Fuzzy Keyword Search Method over Ciphertexts supporting Access Control

Zhuolin Mei[1], Bin Wu[2], Shengli Tian[3], Yonghui Ruan[4], and Zongmin Cui[2,*]
[1]School of Information Science and Technology, Huizhou University, Huizhou, China
[2]School of Information Science and Technology, Jiujiang University, Jiujiang, China
[3]School of Information Engineering, Xuchang University, Xuchang, Henan, China
[4]Department of Information Science and Technology, Wenhua College, Wuhan, Hubei, China
[e-mail: cuizm01@gmail.com]
*Corresponding author: Zongmin Cui

Received Jaunuary 8, 2017; revised May 25, 2017; accepted June 17, 2017; published November 30, 2017

Abstract

With the rapid development of cloud computing, more and more data owners are motivated to outsource their data to cloud for various benefits. Due to serious privacy concerns, sensitive data should be encrypted before being outsourced to the cloud. However, this results that effective data utilization becomes a very challenging task, such as keyword search over ciphertexts. Although many searchable encryption methods have been proposed, they only support exact keyword search. Thus, misspelled keywords in the query will result in wrong or no matching. Very recently, a few methods extends the search capability to fuzzy keyword search. Some of them may result in inaccurate search results. The other methods need very large indexes which inevitably lead to low search efficiency. Additionally, the above fuzzy keyword search methods do not support access control. In our paper, we propose a searchable encryption method which achieves fuzzy search and access control through algorithm design and Ciphertext-Policy Attribute-based Encryption (CP-ABE). In our method, the index is small and the search results are accurate. We present word pattern which can be used to balance the search efficiency and privacy. Finally, we conduct extensive experiments and analyze the security of the proposed method.

Keywords: fuzzy keyword, search, access control, encryption, cloud computing

A preliminary version of this paper appeared in ACISP 2017, July 3-5, Auckland, New Zealand. This version includes a concrete procedure of index generation and decryption, an optimization method and the security analysis under collusion attack. The optimization method is very important in our method. Without the optimization method, the efficiency performance of our method would be no better than the other methods. Thus the optimization method is an very important contribution in our paper. This research was supported by the Jiangxi Provincial Natural Science Foundation of China [No. 20161BAB202036] and the National Natural Science Foundation of China [grant number 61762055]. We express our thanks to Dr. Liang Zhou who checked our manuscript.

https://doi.org/10.3837/tiis.2017.11.027 **ISSN : 1976-7277**

1. Introduction

In recent years, with the rapid development of cloud computing, more and more sensitive information are being centralized into the cloud, such as emails, government documents, etc. By outsourcing data to the cloud, data owners can enjoy various advantages by utilizing high quality of cloud services, such as data storage, maintenance and applications [1, 2, 3]. However, the cloud is not fully trusted by the data owners. Thus, the privacy of sensitive data in the cloud naturally becomes a primary concern of data owners. To mitigate the concern, sensitive data is usually encrypted before outsourcing to prevent from unauthorized access [4, 5]. Since the data is encrypted, the searching of documents which contains specific keywords becomes rather difficult.

To solve the problem above, many searchable encryption methods [6, 7, 8, 9, 10, 11, 12, 13] have been proposed. However, they only support exact keyword matching. Misspelled keywords in the query will result in wrong or no matching. Very recently, a few works [1, 4, 14, 15, 16, 17, 18] extend the exact keyword matching to approximate keyword matching, also known as fuzzy keyword search. According to the techniques adopted in fuzzy keyword search methods, they could be classified into two classes: (1) Wildcard based fuzzy keyword search methods [1, 14, 16]; (2) Locality-Sensitive Hashing (LSH) [19] and Bloom Filter (BF) [20] based fuzzy keyword search methods [4, 15, 17]. In wildcard based fuzzy keyword search methods, data owner has to build an expanded index that covers all the possible misspelling keywords, which leads to a very large index and inefficient keyword search. In LSH and BF based fuzzy keyword search methods, the search is very efficient. However, these methods may miss out some correct search results. This is because that the adopted technique, LSH, only maps the similar items to the same hash value with a possibility. Additionally, the above fuzzy keyword search methods do not support access control, which is an important requirement of data sharing in cloud computing.

In this paper, we propose a method which not only supports fuzzy keyword search but also access control. For each document, the data owner defines a document policy, which consists of several error-tolerance policies and one access control policy. An error-tolerance policy represents the maximal number of misspellings a user can make when searching a keyword. The access control policy represents the users who have the privilege to search the document. Next, according to the document policy, the data owner generates an index for the document by using the Ciphertext-Policy Attribute-based Encryption (CP-ABE) method. Then, the data owner assigns secret keys and attributes to users according to their identifiers. A user can retrieve a document, if and only if (1) the user's attributes satisfy the access control policy in the document policy, and (2) the searched word in the user's query satisfies one error-tolerance policy in the document policy. We present word pattern which could be used to balance the search efficiency and privacy. Finally, we give a fuzzy keyword search algorithm. Compared with wildcard based fuzzy keyword search methods [1, 14, 16], our method has smaller index size and is more efficient. Compared with LSH and BF based fuzzy keyword search methods [4, 15, 17], our method can accurately obtain all the search results. Through extensive experiments and rigorous security analysis, we show that our method is efficient and secure. The contributions of this paper are listed as follows:

(1) We propose a method which supports both the fuzzy keyword search and access control.

(2) We present word pattern and construct a fuzzy keyword search algorithm. By utilizing word pattern, the search efficiency and privacy can be balanced.

(3) We implement our method. The experimental results show that our method is efficient. We analyze the security of our method, and our proposed method is secure under the known ciphertext model.

The reminder of this paper is organized as follows: **Section 2** is the preliminaries. **Section 3** is word pattern. **Section 4** is the construction of our fuzzy keyword search method. **Section 5** shows the experiment results. **Section 6** represents the security analysis and proofs. **Section 7** is the related work.

2. Preliminaries

Definition 1. A ciphertext-policy attribute based encryption (CP-ABE) [21] consists of five algorithms: Setup, Encrypt, KeyGen, Delegate and Decrypt.

$Setup(\lambda) \to (PK, MK)$. The setup algorithm takes the security parameter λ as input, and outputs a public key PK and a master key *MK*.

$Encrypt(PK, M, P) \to CT$. The encryption algorithm takes the public key PK, a message *M*, and a policy *P* as input. The algorithm encrypts *M* and outputs a ciphertext *CT*.

$KeyGen(MK, S) \to SK$. The key generation algorithm takes the master key *MK* and an attribute set *S* as input. It outputs a private key *SK*.

$Delegate(SK, \tilde{S}) \to \widetilde{SK}$. The delegate algorithm takes as input a secret key *SK* for the attributes in *S* and a set $\tilde{S} \subseteq S$. It outputs a secret key $\widetilde{SK}$ for the attributes in $\tilde{S}$.

$Decrypt(PK, CT, SK) \to M$. The decryption algorithm takes as input the public key PK, a ciphertext *CT* (*CT* contains a policy *P*), and a private key *SK* (*SK* contains the attribute set *S*). If *S* satisfies *P*, then the algorithm decrypts *CT* and outputs *M*.

There are some facts related to groups with efficiently computable bilinear maps which are used in CP-ABE [21]. Let G_0 and G_1 be two multiplicative cyclic groups of prime order *p*. Let *g* be a generator of G_0 and *e* be a bilinear map, $e: G_0 \times G_0 \to G_1$. The bilinear map *e* has the following properties: (1) Bilinearity: For all $u, v \in G_0$ and $a, b \in Z_p$, we have $e(u^a, v^b) = e(u, v)^{ab}$. (2) Non-degeneracy: $e(g, g) \neq 1$.

Definition 2. Edit distance [1]. Edit distance is a well-studied method to quantitatively measure the word similarity. The edit distance $ed(w, w')$ between two words *w* and w' is the number of operations required to transform one of them into the other. The operations are substitution, deletion and insertion. Substitution is the operation that changes one character to another in a word. Deletion is the operation that deletes one character from a word. Insertion is the operation that inserts one character into a word.

3. Word Pattern

In this paper, we propose word pattern, word pattern function, and character-appearing order. The word pattern could be used to balance the search efficiency and security. The word pattern function is used to compute the word pattern. The character-appearing order provides the correct way to perform fuzzy keyword search.

Definition 3. Word pattern function $F_M(w^\circ, i)$ is defined as below

$$F_M(w^\circ, i) = \begin{cases} (H_M(c^i_{w^\circ}) - H_M(c^{|w^\circ|}_{w^\circ})) \bmod sp, i = 1 \\ (H_M(c^i_{w^\circ}) - H_M(c^{i-1}_{w^\circ})) \bmod sp, 1 < i \leq |w^\circ| \end{cases}$$

, where H_M is a hash function, $c^i_{w^\circ}$ is the i th character in w° (w° is a keyword w or a searched word w') and sp is a positive integer defined by data owner.

Definition 4. $M_{w^\circ} = (F_M(w^\circ, 1), F_M(w^\circ, 2), ..., F_M(w^\circ, |w^\circ|))$ is the word pattern of w°. $m^i_{w^\circ} = F_M(w^\circ, i)$ is the word pattern value of the i th character in w°, where $i = 1, ..., |w^\circ|$.

Let S_{char} denote the character set which is used to spell all the keywords, and $|S_{char}|$ denote the number of characters in S_{char}. According to the Definition 3, a word pattern value corresponds to average $|S_{char}|^2 / sp$ two-contiguous characters. Given a word pattern value, one can not know the word pattern value is obtained by calculating which two-contiguous characters. Data owner could use sp to balance the privacy (keyword privacy and searched word privacy) and search efficiency: (1) Smaller sp means there are more characters who have the same word pattern value. Thus, when decreasing sp, the security could be enhanced. However, different two-contiguous characters who have the same word pattern value would affect the efficiency of fuzzy keyword search (see Section 4.5). (2) Larger sp means there are fewer characters who have the same word pattern value. Thus, when increasing sp, the search efficiency could be improved, but the security decreases.

Definition 5. Given a word $w^\circ = c^1_{w^\circ} c^2_{w^\circ} ... c^{|w^\circ|}_{w^\circ}$, for $\forall i < j$ ($i, j \in [1, |w^\circ|]$), the character appearing order of w° is that $c^i_{w^\circ}$ is before $c^j_{w^\circ}$.

As keywords and searched words are encrypted, it is difficult to measure the word similarity according to edit distance. Fortunately, we find a method to judge whether $ed(w, w') \leq e_w$ according to $|w \cap w'|_O$ (see **Theorem 2** in Section 4.4), where $|w \cap w'|_O$ is the maximal number of characters which are the same in w and w' in the character appearing order of w and w'. For example, given a keyword $w = cat$ and a searched word $w' = acat$, it is easy to compute $|w \cap w'|_O = 3$.

4. Construction of Fuzzy Keyword Search supporting Access Control

In our scheme, the cloud is considered to be ”honest-but-curious” [1, 14, 15, 17]. Thus the cloud would honestly follow the designated protocols and procedures to fulfill the service provider’s role, while it may analyze the information stored and processed on the cloud in order to learn additional information about its customers. In our scheme, first data owner builds indexes for documents and encrypts documents using a secure encryption method, such as AES. Next, the data owner stores the encrypted documents and indexes on the cloud. Then, the data owner distributes secret keys to users according to their identifiers. A user uses his/her secret key to generate trapdoors for the searched words and sends the trapdoors to the cloud for fuzzy keyword search. Upon receiving the trapdoors, the cloud server performs fuzzy keyword search and replies with the encrypted documents which contain the searched words.

4.1 System Setup

The data owner defines a finite character set S_{char} and a finite attribute set S_{attr}. S_{char} contains all the characters which are used to spell keywords and S_{attr} contains all the attributes. For example, $S_{char} = \{a, b, ..., z, A, B, ..., Z, -, +, /, ...\}$. For a school of computer science, $S_{attr} = \{ student, professor, computerScience, ...\}$. Next, the data owner runs the algorithm $Setup$ [21]. $Setup$ takes the security parameter λ as input. It outputs the public key PK and master key MK. The public key PK is $G_0, g, h = g^{\beta}, f = g^{1/\beta}, e(g,g)^{\alpha}$ and the master key MK is β, g^{α}, where G_0 is a bilinear group of prime order p with generator g, α and β are randomly chosen from Z_p. Then, the data owner chooses a hash function H_M and a positive integer sp to construct the word pattern function F_M. Finally, the data owner publishes F_M, but keeps MK and PK secretly.

4.2 Building Index

For each document $\mathcal{D}$, the data owner defines a document policy p_D. p_D consists of two kinds of policies: (i) Error-tolerance policy; (ii) Access control policy. For each keyword of $\mathcal{D}$, the error-tolerance policy limits the maximal typos that a user could make when searching a keyword. For the document $\mathcal{D}$, the access control policy represents who has the privilege to search the keywords of $\mathcal{D}$. In the following sections, we first describe the error-tolerance policy and access control policy respectively. Then, we show how to construct p_D by using error-tolerance policy and access control policy. Finally, we show how to generate the index Idx_D for the document $\mathcal{D}$ under p_D.

Before describing the policies in our method, we want to explain the threshold gate in detail. In our method, error-tolerance policy, access control policy and document policy can be transform into tree structures. In these tree structures, each internal node is associated with a threshold gate and each leaf is associated with a character (or an attribute). If an internal node is associated with the threshold gate $T(n,m)$, it means that (i) the internal node has m children, (ii) $T(n,m)$ returns true if and only if there are at least n children who return true, (iii) a leaf returns true if and only if the character (or attribute) associated with the leaf matches the character (or attribute) in the query of a user. For example, OR could be represented as $T(1,m)$, AND could be represented as $T(m,m)$.

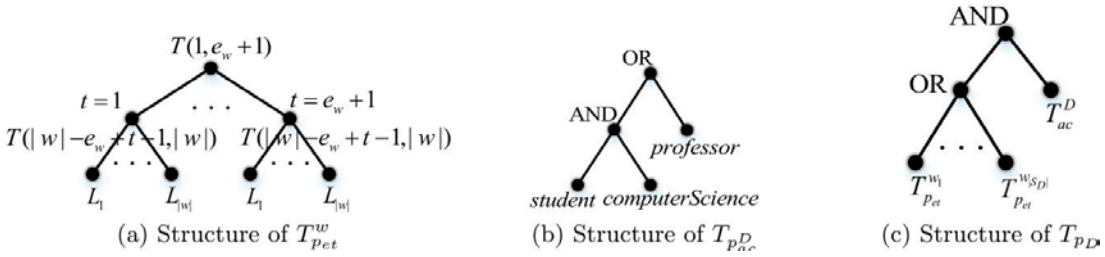

Fig. 1. The tree structures of error-tolerance policy, access control policy and document policy

Error-tolerance Policy. For each keyword w of the document $\mathcal{D}$, the data owner defines the maximal error-tolerance value e_w and an error-tolerance policy p_{et}^w. p_{et}^w could be represented as a three-level tree, denoted by $T_{p_{et}}^w$. As shown in **Fig. 1 (a)**, the root in $T_{p_{et}}^w$ is composed of a threshold gate $T(1, e_w+1)$. The root has e_w+1 children, which are numbered from 1 to e_w+1. The t th child is composed of the threshold gate $T(|w|-e_w+t-1, |w|)$, where $t=1,2,...,e_w+1$. For each subtree of $T_{p_{et}}^w$, L_i is the i th leaf of the subtree and L_i is associated with the tuple $<c_w^i, i, F_M(w,i)>$, where c_w^i is the i th character in w, i is the order of c_w^i in w, and $F_M(w,i)$ is the word pattern value of c_w^i.

These threshold gate $T(|w|-e_w+t-1, |w|)$ $(t=1,...,e_w+1)$ could be used to determine whether a searched word w' satisfies $ed(w,w') \le e_w$. Specifically, our method uses the threshold gate $T(|w|-e_w+t-1, |w|)$ to perform fuzzy keyword search: If $|w| \ge |w'|$, our method chooses the threshold gate in which $t=1$; If $|w'|-|w|>0$, our method chooses the threshold gate in which $t=|w'|-|w|+1$ (see **Theorem 2** in **Section 4.4**). The order i provides the correct comparison order during the fuzzy keyword search (see the fuzzy keyword search algorithm in **Section 4.5**).

Access Control Policy. For each document $\mathcal{D}$, the data owner defines an access control policy p_{ac}^D. p_{ac}^D could be represented as a tree, denoted by $T_{p_{ac}^D}$. Internal nodes in $T_{p_{ac}^D}$ are composed of threshold gates and the leaves are associated with attributes. **Fig. 1 (b)** shows the tree structure of the access control policy p_{ac}^D =*professor* OR (*student* AND *computerScience*). For this access control policy p_{ac}^D, it is true if and only if the user is a professor or a student of computer science. By setting similar access control policies, our method could realize the access control when performing fuzzy keyword search.

Document Policy. For each document $\mathcal{D}$, after constructing the access control policy and error tolerance policies for all the keywords in $\mathcal{D}$, the data owner constructs a document policy p_D. Suppose (1) $\{p_{et}^{w_1}, p_{et}^{w_2}, ... | w_1, w_2, ... \in S_{w_D}\}$ is the set of all the error-tolerance policies of keywords in $\mathcal{D}$, where S_{w_D} is the keyword set of $\mathcal{D}$, (2) p_{ac}^D is the access control policy of $\mathcal{D}$. The formal description of p_D is $p_D = (p_{et}^{w_1} \vee ... \vee p_{et}^{w_{|S_{w_D}|}}) \wedge p_{ac}^D$, where $|S_{w_D}|$ is the total of keywords in S_{w_D}. Because $p_{et}^{w_1}, ..., p_{et}^{w_{|S_{w_D}|}}$ and p_{ac}^D could be represented as trees $T_{p_{et}}^{w_1}, ..., T_{p_{et}}^{w_{|S_{w_D}|}}$ and $T_{p_{ac}^D}$, the policy p_D could be represented as T_{p_D} (as shown in **Fig. 1 (c)**).

Index Generation. For each document $\mathcal{D}$, the data owner runs the algorithm $Encrypt(PK, ID_D \| 0^l, T_{p_D})$ [21] to generate the index, where PK is the public key, ID_D is the identity of $\mathcal{D}$ and T_{p_D} is the document policy of $\mathcal{D}$. Note that, the data owner appends l 0s to the identity ID_D, denoted by $ID_D \| 0^l$. In this way, the cloud could check whether a decryption is valid [22]. If a decryption outputs a plaintext that there are l 0s at the end of the plaintext, then the decryption is valid. Otherwise, it is invalid.

We briefly describe the encryption algorithm *Encrypt* (details are in [21]). In [21], the children of an internal node in T_{p_D} are numbered from 1 to *num*. The function $index(x)$ returns such a number associated with the node x. The function $parent(x)$ returns the parent of the node x. *Encrypt* first chooses a polynomial q_x for each node in the tree T_{p_D}. These polynomials are chosen in a top-down manner, starting from the root of T_{p_D}. For each node x (suppose the threshold gate of x is $T(n_x, m_x)$) in T_{p_D}, set the degree of q_x to $n_x - 1$. For the root R of T_{p_D}, *Encrypt* chooses a random $s \in Z_p$ and sets $q_R(0) = s$. For any other node x, *Encrypt* sets $q_x(0) = q_{parent(x)}(index(x))$ and randomly chooses d_x other points to completely define q_x. Then, *Encrypt* [21] encrypts $ID_D \| 0^l$ under p_D as follows.

$CT = (T_{p_D}, \tilde{C} = (ID_D \| 0^l) e(g,g)^{\alpha s}, C = h^s, \forall y \in Y : C_y = g^{q_y(0)}, C_y' = H(f(y))^{q_y(0)})$,

where Y is the set of leafs in T_{p_D}, $f(y)$ is the function which returns the character or attribute associated with y, and H is a hash function. Note that, the threshold gates now have been embedded in the ciphertexts by using this method.

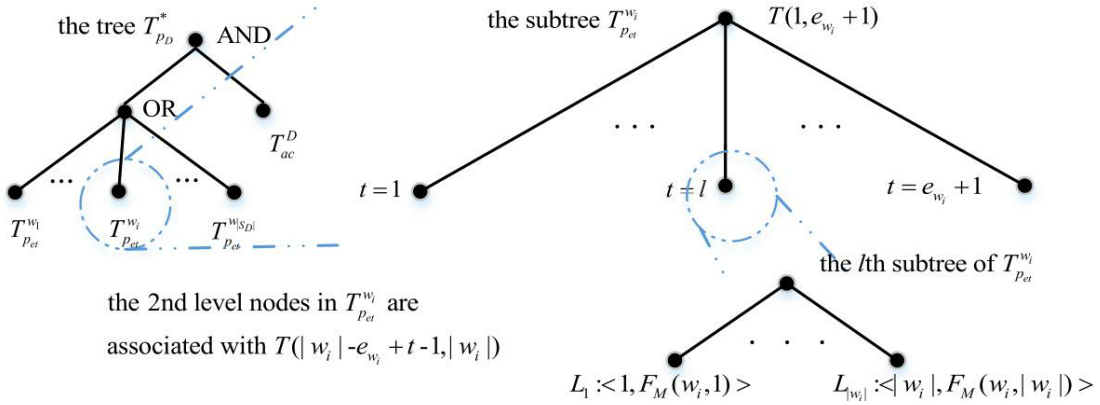

Fig. 2. The tree structure $T^*_{p_D}$ of p_D

In [21], the characters associated with the leafs of T_{p_D} are required to be stored in plaintext. However, data owner should protect the privacy of keywords in indexes. Thus, we transform the tree T_{p_D} into a new tree $T^*_{p_D}$ (see **Fig. 2**) to hide the characters of keywords. Then, the data owner could construct the index Idx_D according to the ciphertext CT of $ID_D \| 0^l$.

$Idx_D = (T^*_{p_D}, \tilde{C} = (ID_D \| 0^l) e(g,g)^{\alpha s}, C = h^s,$

$\forall w_i \in S_{w_D} : e_{w_i},$

$\forall y_k \in Y(T^{w_i}_{P_{et}})$ and y_k is the leaf of the t th subtree of $T^{w_i}_{P_{et}}$:

$C_{y_k} = g^{q_{y_k}(0)}, C_{y_k}' = H(f(y_k))^{q_{y_k}(0)}, i, t, k, F_M(w_i, k),$

$\forall y \in Y(T^D_{p_{ac}})$:

$C_y = g^{q_y(0)}, C_y' = H(f(y)^{q_y(0)})$

, where e_{w_i} is the maximal error-tolerance value of w_i, $char(y_k)$ is the character $c_{w_i}^k$ associated with the leaf node y_k, $f(y_k)$ (or $f(y)$) returns the character (or attribute) associated with the leaf y_k (or y), $Y(T_{p_{et}}^{w_i})$ is the set of leafs of $T_{p_{et}}^{w_i}$ $(i=1,...,w_{|S_{w_D}|})$, $Y(T_{p_{ac}}^{D})$ is the set of leafs of $T_{p_{ac}}^{D}$. For simplicity, in the following paragraphs, $(C_{y_k}=g^{q_{y_k}(0)}, C_{y_k}'=H(f(y_k))^{q_{y_k}(0)})$ is denoted by the notation $\widehat{C_{c_{w_i}^k}}$. $\widehat{C_{c_{w_i}^k}}$ is the **ciphertext component** corresponding to the k th character $c_{w_i}^k$ in the keyword w_i.

We observe that different documents may have the same keywords, and their indexes have the same word patterns. Thus, adversaries may infer the keywords by obersving the word patterns. Namely, if the word patterns of two keywords are the same, the adversaries may guess the two keywords are the same. To prevent such attacks, the data owner could randomize the keywords and word patterns. For each keyword $w_i=c_{w_i}^1...c_{w_i}^{|w_i|}$ in the document $\mathcal{D}$, first the data owner chooses an integer a_{1w_i} $(a_{1w_i}\geq 0)$ randomly, and next chooses a_{1w_i} characters randomly, denoted by $\eta_1,...,\eta_{a_{1w_i}}$. Then, the data owner chooses a_{1w_i} positions in w_i randomly, and put $\eta_1,...,\eta_{a_{1w_i}}$ at these positions. For simplicity, we use the notation w_i^R to denote the randomized w_i. Then, the data owner computes the word pattern and character-appearing order of w_i^R. For each artificial character η_i, the data owner chooses two random values $(C_{\eta_i}, C_{\eta_i}')$ as the ciphertext component of η_i, s.t. $|C_{\eta_i}|=|C_{y_j}|$ and $|C_{\eta_i}'|=|C_{y_j}'|$, where (C_{y_j}, C_{y_j}') is the ciphertext component of $c_{w'}^j$ and $c_{w'}^j$ is a character randomly chosen from the keyword w_i. According to [21], C_{y_j} and C_{y_j}' are computationally indistinguishable from random values. Thus, after randomization, attackers cannot distinguish which character is a real character or an artificial character according to the word pattern values and ciphertext components. So, the index Idx_D can prevent the statistical attacks.

4.3 Trapdoor Generation

For each user u, the data owner runs the algorithm $KeyGen$ [21] to generate the secret key. In our method, the algorithm $KeyGen$ takes as input the master key MK and $S_{char}\cup S_{attr_u}$ (S_{char} is the character set which is used to spell all the keywords of documents, and S_{attr_u} is the attribute set distributed to u by the data owner). It outputs the secret key SK_u for u. $KeyGen$ first randomly chooses a number γ from Z_p, a number γ_c from Z_p for each character $c\in S_{char}$, and a number γ_a from Z_p for each attribute $a\in S_{attr_u}$. Then $KeyGen$ computes the secret key as

$$SK_u=(D=g^{(\alpha+\gamma)/\beta}, \forall c\in S_{char}: D_c=g^{\gamma}\cdot H(c)^{\gamma_c}, D_c'=g^{\gamma_c}, \forall a\in S_{attr_u}: D_a=g^{\gamma}\cdot H(a)^{\gamma_a}, D_a'=g^{\gamma_a})$$

Then, the data owner distributes the secret key SK_u to the user u. For simplicity, in the

following paragraphs, we use the notation $\overline{D_c}$ to denote $(D_c = g^{\gamma} \cdot H(c)^{\gamma_c}, D_c' = g^{\gamma_c})$. $\overline{D_c}$ is the **secret key component** corresponding to the character c.

The trapdoor generation for a searched word $w' = c_{w'}^1 ... c_{w'}^{|w'|}$ is as follows. In the secret key SK_u of u, there are secret key components for all the characters. Thus, for each character in w', u could find a corresponding secret key component. Then, u could generate a trapdoor $Td_{w'}$ for w'. The trapdoor $Td_{w'}$ is the tuple $< Td_{char}, Td_{attr} >$, where $Td_{char} = \{< \overline{D_{c_{w'}^i}}, i, F(w', i) > | i = 1, ..., | w' |\}$ ($\overline{D_{c_{w'}^i}}$ is the secret key component for the i th character $c_{w'}^i$ of w'), and $Td_{attr} = \{(D_a, D_a') | a \in S_{attr_u}\}$.

For the character c, the secret key component $\overline{D_c}$ in different trapdoors are always the same. Thus, adversaries may infer the searched word by obersving the secret key components. To prevent such attacks, u could run the delegate algorithm $Delegate$ [21] to generate new secret keys. Then, for each search, u uses the new secret keys to generate a trapdoor. We also observe that if the searched words are the same, their word patterns in the trapdoors are the same. Thus, adversaries may also infer the searched word by obersving the word patterns. To prevent such attacks, users could choose $a_{2w'}$ characters randomly, denoted by $\mu_1, ..., \mu_{a_{2w'}}$, and use the same randomization method in **Section 4.2** to randomize their trapdoors. Finally, u sends the trapdoor $Td_{w'}$ to the cloud to perform search.

4.4 Theorem, Property and Optimization

In this section, we give some theorems, properties and an optimization approach. These theorems and properties are the basis of our proposed method. Note that, the explanation of the notation $| w \cap w' |_O$ in the following **Theorem 2** can be found in **Section 3**.

Theorem 1. w is a keyword of a document and w' is a searched word. e_w ($e_w \geq 0$) is the maximal error-tolerance value of w. If $ed(w, w') \leq e_w$, then there is $| w | - e_w \leq | w' | \leq | w | + e_w$, where $| w |$ and $| w' |$ denote the number of characters in w and w' respectively.

As **Theorem 1** is easy to be proofed, we do not give the proof of **Theorem 1**. **Theorem 1** is the necessary condition of $ed(w, w') \leq e_w$. Thus, some keywords in indexes, which do not meet users' search requests, can be filtered out by using **Theorem 1**.

Theorem 2. w is a keyword of a document and w' is a searched word. e_w ($e_w \geq 0$) is the maximal error-tolerance value of w. $ed(w, w') \leq e_w$ if and only if $| w \cap w' |_O \geq | w_{\max} | - e_w$ (If $| w | \geq | w' |$, $| w_{\max} | = | w |$ and $| w_{\min} | = | w' |$; Otherwise, $| w_{\max} | = | w' |$ and $| w_{\min} | = | w |$).

Proof.

(1) The proof of "if $ed(w, w') \leq e_w$, then there is $| w \cap w' |_O \geq | w_{\max} | - e_w$".

Case 1: $w_{\min}$ could be transformed into $w_{\max}$ only by using substitutions and insertions.

Suppose $| w_{\max} \cap w_{\min} |_O = n$. As $ed(w_{\max}, w_{\min}) \leq e_w$, we can suppose $ed(w_{\max}, w_{\min}) = e_w - k$, where k ($0 \leq k \leq e_w$) is an integer. According to **Definition 2**, after one operation

(substitution or insertion), the word $w_{\min}$ could be transformed into a new word $w_{\min+1}$. $w_{\min+1}$ satisfies $ed(w_{\max}, w_{\min+1}) = ed(w_{\max}, w_{\min}) - 1$ and $|w_{\max} \cap w_{\min+1}|_O = n+1$. Thus, it is easy to know that, after $e_w - k$ operations (substitutions and insertions), the word $w_{\min}$ could be transformed into a new word $w_{\min+(e_w-k)}$. $w_{\min+(e_w-k)}$ satisfies $ed(w_{\max}, w_{\min+(e_w-k)}) = ed(w_{\max}, w_{\min}) - (e_w - k) = 0$ (note $ed(w_{\max}, w_{\min}) = e_w - k$) and $|w_{\max} \cap w_{\min+(e_w-k)}|_O = n + (e_w - k)$. According to **Definition 2**, as there is $ed(w_{\max}, w_{\min+(e_w-k)}) = 0$, we have the conclution $w_{\max} = w_{\min+(e_w-k)}$. Thus, we have $|w_{\max} \cap w_{\min+(e_w-k)}|_O = |w_{\max}|$. Because there are $|w_{\max} \cap w_{\min+(e_w-k)}|_O = |w_{\max}|$ and $|w_{\max} \cap w_{\min+(e_w-k)}|_O = n + (e_w - k)$, we have the conclusion $|w_{\max}| = n + (e_w - k)$. Then, as $|w_{\max} \cap w_{\min}|_O = n$ and $0 \le k \le e_w$, there is $|w_{\max} \cap w_{\min}|_O \ge |w_{\max}| - e_w$. Namely, we have $|w \cap w'|_O \ge |w_{\max}| - e_w$.

Case 2: Substitutions, insertions and deletions are required to transform $w_{\min}$ into $w_{\max}$.

To transform $w_{\min}$ into $w_{\max}$, substitutions and insertions can change the value of $|w_{\max} \cap w_{\min}|_O$, but deletions can not change the value of $|w_{\max} \cap w_{\min}|_O$. Thus, if substitutions, insertions and deletions are required to transform $w_{\min}$ into $w_{\max}$, the conclusion $|w_{\max} \cap w_{\min}|_O \ge |w_{\max}| - e_w$ in Case 1 is still correct in Case 2. Namely, we have $|w \cap w'|_O \ge |w_{\max}| - e_w$.

(2) The proof of "if $|w \cap w'|_O \ge |w_{\max}| - e_w$, then there is $ed(w, w') \le e_w$".

As $|w_{\max}| \ge |w_{\min}| \ge |w_{\max} \cap w_{\min}|_O$ and $|w_{\max} \cap w_{\min}|_O \ge |w_{\max}| - e_w$, we have: $|w_{\max}| \ge |w_{\min}| \ge |w_{\max}| - e_w$. As $|w_{\max} \cap w_{\min}|_O \ge |w_{\max}| - e_w$, there are at least $|w_{\max}| - e_w$ characters are the same in $w_{\min}$ and $w_{\max}$. Namely, there are at most e_w characters are different in $w_{\min}$ and $w_{\max}$. Thus, to transform $w_{\min}$ into $w_{\max}$, the total of operations is no greater than e_w. Thus, we can know that the edit distance between $w_{\min}$ and $w_{\max}$ is less than or equal to e_w. Then, we have the conclusion $ed(w, w') \le e_w$. □

Theorem 2 is the sufficient and necessary condition of $ed(w, w') \le e_w$. Thus, by using **Theorem 2**, our method could correctly perform fuzzy keyword search. According to **Theorem 1**, if w and w' satisfy $ed(w, w') \le e_w$, then there are $|w_{\max}| = |w|$, $|w|+1$, … , $|w|+e_w$. Thus, thresholds in the error-tolerance policy p_{et}^w are $T(|w|-e_w, |w|)$, $T(|w|-e_w+1, |w|)$, … , $T(|w|, |w|)$. For simplicity, these threshold gates could be written as $T(|w|-e_w+t-1, |w|)$, where $t = 1, 2, \ldots, e_w+1$. When $|w| \ge |w'|$, it is obvious that the threshold gate in which $t = 1$ could be used to test whether $ed(w, w') \le e_w$. When $|w'| > |w|$, it requires at least $|w'| - |w|$ deletions to transform w' into w. Then, at most $e_w - (|w'| - |w|)$ operations are left which could be used to transform w' into w. Thus, there are at least $|w| - [e_w - (|w'| - |w|)]$ characters should be the same in w and w'. If one uses $T(|w|-e_w+t-1, |w|)$ to test whether $ed(w, w') \le e_w$, then there is

$|w|-[e_w-(|w'|-|w|)]=|w|-e_w+t-1$. Thus, we can compute $t=|w'|-|w|+1$. Namely, if $|w'|>|w|$, the threshold gate in which $t=|w'|-|w|+1$ could be used to test whether $ed(w,w')\le e_w$ (see **Section 4.2**).

As each character of w corresponds to a ciphertext component in Idx_D (see **Section 4.2**) and each character of w' corresponds to a secret key component in the trapdoor (see **Section 4.3**). A secret key component can be used to decrypt a ciphertext component, if and only if they have the correct corresponding relationship. Thus, we have the following definition.

Definition 6. Corresponding relationship. In an index Idx_D, $\widehat{C_{c_w^i}}$ is a ciphertext component, which corresponds to the ith character c_w^i in w. In a trapdoor $Td_{w'}$, $\overline{D_{c_{w'}^j}}$ is a secret key component, which corresponds to the jth character $c_{w'}^j$ in w'. A corresponding relationship is the tuple $<\widehat{C_{c_w^i}},\overline{D_{c_{w'}^j}}>$. $<\widehat{C_{c_w^i}},\overline{D_{c_{w'}^j}}>$ is correct if $c_w^i=c_{w'}^j$. Otherwise, it is wrong.

As keywords in indexes and searched words in trapdoors have been hidden to protect the privacy, it is difficult to find out the correct corresponding relationships. According to the decryption algorithm *Decrypt* [21], if there are not **enough** correct corresponding relationships, the cloud server cannot decrypt the indexes to obtain the identities of documents. According to **Theorem 2**, the "enough" means that $|w\cap w'|_O\ge|w_{\max}|-e_w$.

We give the following four properties, which could help the cloud to efficiently find out the corresponding relationships which may be correct. In the following properties, m_w^i is the ith value in the word pattern of w, and $m_{w'}^j$ is the jth value in the word pattern of w'. $A\xrightarrow{p}B$ denotes, if there is A, then there is B with the probability p. Note that: (1) For each c_w^i and m_w^i, $i=i \bmod |w|$ if $i>|w|$; (2) For each $c_{w'}^j$ and $m_{w'}^j$, $j=j \bmod |w'|$ if $j>|w'|$. Thus, the meaning of "contiguous" in the following properties is more general. For example, $c_w^{|w|}c_w^1$ in w, and $c_{w'}^{|w'|}c_{w'}^1$ in w' are all contiguous characters.

For clarity, we use the notations c_w^i, $c_{w'}^j$ to illustrate the idea of properties and optimization approach. In fact, the properties and optimization approach are used to handle the components $\widehat{C_{c_w^i}}$, $\overline{D_{c_{w'}^j}}$. c_w^i and $\widehat{C_{c_w^i}}$ have the same order i. $c_{w'}^j$ and $\overline{D_{c_{w'}^j}}$ have the same order j. Thus, if there is $<c_w^i,c_{w'}^j>$, then there is the corresponding relationship $<\widehat{C_{c_w^i}},\overline{D_{c_{w'}^j}}>$.

The following properties show the relationships between two words and their word patterns. As these properties are easy to be proofed, we do not give the proof of them.

Property 1. $m_w^{i+1}=m_{w'}^{j+1}\xrightarrow{p}c_w^i=c_{w'}^j$ and $c_w^{i+1}=c_{w'}^{j+1}$

Property 2. $(m_w^{i+1}+m_w^{i+2}+...+m_w^{i+k}) \bmod sp=m_{w'}^{j+1}\xrightarrow{p}c_w^i=c_{w'}^j$ and $c_w^{i+k}=c_{w'}^{j+1}$

Property 3. $m_w^{i+1}=(m_{w'}^{j+1}+m_{w'}^{j+2}+...+m_{w'}^{j+k}) \bmod sp\xrightarrow{p}c_w^i=c_{w'}^j$ and $c_w^{i+1}=c_{w'}^{j+k}$

Property 4. $(m_w^{i+1}+m_w^{i+2}+...+m_w^{i+k}) \bmod sp=(m_{w'}^{j+1}+m_{w'}^{j+2}+...+m_{w'}^{j+t}) \bmod sp$
$\xrightarrow{p}c_w^i=c_{w'}^j$ and $c_w^{i+k}=c_{w'}^{j+t}$

According to the above properties, many corresponding relationships could be found. However, in these corresponding relationships, there are many wrong corresponding relationships. This is because that a corresponding relationship is correct only with the probability $p=1/(|S_{char}|^2/sp)$ (see **Section 3**). As wrong corresponding relationships will reduce the efficiency of the query, we give an optimization approach to eliminate the wrong corresponding relationships as many as possible. The optimization approach is given below.

Step 1, given the word patterns of w and w', the cloud could compute quite a few corresponding relationships according to **Property 1**, **2**, **3** and **4**. Note that, if the maximal error-tolerance value of w is e_w, the k in the above properties should satisfy $k \le e_w+1$. This is because, if the first and the last character in a $k+1$ contiguous-character string have been matched, it means that there are $k-1$ typos. Thus we have $k-1 \le e_w$ (namely, $k \le e_w+1$). For the same reason, the t in the **Property 4** should satisfy $t \le e_w+1$. Finally, the cloud can obtain a set, denoted by $Set_{Step1}=\{(<c_w^i,c_{w'}^j>,<c_w^p,c_{w'}^q>),...\}$.

Step 2, if the ith character in w' corresponds to the jth character in w, it is obvious that i and j should satisfy $|i-j| \le e_w$. Thus, the cloud should delete $(<c_w^i,c_{w'}^j>,<c_w^p,c_{w'}^q>)$ in Set_{Step1} if $|i-j| \not\le e_w$ or $|p-q| \not\le e_w$. For clarity, let Set_{Step2} denote the set, in which some wrong corresponding relationships in Set_{Step1} have been deleted.

Step 3, if $(<c_w^i,c_{w'}^j>,<c_w^p,c_{w'}^q>)$ is correct, both $<c_w^i,c_{w'}^j>$ and $<c_w^p,c_{w'}^q>$ should appear in Set_{Step2} at least 2 times. Otherwise, $(<c_w^i,c_{w'}^j>,<c_w^p,c_{w'}^q>)$ is wrong and should be deleted. Let Set_{Step3} denote the set of corresponding relationships after executing Step 3.

Step 4, for each $(<c_w^i,c_{w'}^j>,<c_w^p,c_{w'}^q>) \in Set_{Step3}$, the cloud extracts $<c_w^i,c_{w'}^j>$ and $<c_w^p,c_{w'}^q>$. Then, the cloud adds $<c_w^i,c_{w'}^j>$ and $<c_w^p,c_{w'}^q>$ into a set, denoted by $Set_{Step4}=\{<c_w^i,c_{w'}^j>\}$. If a tuple has been added into the set, the cloud does not add it again.

Step 5, the cloud extracts all the corresponding relationships from Set_{Step4}. If the total of corresponding relationships is less than $|w_{\max}|-e_w$, this means that w' and w do not fuzzy match. Otherwise, the cloud calculates all the combinations (each combination contains $|w_{\max}|-e_w$ corresponding relationships). Let Set_{Step5} denote the set of these combinations.

Step 6, for each combination in Set_{Step5}, the cloud sorts its corresponding relationships. Let $(<c_w^{i_1},c_{w'}^{j_1}>,<c_w^{i_2},c_{w'}^{j_2}>,<c_w^{i_3},c_{w'}^{j_3}>,...,<c_w^{i_n},c_{w'}^{j_n}>)$ denote a sorted combination, where $n=|w_{\max}|-e_w$ and $i_1<i_2<...<i_n$. Then, the cloud checks whether $j_1<j_2<...<j_n$. If the combination does not satisfy $j_1<j_2<...<j_n$, the cloud deletes the combination in Set_{Step5}. Recall the explanation of $|w \cap w'|_O$ in **Section 3**, $(i_1,i_2,...,i_n)$ and $(j_1,j_2,...,j_n)$ represents the orders of character-appearing orders of w and w'. Thus, $(i_1,i_2,...,i_n)$ and $(j_1,j_2,...,j_n)$ should be in strictly monotone increasing order. Let Set_{Step6} denote the set of combinations after executing Step 6.

Step 7, for each combination $(<c_w^{i_1}, c_{w'}^{j_1}>, <c_w^{i_2}, c_{w'}^{j_2}>, <c_w^{i_3}, c_{w'}^{j_3}>, ..., <c_w^{i_n}, c_{w'}^{j_n}>)$ in Set_{Step6}, the cloud checks whether it is correct. According to the definition of word pattern (**Definition 4**), a correct combination should satisfy,

$(m_w^{i_1+1} + m_w^{i_1+2} + ... + m_w^{i_2}) \bmod sp = (m_{w'}^{j_1+1} + m_{w'}^{j_1+2} + ... + m_{w'}^{j_2}) \bmod sp$, $(m_w^{i_2+1} + m_w^{i_2+2} + ... + m_w^{i_3}) \bmod sp = (m_{w'}^{j_2+1} + m_{w'}^{j_2+2} + ... + m_{w'}^{j_3}) \bmod sp, ..., (m_w^{i_{n-1}+1} + m_w^{i_{n-1}+2} + ... + m_w^{i_n}) \bmod sp = (m_{w'}^{j_{n-1}+1} + m_{w'}^{j_{n-1}+2} + ... + m_{w'}^{j_n}) \bmod sp$, $(m_w^{i_n+1} + m_w^{i_n+2} + ... + m_w^{|w|} + m_w^1 + m_w^2 + ... + m_w^{i_1}) \bmod sp = (m_{w'}^{j_n+1} + m_{w'}^{j_n+2} + ... + m_{w'}^{|w'|} + m_{w'}^1 + m_{w'}^2 + ... + m_{w'}^{j_1}) \bmod sp$

Otherwise, the combination is wrong and should be deleted from Set_{Step6}. Let Set_{Step7} denote the set of combinations after executing Step 7.

4.5 Fuzzy Keyword Search Algorithm supporting Access Control

Before describing the fuzzy keyword search algorithm supporting access control (FKSAAC), we first illustrate the algorithm $Decrypt$. Then, we give the search algorithm FKSAAC.

Decryption Algorithm in FKSAAC. Most parts of the decryption algorithm in our method are the same as the decryption algorithm in Bethencourt's method [21]. However a few parts are different. In Bethencourt's method [21], in the trees (e.g. T_{p_D}) which are associated with ciphertexts, the attributes in leafs are stored in the form of plaintext. In our method, in the trees (e.g. $T_{p_D}^*$) which are associated with indexes: (1) If the leafs are associated with characters of keywords, the word pattern values (for example, $F_M(w,i)$, $i = 1,2,...,|w|$) are stored in these leafs (the characters of keywords are not stored because the privacy of keywords should be proctected). (2) If the leafs are associated with attributes, these attributes are stored in these leafs in the form of plaintext (this part is the same as [21]). Thus, before representing the algorithm FKSAAC, we want to briefly illustrate the decryption algorithm $Decrypt$ which has been slightly modified in our method.

$Decrypt$ takes as input an index Idx_D and a trapdoor $Td_{w'}$ of w'. A document policy is embedded in the index Idx_D, and the document policy consists of several error-tolerance policies and an access control policy. If and only if (1) w' satisfies one of the error-tolerance policies in Idx_D and (2) the attributes of u satisfy the access control policy, $Decrypt$ could decrypt Idx_D and output the identity of $\mathcal{D}$. Otherwise, it outputs a meaningless string $\perp$.

First, we introduce the recursive algorithm $DecryptNode(Idx_D, Td_{w'}, x)$ in $Decrypt$, where x is a leaf of $T_{p_D}^*$. Let $i = f(x)$ (i is a character or an attribute associated with x). We should note that: (1) If x is associated with an attribute, $DecryptNode$ could efficiently compute $i = f(x)$. This is because attributes in the leafs of $T_{p_D}^*$ are stored in the form of plaintext (this part is the same as [21]). (2) If x is associated with a character of a keyword, $DecryptNode$ can compute $i = f(x)$ by using word patterns (see **Section 3** and **4.4**), where $i = f(x)$ is a correct corresponding relationship (this part is the difference between our method and [21]). Then, the algorithm $DecryptNode$ computes as follows,

$DecryptNode(Idx_D, Td, x) = \frac{e(D_i, C_x)}{e(D_i', C_x')} = \frac{e(g^{\gamma} \cdot H(i)^{\gamma_i}, h^{q_x(0)})}{e(g^{\gamma_i}, H(i)^{q_x(0)})} = e(g,g)^{\gamma q_x(0)}$. Otherwise, $DecryptNode(Idx_D, Td)$ outputs $\perp$.

The rest parts of $Decrypt$ in our method are the same as [21] and we briefly describe them as follows. When x is a non-leaf node, the algorithm $DecryptNode(Idx_D, Td_{w'}, x)$ proceeds as follows. For all nodes z that are children of x, it calls $DecryptNode$ and stores the output as F_z. Suppose x is associated with the threshold gate $T(n_x, m_x)$. If there exists n_x outputs, which are not $\perp$, $DecryptNode(Idx_D, Td_{w'}, x)$ outputs $e(g,g)^{\gamma q_x(0)}$. When $x = R$ is the root of $T_{p_D}^*$, $DecryptNode(Idx_D, Td_{w'}, R)$ outputs $e(g,g)^{\gamma q_R(0)}$. Recall that, $q_R(0)$ is set to s (see **Section 4.2**). Thus, we have $e(g,g)^{\gamma q_R(0)} = e(g,g)^{\gamma s}$. Then, the algorithm $Decryption$ decrypts Idx_D by computing

$\tilde{C} / (e(C,D) / e(g,g)^{\gamma s}) = (ID_D \| 0^l) e(g,g)^{\alpha s} / (e(h^s, g^{(\alpha+\gamma)/\beta}) / e(g,g)^{\gamma s}) = ID_D \| 0^l$.

Finally, the algorithm $Decrypt$ outputs the identity ID_D of $\mathcal{D}$.

Search Algorithm FKSAAC. For each document, the cloud server runs the algorithm FKSAAC to test whether the searched word fuzzy matches the keywords in a document. FKSAAC takes as input the trapdoor $Td_{w'}$ of a searched word w' and the index Idx_D of an encrypted document $\mathcal{D}$. FKSAAC outputs the identity ID_D of $\mathcal{D}$, if and only if (1) w' satisfies $ed(w_i, w') \le e_{w_i}$ (represented by an error-tolerance policy of $\mathcal{D}$), where w_i is one of the keywords in $\mathcal{D}$, and (2) attributes in $Td_{w'}$ satisfy the access policy of $\mathcal{D}$. Otherwise, FKSAAC outputs false. The search algorithm FKSAAC is described as follows.

Step 1. For each subtree $T_{p_{et}}^{w_i}$ (it represents the keyword w_i) in $T_{p_D}^*$, FKSAAC computes $|w_i|$ according to the number of ciphertext components and computes $|w'|$ according to the number of secret key components in $Td_{w'}$. If $|w_i| - e_{w_i} \not\le |w'|$ or $|w'| \not\le |w_i| + e_{w_i}$ (see **Theorem 1**), FKSAAC aborts the keyword w_i and then executes **Step 1** to test the next keyword in Idx_D. Otherwise, FKSAAC computes $|w'| - |w_i|$: (1) If $|w'| - |w_i| \le 0$, FKSAAC extracts ciphertext components from Idx_D s.t. these ciphertext components are associated with the value $t = 1$; (2) If $|w'| - |w_i| > 0$, FKSAAC extracts ciphertext components from Idx_D s.t. these ciphertext components are associated with the value $t = |w'| - |w| + 1$ (see **Theorem 2**).

Step 2. FKSAAC extracts the word pattern of w_i from the index Idx_D and the word pattern of w' from the trapdoor $Td_{w'}$. Then, FKSAAC could calculate the set Set_{Step7} by executing the optimization approach in **Section 4.4**. Next, FKSAAC finds out which secret key component of an attribute in $Td_{w'}$ corresponds to which ciphertext component of an attribute in Idx_D (these relationships about attributes are easy to be obtained, because they are stored in plaintext). For each combination in Set_{Step7}, FKSAAC extracts the corresponding

relationships in it. Then FKSAAC runs the algorithm $Decrypt$ in our method to try to decrypt Idx_D by using these corresponding relationships and the relationships about attributes: (1) If $Decrypt$ outputs the identity ID_D of $\mathcal{D}$, FKSAAC returns ID_D. (2) If $Decrypt$ outputs $\perp$, FKSAAC tests the next combination in Set_{Step7}. If all the outputs of $Decrypt$ are $\perp$, FKSAAC executes **Step 1** to test the next keyword in Idx_D.

Step 3. FKSAAC returns false.

The cloud server runs FKSAAC to test all the indexes of encrypted documents, and then returns the encrypted documents whose identities have been retrieved to the user u.

5. Experiments

We compare our method FKS-AC with Fuzzy Keyword Search over Encrypted Data in Cloud Computing (FKS) [1] and Privacy-Preserving Multi-Keyword Fuzzy Search over Encrypted Data in the Cloud (PPMKFS) [17]. We also do the comparison works of FKS-AC when choosing different values as sp (sp is the parameter in the word pattern function F_M).

FKS-AC is implemented by using Java Pairing-Based Cryptography Library 2.0.0, which could support the calculations in bilinear groups. In FKS-AC, the character set S_{char} is $\{a, b, ..., z, A, B, ..., Z, -\}$, the attribute set S_{attr} is $\{a_1, a_2, a_3\}$ and the access control policies are "a_1 AND a_3", "a_2 AND a_3", "a_1 OR a_2", etc. In order to compare FKS-AC, PPMKFS and FKS fairly (as PPMKFS and FKS do not support access control, the cloud has to test all the indexes for fuzzy search), we suppose the data owner distributes the attributes a_1, a_2 and a_3 to users. Then, users have the privilege to search all the documents. Thus, FKS-AC also has to test all the indexes of documents after receiving a trapdoor from a user.

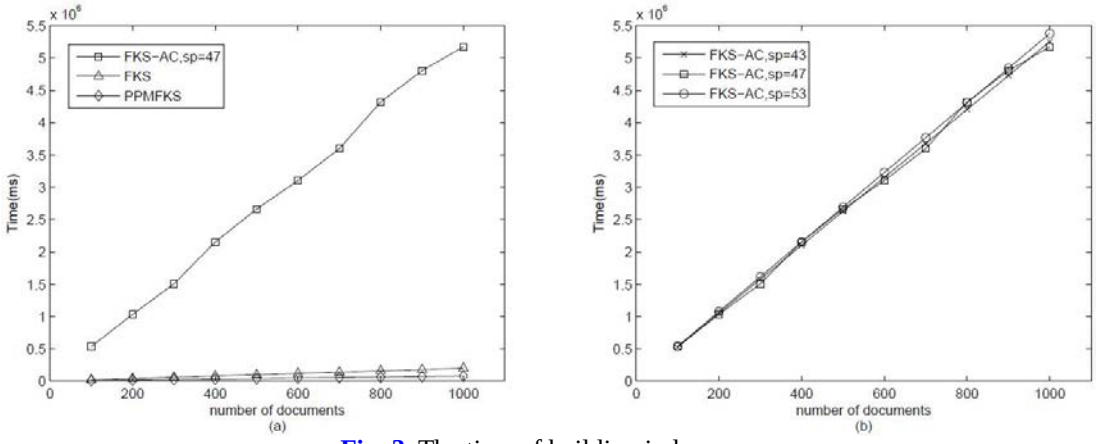

Fig. 3. The time of building indexes

Our experiments run on a win7 computer with four 2.80GHz CPUs and 4G RAM. We randomly extract 400 distinct keywords from the documents in ACM Digital Library. In these keywords, the minimal, maximal and average number of characters are 3, 14 and 8 respectively. The documents is 1000 in total. Each document has 5 keywords, which are randomly chosen from these 400 keywords. Then, we compare the running times of index generation, trapdoor generation and fuzzy keyword search. Given a keyword w, its maximal

error-tolerance value is e_w. In FKS-AC and FKS, we set (1) $e_w = 1$, if $|w| \leq 5$; (2) $e_w = 2$, if $5 < |w| \leq 10$; (3) $e_w = 3$, if $|w| > 10$. To generate a searched word w', we randomly chooses e_w characters as typos, and insert them into w. Thus, the number of characters in w' is $|w| + e_w$ ($|w'| = 4, 5, ..., 17$). In PPMKFS, we set $e_w = 1$ whether w is a long keyword or not. This is because the number of typos allowed by PPMKFS is fixed when LSH has been chosen.

The time of index generation. **Fig. 3 (a)** shows the times of index generation in FKS-AC, FKS and PPMKFS: (i) The index generation times are linear to the number of documents; (ii) FKS and PPMKFS are more efficient than FKS-AC. **Fig. 3 (b)** shows that the times of index generation of FKS-AC when setting the parameter sp to different values.

Analysis of the results. As FKS-AC, FKS and PPMKFS generate index per document, thus the times of index generation are linear to the number of documents. The index generation of FKS is constructed on AES. As AES is a symmetric encryption method and the computing overhead is very low, FKS is very efficient. PPMKFS is constructed on LSH and BF (LSH and BF consist of hash functions). ad of LSH and BF is much less than AES, the index generation of PPMKFS is more efficient than FKS. In order to support access control, FKS-AC is constructed on CP-ABE. CP-ABE is an asymmetric encryption scheme and requires complicated calculations. Thus, FKS-AC spends more time building indexes. Additionally, FKS-AC should calculate the word patterns for keywords. As the computing overhead of word patterns does not increase when choosing different values as sp, the times of index generation of FKS-AC are the same when sp is set to 43, 47 and 53 respectively.

As the data owner generates indexes before outsourcing documents to the cloud, index generation could be seen as the initialization work before providing the search service. Thus, we think the low efficiency of index generation in FKS-AC could be tolerated.

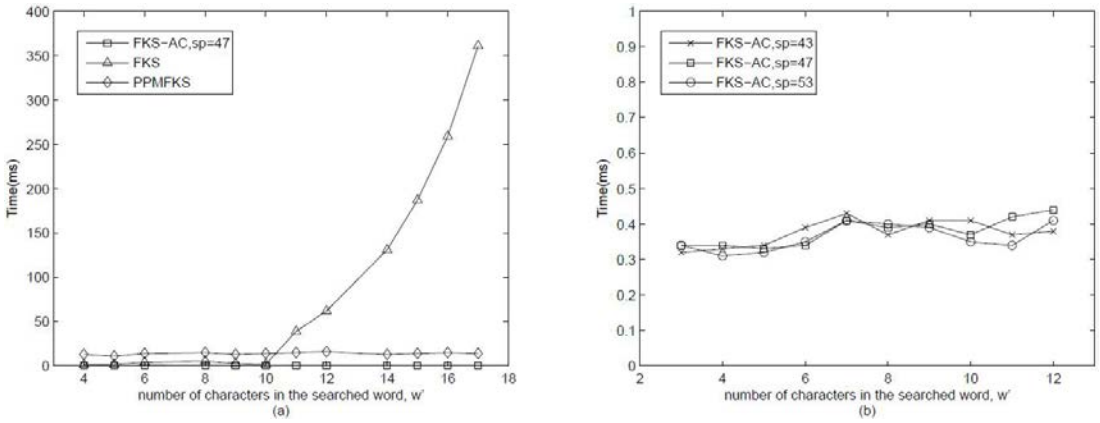

Fig. 4. The time of generating a trapdoor

The time of trapdoor generation. As shown in **Fig. 4 (a)**, FKS-AC is the most efficient, and FKS costs more time for trapdoor generation. From **Fig. 4 (b)**, we can see that the times of FKS-AC are almost the same when sp is set to different values.

Analysis of the results. In FKS-AC, trapdoor is generated on user client. A user only puts some secret key components together according to the character-appearing order of w', and then computes the word pattern of w'. Thus, the trapdoor generation in FKS-AC is very efficient. In PPMKFS, trapdoor is generated by data owner. The data owner generates the

trapdoor for a user by executing LSH and BF. As LSH and BF consist of dozens of hash functions, the trapdoor generation of PPMKFS is slower than FKS-AC. In FKS, the trapdoor is generated on user client. Before generating a trapdoor for a searched word, the user should first generate a fuzzy keyword set. However, a long searched word necessitates to issue a large set whose size is $O(|w'|^{e_{w'}})$ ($e_{w'}=1$ if $w' \le 5$; $e_{w'}=2$ if $5 < w' \le 10$; $e_{w'}=3$ if $w' > 10$). Thus, when $|w'|$ increases, the size of the set increases rapidly. Then, the user encrypts each word in the set, and their ciphertexts are as the trapdoor of the searched word. Thus, the efficiency of trapdoor generation of FKS is the lowest. As shown in **Fig. 4 (b),** when setting sp to different values, the trapdoor generation times of FKS-AC are almost the same. This is because the computing overhead of word pattern does not increase when choosing different values as sp.

The time of fuzzy search. **Fig. 5** shows the average time of search when $|w'| = 4,5,...,17$. From **Fig. 5 (a)**, we can see that PPMKFS is the most efficient. FKS has a better performance than FKS-AC when $|w'| < 10$. FKS-AC is more efficient than FKS when $|w'| > 10$. From **Fig. 5 (b)**, we can see that the search efficiency of FKS-AC could be improved by increasing sp.

Analysis of the results. As PPMKFS performs fuzzy search only by multiplying two groups of vectors (one group of vectors is a trapdoor and the other is the index of a document), PPMKFS is very efficient. However, the search result of PPMKFS is not accurate. This is because PPMKFS is based on BL and LSH. Both BL and LSH introduce **false positives** (a false positive is that, a document should not be in the search result, but it is). Additionally, LSH introduces **false negatives** (a false negative is that, a document should be in the search result, but it is not). As the shortcomings of BL and LSH, PPMKFS can not provide the accurate search results. Compared with PPMKFS, FKS and FKS-AC are accurate methods and do not introduce any false positives or false negatives. We explain the exerimental results (as shown in **Fig. 5**) of FKS and FKS-AC in detail below.

In FKS, each trapdoor consists of all the ciphertexts of possible misspellings of w'. Thus, when $|w'|$ increases, the size of trapdoor increases rapidly. When performing a fuzzy search, as the cloud has to compare each ciphertext in the trapdoor of w' with the indexes of documens, the computing overhead of fuzzy search increases rapidly and inevitably results in a long search time.

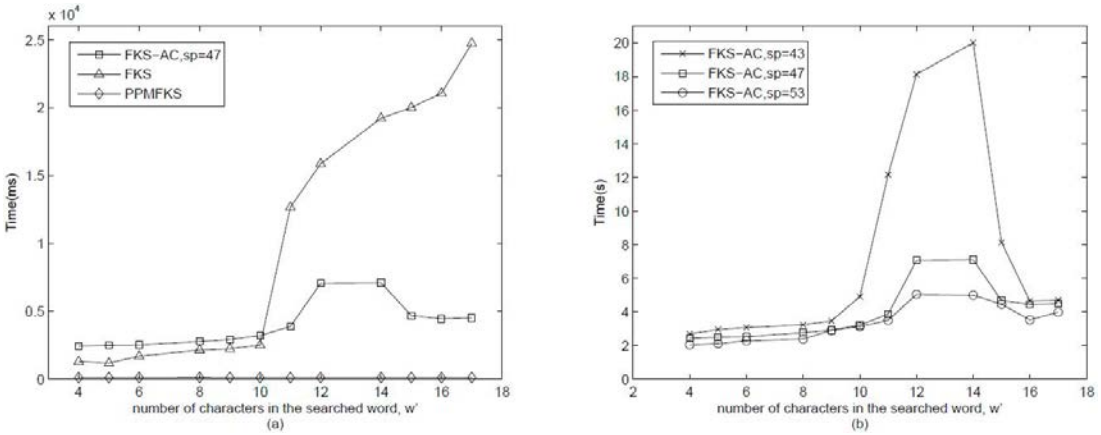

Fig. 5. The average time of fuzzy search

In FKS-AC, if the searched word w' is long, there are more secret key components in the trapdoor of w' which should be tested. Thus, FKS-AC should do more calculations in bilinear groups. Thus, as shown in **Fig. 5 (a)**, the search time of FKS-AC increases when $|w'|$ increases. Now we explain the reason why the efficiency decreases a lot when using the trapdoor of w' ($|w'|=12$ or 14) to perform fuzzy search. Recall that, FKS-AC should compute a set of combinations and try to decrypt indexes using these combinations. In the procedure of fuzzy search, wrong combinations would reduce the search efficiency of FKS-AC. When the searched word w' is very similar to a keyword w and the typos in w' does not exceed e_w, according to **Theorem 2**, there must exist some combinations that could be used to correctly decrypt the indexex of documents which contains w. However, when the searched word w' is very similar to a keyword w, but the typos in w' exceeds e_w, according to **Theorem 2**, all these combinations found by our algorithm FKSAAC are wrong combinations, which would waste a lot of time to try to decrypt indexes. For example, when using $w'=consttructionn$ to search $w_1=construction$ ($e_{w_1}=3$), as the typos in w' does not exceed $e_{w_1}=3$, FKS-AC could find out some right combinations. Then FKS-AC could try to decrypt the index by using these combinations, and the search efficiency would not decrease obviously. However, when using $w'=consttructionn$ to search $w_2=contributions$ ($e_{w_2}=3$), as w' is very similar to w_2, our algorithm could also find some combinations. However, as the number of typos in w' exceeds $e_{w_2}=3$, according to **Theorem 2**, all these combinations found by our algorithm FKSAAC are wrong combinations. As the cloud does not know which combinations are wrong, all these wrong combinations should be used to try to decrypt the indexes of documents which contains w_2. Thus, the search efficiency decreases obviously. Note that, in the above example, w_1 and w_2 look very similar, but the edit distance between them is great than the maximal error-tolerance values allowed by them. In our experiments, the tested keyword are randomly chosen from ACM Digital Library. When $|w'|=12$ or 14, there are many such keywords like w_1 and w_2. For example, "adaptation" and "adsorption", "Automation" and "Automobiles", etc. Thus, the search time of FKS-AC increases a lot when $|w'|=12$ and 14.

From **Fig. 5 (b)**, we can see that, the search efficiency increases with the value of sp. This is because larger sp could help the cloud to find corresponding relationships more accurate and efficient. Namely, a larger sp could reduce the number of wrong combinations found by our algorithm FKSAAC. Thus, the search efficiency increases when sp increases.

6. Security Analysis

6.1 Security Analysis of FKS-AC about Collusion Attack.

In our method, we adopt CP-ABE scheme [21] to generate secret keys for different users. Users use their secret keys to generate trapdoors for searching keywords. As the CP-ABE scheme is collusion-resistance (it has been proofed in [21]). Thus, users' secret key components generated by CP-ABE scheme cannot be colluded to decrypt the ciphertexts which beyond the privileges of users. Additionally, in our method, a user could generate new

secret keys by running the algorithm $Delegate$, and then use the new secret keys to construct trapdoors. As new secret keys and old secret keys can not be colluded (it has been proofed in [21]), the secret key components in the trapdoors from the same user cannot be colluded.

6.2 Security Analysis of FKS-AC in the Known Ciphertext Model.

Known Ciphertext Model [17, 23, 24]: The cloud server can only access (1) the encrypted documents, (2) the indexes, (3) the word patterns of keywords and searched words, (4) the submitted trapdoors, and (5) the search results.

According to the known ciphertext model, if the adversary records the word patterns, trapdoors and search results, the adversary can build up access patterns. Therefore, under the known ciphertext model, nothing beyond the access patterns and the search results should be leaked. In the following paragraphs, w_i denotes a keyword and w_i' denotes a searched word. We use the notation $S_{w_i,e_{w_i}}$ to denote the collection of w_i' satisfying $ed(w_i, w_i') \le e_{w_i}$, where $ed(w_i, w_i')$ denotes the edit distance between w_i and w_i', and e_{w_i} denotes the the maximal error-tolerance value of w_i. We adapt the definitions in [4, 17, 25] for our proofs.

Definition 7. Search Pattern (π): Let $Q = \{w_1', ..., w_n'\}$ be the set of searched words for n consecutive queries, then π be a binary matrix s.t. $\pi[i, j] = 1$ if $w_i' \in S_{w_i,e_{w_i}}$ and $w_j' \in S_{w_i,e_{w_i}}$, otherwise $\pi[i, j] = 0$.

Definition 8. Access Pattern (A_p): Let $D(w_i')$ ($w_i' \in S_{w_i,e_{w_i}}$) be a collection that contains the identities of documents which contain the keyword w_i. Let $T = \{T_1, ..., T_n\}$ be the trapdoors for the query set $Q = \{w_1', ..., w_n'\}$. Then, Access Pattern for the n trapdoors is defined as $\{A_p(T_1) = D(w_1'), ..., A_p(T_n) = D(w_n')\}$.

Definition 9. History (H_n): Let D be the document collection and $Q = \{w_1', ..., w_n'\}$ be the searched words for n consecutive queries. Then, $H_n = (D, Q)$ is defined as a n-query History.

Definition 10. Trace (γ): Let $C = \{C_1, ..., C_l\}$ be the collection of encrypted documents, $id(C_i)$ be the identity of C_i, $|C_i|$ be the size of C_i, P_{w_i} be the word pattern of w_i, $P_{w_i'}$ be the word pattern of w_i', $S_p(H_n)$ be the Search Pattern of H_n and $A_p(H_n)$ be the Access Pattern of H_n. Then, $\gamma(H_n) = \{(id(C_1), ..., id(C_l))$, $(|C_1|, ..., |C_l|)$, $(P_{w_1}, ..., P_{w_n})$, $(P_{w_1'}, ..., P_{w_n'})$, $S_p(H_n)$, $A_p(H_n)\}$ is defined as the trace of H_n. Trace is the maximum amount of information that a data owner allows to leak to an adversary.

Definition 11. View (V): Let $C = \{C_1, ..., C_l\}$ be the collection of encrypted documents, $id(C_i)$ be the identity of C_i, I be the collection of indexes of C, $P_w = (P_{w_1}, ..., P_{w_n})$ be the collection of word patterns of keywords, $P_{w'} = (P_{w_1'}, ..., P_{w_n'})$ be the collection of word patterns of searched words, and $T = \{T_1, ..., T_n\}$ be the collection of trapdoors. Then, $V(H_n) = \{(id(C_1), ..., id(C_l)), C, I, P_w, P_{w'}, T\}$ is defined as the view of H_n. View is the information that is accessible to an adversary.

We adopt a similar simulation based proof, which is widely used in [4, 17]. Intuitively, given two histories with the same trace, if the adversary cannot distinguish which of them is generated by the simulator, the adversary cannot learn additional information about the index, trapdoors and the encrypted documents beyond the search result and the access pattern [17].

Theorem 3. FKS-AC is secure under the known ciphertext model.

Proof. The notation S denotes the simulator, which can simulate a view V^*. V^* is indistinguishable from an adversary's view $V(H_n)=\{(id(C_1),...,id(C_l)),C,I,P_w,P_{w'},T\}$. To achieve this, the simulator S does the followings:

(1) Identities of documents are available in the trace. Thus, S can copy these identities, that is, $\{id(C_1)^*=id(C_1),...,id(C_l)^*=id(C_l)\}$. As identity lists of the adversary's view V and the simulated view V^* are the same, they are computationally indistinguishable.

(2) S chooses l random values $\{C_1^*,...,C_l^*\}$, s.t. $|C_1^*|=|C_1|,...,|C_l^*|=|C_l|$. The documents are encrypted by using a secure encryption scheme (e.g. AES). Thus, the outputs of the secure encryption scheme is computationally indistinguishable from random values. Hence, C_i^* and C_i are computationally indistinguishable.

(3) S runs the algorithm $Setup$ to obtain a public key PK and a master key MK, and then, S runs the algorithm $KeyGen$ to obtain a secret key SK.

(4) S constructs n consecutive queries $Q^*=\{w_1'^*,...,w_n'^*\}$, the word patterns $P_{w'^*}=(P_{w_1'^*},...,P_{w_n'^*})$, and the trapdoors $T^*=\{T_1^*,...,T_n^*\}$. For each $w_i'\in Q$, $1\le i\le n$, S generates the searched word $w_i'^*$ randomly, s.t. $|w_i'^*|=|w_i'|$. Then, S computes the word pattern of $w_i'^*$. As $w_i'^*$ is generated randomly, the word pattern of $w_i'^*$ is computationally indistinguishable from random values. Note that, the searched word w_i' may have typos, and additionally, w_i' has been inserted several random characters at random positions (see the trapdoor in **Section 4.3**). Thus, the word patterns of $w_i'^*$ and w_i' are computationally indistinguishable. According to the characters in $w_i'^*$, S generates the trapdoor T_i^* for $w_i'^*$ by utilizing the secret key components in SK. As the secret key SK is indistinguishable from random values [21], the trapdoor T_i^* generated by utilizing SK is indistinguishable from random values. For the same reason, the trapdoor T_i for w_i' is also indistinguishable from random values. Hence, the trapdoors T_i^* and T_i are computationally indistinguishable.

(5) For each C_i^*, S sets an empty set $Set_{C_i^*}$, where $1\le i\le l$. According to the access pattern A_p, if $id(C_i)$ could be retrieved by using the word w_j', then S adds $w_j'^*$ to $Set_{C_i^*}$, where $1\le j\le n$. The set $Set_{C_i^*}$ is as the keyword set of the encrypted document C_i^*. Next, S constructs the document policy for C_i^* by using the keywords in $Set_{C_i^*}$, and runs the algorithm $Encrypt$ to generate the index $I_{C_i^*}$ for C_i^*. The ciphertext

generated by $Encrypt$ is as the index $I_{C_i^*}$ of C_i^*. As the ciphertext is indistinguishable from random values [21], the index $I_{C_i^*}$ is indistinguishable from random values. For the same reason, the index I_{C_i} generated by $Encrypt$ is also indistinguishable from random values. So I_{C_i} and $I_{C_i^*}$ are computationally indistinguishable. Then, the index collection I for $\{C_i \mid 1 \le i \le l\}$ and I^* for $\{C_i^* \mid 1 \le i \le l\}$ are computationally indistinguishable.

Since each item of V and V^* are computationally indistinguishable, we have the onclusion that FKS-AC satisfies the security definition presented in **Theorem 3**. □

7. Related Work

Li et al. [1] first propose a searchable encryption method supporting fuzzy keyword search. For each keyword, data owner use the wildcard technique to build a fuzzy keyword set which contains all the possible misspellings. The index and trapdoor are built on the set. To perform a search, the cloud checks whether there is intersection between the index and the trapdoor. To limit the size of the set, Liu et al. [14] propose a method which is based on a predefined dictionary. The dictionary is as a filter to delete the meaningless words in a user's search. However, this method requires that a user should know much about the filed he/she queries. Kuzu et al. [4] propose a generic similarity search method based on Bloom Filter (BF) and Locality-Sensitive Hashing (LSH) [19, 26]. A LSH function hashes close items to the same hash value with higher probability than the items that are far apart. Thus the similarity of the keywords could be measured by using LSH functions. According the hash values of keywords, the data owner builds indexes using BF. Thus, the indexes could support fuzzy keyword search. In [17], Wang et al. propose a multi-keyword fuzzy search method. This method is also based on BF and LSH. However, as LSH can not hash close items to the same hash value with the probability 1, LSH inevitably results the in inaccurate search results.

References

[1] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *Proc. of INFOCOM 2010 IEEE*, pp. 1–5. IEEE, 2010. Article (CrossRef Link)

[2] L. Zhou, D. Wu, B. Zheng, and M. Guizani, "Joint physical-application layer security for wireless multimedia delivery," *Communications Magazine IEEE*, 52(3):66–72, 2014. Article (CrossRef Link)

[3] L. Zhou, H. C. Chao, and A. V. Vasilakos, "Joint forensics-scheduling strategy for delay-sensitive multimedia applications over heterogeneous networks," *IEEE Journal on Selected Areas in Communications*, 29(7):1358–1367, 2011. Article (CrossRef Link)

[4] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *Proc. of 2012 IEEE 28th International Conference on Data Engineering*, pp. 1156–1167, IEEE, 2012. Article (CrossRef Link)

[5] Z. Fu, K. Ren, J. Shu, X. Sun, and F. Huang, "Enabling personalized search over encrypted outsourced data with efficiency improvement," *IEEE Transactions on Parallel and Distributed Systems*, 27(9):2546–2559, 2016. Article (CrossRef Link)

[6] Z. Fu, F. Huang, X. Sun, A. Vasilakos, and C. N. Yang, "Enabling semantic search based on conceptual graphs over encrypted outsourced data," *IEEE Transactions on Services Computing*, PP(99):1–1, 2016. Article (CrossRef Link)

[7] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, 27(2):340–352, 2016. Article (CrossRef Link)
[8] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Transactions on Parallel & Distributed Systems*, 25(1):829–837, 2011. Article (CrossRef Link)
[9] H. H. Yong and P. J. Lee, "Public key encryption with conjunctive keyword search and its extension to a multi-user system," in *Proc. of International Conference on Pairing-Based Cryptography*, pp. 2–22, 2007. Article (CrossRef Link)
[10] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proc. of Theory of Cryptography Conference*, pp. 535–554. Springer, 2007. Article (CrossRef Link)
[11] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *Proc. of International Conference on Applied Cryptography and Network Security*, pp. 31–45, Springer, 2004. Article (CrossRef Link)
[12] E. Shen, E. Shi, and B. Waters, "Predicate privacy in encryption systems," in *Proc. of Theory of Cryptography Conference*, pp. 457–473, Springer, 2009. Article (CrossRef Link)
[13] N. Attrapadung and B. Libert, "Functional encryption for inner product: Achieving constant-size ciphertexts with adaptive security or support for negation," in *Proc. of International Workshop on Public Key Cryptography*, pp. 384–402, Springer, 2010. Article (CrossRef Link)
[14] C. Liu, L. Zhu, L. Li, and Y. Tan, "Fuzzy keyword search on encrypted cloud storage data with small index," in *Proc. of 2011 IEEE International Conference on Cloud Computing and Intelligence Systems*, pp. 269–273, IEEE, 2011. Article (CrossRef Link)
[15] M. Chuah and W. Hu, "Privacy-aware bedtree based solution for fuzzy multi-keyword search over encrypted data," in *Proc. of 2011 31st International Conference on Distributed Computing Systems Workshops*, pp. 273–281, IEEE, 2011. Article (CrossRef Link)
[16] J. Wang, H. Ma, Q. Tang, J. Li, H. Zhu, S. Ma, and X. Chen, "Efficient verifiable fuzzy keyword search over encrypted data in cloud computing," *Computer Science & Information Systems*, 10(2):667–684, 2013. Article (CrossRef Link)
[17] B. Wang, S. Yu, W. Lou, and Y. T. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *Proc. of IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pp. 2112–2120, IEEE, 2014. Article (CrossRef Link)
[18] Z. Fu, X. Wu, C. Guan, X. Sun, and K. Ren, "Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement," *IEEE Transactions on Information Forensics and Security*, 11(12):2706–2716, 2016. Article (CrossRef Link)
[19] A. Gionis, P. Indyk, R. Motwani, et al., "Similarity search in high dimensions via hashing," *VLDB*, volume 99, pp. 518–529, 1999. Article (CrossRef Link)
[20] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, 13(7):422–426, 1970. Article (CrossRef Link)
[21] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. of 2007 IEEE symposium on security and privacy (SP'07)*, pp. 321–334, IEEE, 2007. Article (CrossRef Link)
[22] E. Shi, J. Bethencourt, T. H. Chan, D. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *Proc. of 2007 IEEE Symposium on Security and Privacy (SP'07)*, pp. 350–364, IEEE, 2007. Article (CrossRef Link)
[23] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," in *Proc. of Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*, pp. 253–262, IEEE, 2010. Article (CrossRef Link)
[24] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis, "Secure knn computation on encrypted databases," in *Proc. of the 2009 ACM SIGMOD International Conference on Management of data*, pp. 139–152, ACM, 2009. Article (CrossRef Link)
[25] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," *Journal of Computer Security*, 19(5):895–934, 2011. Article (CrossRef Link)

[26] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proc. of the thirtieth annual ACM symposium on Theory of computing*, pp. 604–613, ACM, 1998. Article (CrossRef Link)

Zhuolin Mei received his Ph.D. degree from Huazhong University of Science and Techn ology in 2017. He is currently a lecturer with the School of Information Science and Technology, Huizhou University. His research interests include cloud computing, data security, data search and access control.

BinWu received his Ph.D. degree from Huazhong University of Science and Technology in 2017. His research interests include database security and cloud security.

Shengli Tian received his Ph.D. Degree from Huazhong University of Science and Technology in 2014. He is currently a lecturer with the School of Information Engineering, Xuchang University. He research interests include information security technology and machine learning, etc.

Yonghui Ruan received his Ph.D. degree from Huazhong University of Science and Technology in 2015. He is currently a lecturer with the Department of Information Science and Technology, Wenhua College. His research interests include cloud computing, and big data technology.

Zongmin Cui received his Ph.D. Degree from Huazhong University of Science and Technology in 2014. He is currently an associate professor with the School of Information Science and Technology, Jiujiang University. His research interests include cloud computing, data security, publish/subscribe system and data query.