goh formally formulated a security model known as semantic security against adaptive chosen keyword attack (IND-CKA), and developed an efficient IND-CKA secure index construction by using pseudo-random functions and Bloom filters [3]. Chang et al. proposed some efficient searchable encryption schemes [5]. Curtmola et al. showed that the indistinguishability and simulation-based definitions for both indexes and trapdoors were equivalent and proposed some searchable encryption schemes [4]. Kamara et al. proposed a dynamic searchable encryption scheme using inverted index [6]. They assume one user and one server in these schemes, where the user can generate searchable indexes and store them at the server, and later entrust the server to search on his behalf, and we cannot use these schemes to address our problem. Goldreich et al. studied software protection based on oblivious RAM, we can realize a searchable encryption using his work, however, the overhead of the oblivious RAMS is too big to be applied to cloud storage [23].

There also exist some searchable encryption schemes for multi-user setting. Curtmola et al. proposed the first multi-user searchable encryption scheme, where a user can be authorized to search the files of others [4]. Bao et al. introduced a user manager to manage the search capabilities of multiple users [10]. As the user manager can submit search queries and decrypting encrypted data, it needs to be fully trusted. However, there usually exists no trusted third party in a cloud storage, so we cannot apply the scheme to the cloud storage. There exist similar problems in the schemes of Dong, Zhao and Yang [11,12,14].

López-Alt et al. investigated the computation over data encrypted with different key [24]. They designed a fully homomorphic encryption (FHE) scheme in which anyone can evaluate a function over data encrypted with different keys. However, their scheme is not practical due to its low efficiency. Popa et al. proposed a multi-user searchable encryption scheme which is the first practical work that addresses a similar problem to ours [16]. However the granularity of authorization in his scheme is very coarse, and they did not explicitly specify how a user u_i can authorize another user u_j to search his index, where $i \neq j$. Later, Tang put forward a secure and scalable multi-party searchable encryption scheme [17], but the authorization is granted on the index level. If u_i wants to authorize u_j to search for a subset of keywords over its indexes, then his scheme cannot complete this task. The scheme

did not explicitly specify how a user u_i can revoke the authorization of other user u_j to search over its index.

Boneh et al. proposed some encryption schemes with keyword search in the public-key setting (PEKS) [25]. Later, Abdalla et al. studied how to transform an anonymous IBE scheme to a secure PEKS scheme [26]. Bellare et al. presented PEKS schemes that permitted fast search with provable privacy and the length preserving feature [27]. Golle et al. proposed schemes for conjunctive keyword search in public-key setting [28]. However the above PEKS schemes cannot complete our task as well.

9. Conclusion

Cloud storage has become an important storage scheme, where we can store our files and share our files with others conveniently. To keep our files from information leakage, we usually encrypt our files before storing them on the cloud storage. Encryption makes the function of file sharing and search useless. In this paper we formally define a secure and efficient searchable encryption scheme in a cloud storage, which can enable a file owner share its files with others and authorize some designated users to search its files in encrypted form on a cloud storage, and use asymmetric bilinear map groups of Type-3 and keyword authorization binary tree (KABtree) to construct this scheme that achieves better performance. Using this scheme, we can authorize a designated user to search for a subset of keywords. Our theoretical proofs and experimental results demonstrate that it is feasible.

Acknowledgments

The authors would like to express their gratitude to the anonymous reviewers for their insightful comments. The research was partially funded by the Key Program of National Natural Science Foundation of China (Grant Nos. 61133005, 61432005), and the National Natural Science Foundation of China (Grant Nos. 61370095, 61472124, 61502163).

References

[1] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility, Future Gener. Comput. Syst. 25 (2009) 599–616.
[2] D.X. Song, D. Wagner, A. Perrig, Practical techniques for searches on encrypted data, in: Proceedings of the 21st IEEE Symposium on Security and Privacy, IEEE, New York, NY, USA, 2000, pp. 44–55.
[3] E.-J. Goh, Secure indexes, IACR Cryptology ePrint Archive, 2003. Accessible on https://eprint.iacr.org/2003/216.pdf, July, 2015.
[4] R. Curtmola, J. Garay, S. Kamara, R. Ostrovsky, Searchable symmetric encryption: improved definitions and efficient constructions, in: Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS'06, ACM, New York, NY, USA, 2006, pp. 79–88.
[5] Y.-C. Chang, M. Mitzenmacher, Privacy preserving keyword searches on remote encrypted data, in: J. Ioannidis, A. Keromytis, M. Yung (Eds.), Applied Cryptography and Network Security, Springer, Berlin, Heidelberg, 2005, pp. 442–455.
[6] S. Kamara, C. Papamanthou, T. Roeder, Dynamic searchable symmetric encryption, in: Proceedings of the 19th ACM Conference on Computer and Communications Security, CCS'12, ACM, New York, NY, USA, 2012, pp. 965–976.
[7] M. Raykova, A. Cui, B. Vo, B. Liu, T. Malkin, S.M. Bellovin, S.J. Stolfo, Usable, secure, private search, IEEE Secur. Privacy 10 (2012) 53–60.
[8] B. Hore, S. Mehrotra, M. Canim, M. Kantarcioglu, Secure multidimensional range queries over outsourced data, VLDB J. 21 (2012) 333–358.
[9] B.K. Samanthula, W. Jiang, E. Bertino, Privacy-preserving complex query evaluation over semantically secure encrypted data, in: Computer Security-ESORICS 2014, Springer, Berlin, Heidelberg, 2014, pp. 400–418.
[10] F. Bao, R.H. Deng, X. Ding, Y. Yang, Private query on encrypted data in multi-user settings, in: L. Chen, Y. Mu, W. Susilo (Eds.), Information Security Practice and Experience—ISPEC 2008, Springer, Berlin, Heidelberg, 2008, pp. 71–85.
[11] C. Dong, G. Russello, N. Dulay, Shared and searchable encrypted data for untrusted servers, J. Comput. Secur. 19 (2011) 367–397.
[12] F. Zhao, T. Nishide, K. Sakurai, Multi-user keyword search scheme for secure data sharing with fine-grained access control, in: H. Kim (Ed.), Information Security and Cryptology—ICISC 2011, Springer, Berlin, Heidelberg, 2012, pp. 406–418.
[13] R. Rajan, A.V.V.P. Coimbatore, Efficient and privacy preserving multi user keyword search for cloud storage services, Int. J. Adv. Technol. Eng. Res. (IJATER) 2 (2012).
[14] Y. Yang, H. Lu, J. Weng, Multi-user private keyword search for cloud computing, in: Proceedings of 2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, New York, NY, USA, 2011, pp. 264–271.
[15] D.D. Rane, V.R. Ghorpade, Multi-user multi-keyword privacy preserving ranked based search over encrypted cloud data, in: Proceedings of the 2015 International Conference on Pervasive Computing, ICPC 2015, IEEE, New York, NY, USA, 2015, pp. 1–4.
[16] R.A. Popa, N. Zeldovich, Multi-Key Searchable Encryption, Technical Report, MIT, 2013, Accessible on http://people.csail.mit.edu/nickolai/papers/popa-multikey-eprint.pdf, July, 2015.
[17] Q. Tang, Nothing is for free: security in searching shared and encrypted data, IEEE Trans. Inf. Forensics Secur. 9 (2014) 1943–1952.
[18] X. Boyen, The uber-assumption family, in: S.D. Galbraith, K.G. Paterson (Eds.), Pairing-Based Cryptography—Pairing 2008, Springer, Berlin, Heidelberg, 2008, pp. 39–56.
[19] G. Ateniese, J. Camenisch, S. Hohenberger, B. de Medeiros, Practical group signatures without random oracles, IACR Cryptology ePrint Archive, 2005. Accessible on https://eprint.iacr.org/2005/385.pdf, July, 2015.
[20] M. Abdalla, E. Bresson, O. Chevassut, D. Pointcheval, Password-based group key exchange in a constant number of rounds, in: M. Yung, Y. Dodis, A. Kiayias, T. Malkin (Eds.), Public-Key Cryptography—PKC 2006, Springer, Berlin, Heidelberg, 2006, pp. 427–442.
[21] B. Lynn, Pbc library: the pairing-based cryptography library, https://crypto.stanford.edu/pbc/, 2007.
[22] W.W. Cohen, Enron email dataset, http://www.cs.cmu.edu/~enron/, 2005.
[23] O. Goldreich, R. Ostrovsky, Software protection and simulation on oblivious rams, J. ACM 43 (1996) 431–473.
[24] A. López-Alt, E. Tromer, V. Vaikuntanathan, On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption, in: Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing, STOC'12, ACM, New York, NY, USA, 2012, pp. 1219–1234.
[25] D. Boneh, G. Crescenzo, R. Ostrovsky, G. Persiano, Public key encryption with keyword search, in: C. Cachin, J.L. Camenisch (Eds.), Advances in Cryptology—EUROCRYPT 2004, Springer, Berlin, Heidelberg, 2004, pp. 506–522.
[26] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, H. Shi, Searchable encryption revisited: consistency properties, relation to anonymous ibe, and extensions, in: V. Shoup (Ed.), Advances in Cryptology—CRYPTO 2005, Springer, Berlin, Heidelberg, 2005, pp. 205–222.
[27] M. Bellare, A. Boldyreva, A. O'Neill, Deterministic and efficiently searchable encryption, in: A. Menezes (Ed.), Advances in Cryptology—CRYPTO 2007, Springer, Berlin, Heidelberg, 2007, pp. 535–552.
[28] P. Golle, J. Staddon, B. Waters, Secure conjective keyword search over encrypted data, in: M. Jakobsson, M. Yung, J. Zhou (Eds.), Applied Cryptography and Network Security—ACNS 2004, Springer, Berlin, Heidelberg, 2004, pp. 31–45.

Zuojie Deng received the M.S. degree in Department of Computer Engineering from Hunan University, Changsha, China, in 2004. He is an associate professor in School of Computer and Communication at Hunan Institute of Engineering, Xiangtan, China. Currently, he is a Ph.D. candidate in the School of Computer Science and Technology at Huazhong University of Science and Technology, Wuhan, China. His research areas include storage security, searchable encryption, network security and privacy enhanced technologies.

Kenli Li received the Ph.D. degree in computer science from the Huazhong University of Science and Technology, China, in 2003. He was a visiting scholar at the University of Illinois at Urbana–Champaign from 2004 to 2005. He is currently a full professor of computer science and technology at Hunan University and an associate director of National Supercomputing Center in Changsha. His major research includes parallel computing, grid and cloud computing, and DNA computing. He has published more than 90 papers in international conferences and journals, such as IEEE Transactions on Computers, IEEE Transactions on Parallel and Distributed Systems, ICPP, and CCGrid. He is an outstanding member of CCF.

Keqin Li is a SUNY Distinguished Professor of computer science. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU–GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things and cyber–physical systems. He has published over 390 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, IEEE Transactions on Cloud Computing, Journal of Parallel and Distributed Computing. He is an IEEE Fellow.

Jingli Zhou is a Professor in the School of Computer Science and Technology at Huazhong University of Science and Technology. Her current research interests include high performance storage technology and system, multimedia data processing and communication, network security. She has published more than 50 papers in international journals.