JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 35, 137-158 (2019)
DOI: 10.6688/JISE.201901_35(1).0008

Chinese Multi-Keyword Fuzzy Rank Search over Encrypted Cloud Data Based on Locality-Sensitive Hashing

YANG YANG[1,2,3], YU-CHAO ZHANG[1,2], JIA LIU[1,2], XI-MENG LIU[1,2], FENG YUAN[4] AND SHANG-PING ZHONG[1,2,+]
[1]*College of Mathematics and Computer Science*
[2]*University Key Laboratory of Information Security of Network Systems*
Fuzhou University
Fuzhou, 350116 P.R. China
[3]*Fujian Provincial Key Laboratory of Information Processing and Intelligent Control*
Minjiang University
Fujian, 350108 P.R. China
[4]*Information Security Institute*
Beijing Electronic Science and Technology Institute
Beijing, 100070 P.R. China
E-mail: yang.yang.research@gmail.com; spzhong@fzu.edu.cn

Most of the existing Chinese keyword fuzzy searchable encryption schemes realize fuzzy keyword search utilizing the wildcard and gram methods to construct the fuzzy set, which consumes a lot of storage and computation overheads. In this paper, we propose a novel Chinese multi-keyword fuzzy rank searchable encryption scheme, which achieves efficient fuzzy keyword search without constructing a large fuzzy set. First, the Chinese keyword is converted to the pinyin string, which is partitioned based on unigram, or the mandarin consonant, vowel and tone of pinyin. Then, we design two Chinese keyword vector generation algorithms to convert a pinyin string into a keyword vector. Moreover, the locality-sensitive hashing and Bloom filter are utilized to construct the fuzzy keyword search algorithm. We design two schemes to realize the Chinese fuzzy multi-keyword search, and all of them utilize a single Bloom filter as the encryption index of a document. The cloud storage server only needs to add (or delete) an encrypted file and its encrypted index to realize the dynamic update of the files. To improve the accuracy of the rank, a three-factor rank algorithm is proposed. The theoretical analysis and experimental results indicate that the proposed schemes realize Chinese multi-keyword fuzzy search, more accurate search result rank, guarantee the data security, and save a large amount of storage and computation costs.

Keywords: searchable encryption, cloud computing, fuzzy Chinese keyword, locality-sensitive hashing, security

1. INTRODUCTION

With the rapid development of cloud computing [1], a large amount of sensitive data is stored in the cloud. The cloud server provides high-quality data storage services and reduces the data storage and maintenance overheads at the user side. In order to protect the data privacy, it is a common solution to encrypt data and then upload them into cloud server for outsourced storage. However, the unreadability of the ciphertext also hinders the data usability [2], such as the commonly used keyword search function. If the

Received July 6, 2017; revised October 15, 2017 & January 7, 2018; accepted February 20, 2018.
Communicated by Berlin Chen.
[+] Corresponding author.

quantity of stored data is not large, the user can download all the data to the local computer, decrypt them and then operate keyword search on the plaintext documents. With the increase of the outsourced data, this method is obviously not applicable. It is a tough task to search over a large amount of encrypted data.

Song *et al.* [3] put forth the concept of searchable encryption to deal with the problem of keyword search over encrypted data. To improve the search efficiency, Chang *et al.* [4] built an index for each document to facilitate the data retrieval. Wang *et al.* [5, 6] proposed a rank searchable encryption scheme to support the single-keyword search. It calculates the $tf - idf$ (term frequency − inverse document frequency) of the keyword in the document [7], which is encrypted by the order-preserving encryption (OPE) methodology [8, 9]. Then, rank the match files according to the encrypted $tf - idf$ values, and return the top-k documents to users, which are the most relevant documents. Hong *et al.* [30] proposed a multi-keyword searchable encryption scheme, which builds multi-keyword search indices at the server side. Cao *et al.* [10] constructed a new multi-keyword rank searchable encryption scheme using the vector space model [11] and secure KNN (k-nearest neighbor) algorithm [12]. The scheme encrypted the index vector and the query vector with two different matrixes. Then, the generated encrypted indexes vector and the encrypted query vector are multiplied using the inner product algorithm, and the multiplication product is the relevance score, which is utilized in the rank process. However, these schemes only support accurate keyword search and the queried keyword must be exactly the same as the predefined keyword in the encrypted index, which is obviously not quite practical.

Li *et al.* [13] proposed a fuzzy search scheme, which constructs the keyword fuzzy sets using the gram method. Later, Li *et al.* [14] proposed a fuzzy keyword searchable encryption scheme, which constructs the keyword fuzzy set with wildcards. If the queried keywords contain spelling mistake, the scheme can also find out the match result, which improves the users' search experience. Wang *et al.* [15] utilized the wildcards and index trees to design fuzzy search scheme. Chuah and Hu [16] constructed a multi-keyword fuzzy search scheme based on BedTree to improve the search efficiency.

But the above schemes are aimed to achieve fuzzy search of English keyword, because Chinese characters are typical non-letter language and words are flexible and diverse, so the above schemes do not apply to Chinese keyword fuzzy search. Cao *et al.* [17] proposed a plaintext fuzzy search scheme based on Chinese pinyin, but it cannot achieve the search on ciphertext. Chen *et al.* [18] used alphabet-based string similarity measure to achieve ciphertext fuzzy search scheme of Chinese keywords.

However, the above fuzzy searchable encryption schemes must pre-construct the fuzzy sets, which consumes a large storage space in the cloud. For example, in the wildcard based fuzzy keyword set construction method, the size of fuzzy keyword set exponentially increases with the edit distance, which consumes a lot of computation and storage overheads. Yang *et al.* [19] utilized n-gram method to deal with keywords and generated the simhash fingerprint of the fuzzy keyword set using simhash algorithm, which greatly reduces the storage space and computation time. Wang *et al.* [20] and Fu *et al.* [21] combined LSH (locality-sensitive hashing) [22] and secure KNN (k-nearest neighbor) methods to design a new multi-keyword fuzzy searchable encryption scheme. Although the above schemes do not need to construct the fuzzy keyword sets, they are designed for English fuzzy keyword search rather than Chinese fuzzy search.

In this paper, we construct a novel Chinese multi-keyword fuzzy rank searchable encryption scheme. The main contributions are summarized as follows.

(1) **Novel Chinese fuzzy multi-keyword search** We propose a Chinese fuzzy multi-keyword rank searchable encryption scheme without pre-constructed fuzzy keyword set. A Chinese keyword is converted into the corresponding pinyin string. Then, two vector generation algorithms are designed to transform the keyword pinyin string into keyword vector: Chinese keyword vector generation algorithm based on pinyin string (see section 4.1 for the details), Chinese keyword vector generation algorithm based on the unigram (see section 5.2 for the details). Then, build a Bloom filter as the encrypted keyword index for each document and insert the keyword vectors (in the document) into Bloom filter using the LSH function. If the input values of the LSH function are similar, the outputs are equal with a high probability. Using this characteristic, the authorized user can get the match results with high probability even though the query keyword contains some spelling errors. Thus, our scheme realizes fuzzy Chinese multi-keyword rank search.
(2) **Efficient keyword index storage** This scheme does not need to construct a large pre-constructed keyword fuzzy set. We only need to convert a keyword to a vector, and then insert the keyword vectors into the Bloom filter using the LSH function. The keyword index of each document is a Bloom filter, which greatly reduces the computation and storage overhead.
(3) **Return more accurate rank result** In this scheme, the weighted zone score is introduced for more precise match result rank, which gives keyword different weights according to its occurrence domains in the document. The weighted zone score of keyword, Euclidean distance and keyword frequency are combined to realize three-factor ranking algorithm, which realizes more accurate rank.
(4) **Supporting dynamic document update** The other existing schemes [5, 6, 23-26] utilize $tf-idf$ value in the keyword index encryption algorithm, which is influenced by the document updates such that these schemes [5, 6, 23-26] cannot support dynamic document updates. In our scheme, we utilize term frequency instead of $tf-idf$ to avoid the influence brought by the document update. When an encrypted document is inserted (or deleted), the other encrypted keywords indexes are not influenced.
(5) **Simulation demonstrates the efficiency of the scheme** Extensive simulations are done to evaluate the performance of our scheme, which show that this scheme realizes the fuzzy Chinese multi-keyword rank search based on LSH, has better efficiency and returns more accurate rank results.

2. SYSTEM AND THREAT MODEL

2.1 System Model

The system model is shown in Fig. 1, which consists of the following parties. The main notations used in this paper are shown in Table 1.

(1) **Data owner** is responsible to generate encrypted keyword index and encrypted documents, which are outsourced to the cloud server for secure storage. The data owner extracts the keyword set $W = (w_1, w_2, \ldots, w_n)$ from the document, utilizes Chinese keyword vector generation algorithm to convert the keywords into vectors, generates the encrypted index $Enc_{SK}(\mathcal{I})$ and encrypts the plaintext document set $F = (f_1, f_2, \ldots, f_m)$ into ciphertext set $C = (c_1, c_2, \ldots, c_m)$. Finally, the data owner sends the ciphertext set C and the encrypted index $Enc_{SK}(\mathcal{I})$ to cloud server.

(2) **Authorized data user** submits search requests to the cloud server to issue keyword search query. The authorized data user figures out the query keyword set $Q = (q_1, q_2, \ldots, q_n)$, converts the query keywords into vectors using Chinese keyword vector generation algorithm, and then constructs the trapdoor T_Q. Then, the generated trapdoor $Enc_{SK}(T_Q)$ is submitted to cloud server for multi-keyword rank search.

(3) **Cloud server** is responsible to store data owners' encrypted files with encrypted indexes, and responds to data users' search request. Receiving the query trapdoor $Enc_{SK}(T_Q)$ from the authorized data user, the cloud server calculates the relevance score of the encrypted index $Enc_{SK}(\mathcal{I})$ and trapdoor $Enc_{SK}(T_Q)$. Then, the cloud server ranks the search result and returns top-k most relevant results to the data user according to the calculated relevance scores.

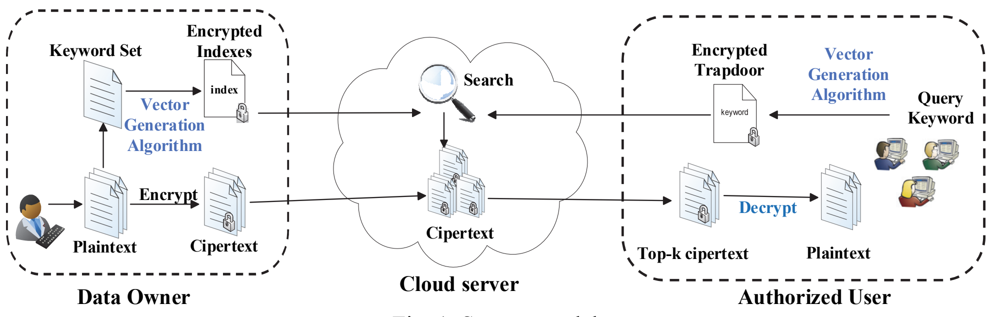


Fig. 1. System model.

Table 1. Notations.

Symbols	Notations	Symbols	Notations
f_i/c_i	Plaintext/ciphertext documents	$\vec{Q} = (\vec{q}_1, \vec{q}_2, \ldots, \vec{q}_\eta)$	Query keyword vector set of file Q
$W = (w_1, w_2, \ldots, w_n)$	Keyword set	p_i/y_i	Syllable segmentation set of keyword w_i/q_i
$\vec{w}_i, \vec{q}_i$	Keyword vector of w_i/q_i	$\vec{W}_i = (\vec{w}_1, \vec{w}_2, \ldots)$	Keyword vector set of file f_i
$\mathcal{I} = (I_1, I_2, \ldots, I_m)$	Index set	$Q = (q_1, q_2, \ldots, q_\eta)$	Query word set

2.2 Threat Model

Assume that cloud server is "honest-but-curious", who is honest to perform the search operation for the authorized data user, returns the match results, and does not delete the encrypted index and ciphertext stored by data owners. Cloud server is also curious to make statistical analysis on the search query and search result to get some additional information about keywords and the plaintext of the stored encrypted documents. This paper focuses on the known ciphertext model [20, 21, 26], which assumes that cloud server can only access the encrypted documents, the secure indexes and the submitted trapdoors. The cloud server also records the search results.

3. PRELIMINARIES

3.1 Weighted Zone Core

The title, abstract and body in a document are usually considered as different domains [7]. Keywords in different domains usually have different importance. Keywords in the title are the most important, keywords in the abstract are the second important, and keywords in the text are the least important. Assume that each document has φ domains and the corresponding weight coefficients are $g_1, \ldots, g_\varphi \in [0,1]$ which satisfies $\sum_{i=1}^{\varphi} g_i = 1$. Let $\upsilon_i = 1$ denote that a keyword appears in the ith domain; otherwise, $\upsilon_i = 0$. The weighted zone score of a keyword in a document is defined as $Z = \sum_{i=1}^{\varphi} g_i \upsilon_i$.

3.2 Locality Sensitive Hashing

Locality sensitive hashing (LSH) functions [22] hash close items to the same hash values with high probability. A hash function family H is (r_1, r_2, p_1, p_2)-sensitive if any two items x, y and $h \in H$, $r_1 < r_2$, $p_1 > p_2$ satisfy: if $d(x, y) \leq r_1$, $Pr[h(x) = h(y)] \geq p_1$; if $d(x, y) \geq r_2$, $Pr[h(x) = h(y)] \leq p_2$; where $d(x, y)$ is the distance between x and y, *e.g.* Euclidean distance.

3.3 Bloom Filter

Bloom filter [27] is an efficient data structure which can quickly determine whether an element belongs to a collection. A Bloom Filter is a λ-bit array, which is initially set to 0 in all positions. For a given set $S = \{a_1, a_2, \ldots, a_n\}$, it uses l-independent hash functions $H = \{h_\sigma \mid h_\sigma: \{0, 1\}^* \rightarrow [1, \lambda], \sigma \in [1, l]\}$ to map each element $a_i \in S$ into the λ-bit array by setting the value of the position to be 1. To check whether an element q is in the set S, we map it using the hash functions $H = \{h_\sigma \mid h_\sigma: \{0, 1\}^* \rightarrow [1, \lambda], \sigma \in [1, l]\}$ to get the l positions. If any value of the l position is 0, $q \notin S$; otherwise, $q \in S$ or q yields a false positive. As shown in Fig. 2, the keyword "searchable" is mapped to Bloom filter, where $\lambda = 20$, $l = 4$. The values of the four positions $P = \{2, 8, 14, 17\}$ in the Bloom filter are set to be 1.

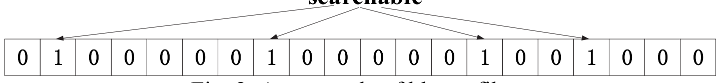


Fig. 2. An example of bloom filter.

4. BASIC CHINESE MULTI-KEYWORD FUZZY RANK SEARCH SCHEME

In this section, we propose a **b**asic **C**hinese **m**ulti-keyword fuzzy rank **s**earch scheme (BCMS), which utilizes Chinese keyword vector generation algorithm based on pinyin string to convert a pinyin string into a keyword vector. Moreover, the LSH function and Bloom filter are utilized to realize the fuzzy keyword search algorithm. The BCMS scheme supports dynamic document update and saves a large amount of storage and computation costs.

4.1 Chinese Keyword Vector Generation Algorithm based on Pinyin String

If pinyin string of a Chinese keyword is considered as an English letter string, the inserting, deleting and replacing of letters in the pinyin string may result in an invalid one. The edition operation over a pinyin string is defined as follows. (1) Only the consonants or vowels in the same syllable change; (2) The consonants and vowels in the same syllable simultaneously change; (3) The mandarin tone changes.

Firstly, Chinese keyword vector generation algorithm converts a Chinese keyword into a pinyin string, which is separated to consonant, vowel and tone. The structure of a keyword vector is shown in Fig. 3. This scheme uses a 63-bit vector to represent a keyword. The first 23 bits represent 23 mandarin consonants in Chinese, the middle 24 bits represent 24 vowels, and the last 16 bits represent the word position and tone (it has the format “ab”, where “a” represents the 1st, 2nd, 3rd, 4th word in the keyword and “b” represents four Chinese tones in pinyin). If the element exists in the pinyin string, the corresponding position is set to 1; otherwise, the position is 0. For example, the syllable segmentation set of the keyword “实验” is {sh, i, 12, y, an, 24}, where “12” indicates that the 1st word “实” in the keyword “实验” is 2nd tone, “24” indicates that the 2nd word “验” in the keyword “实验” is 4th tone.

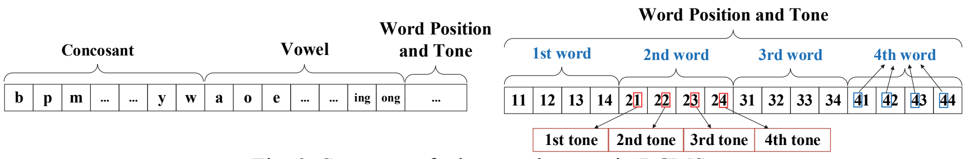
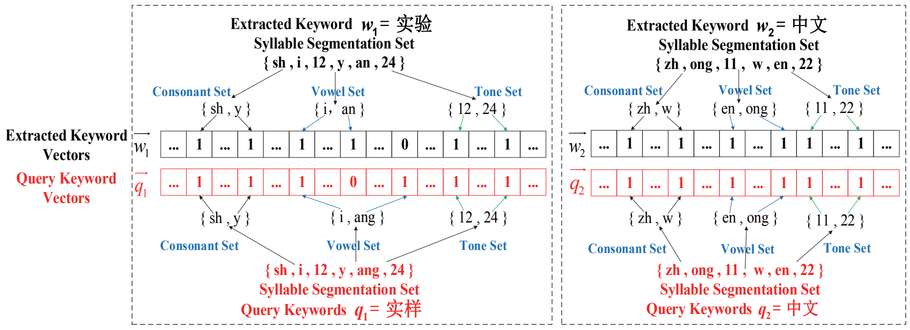


Fig. 3. Structure of a keyword vector in BCMS.


Extracted Keyword w_1= 实验
Syllable Segmentation Set
{ sh , i , 12 , y , an , 24 }
Consonant Set
Vowel Set
Tone Set
{ sh , y }
{ i , an }
{ 12 , 24 }
Extracted Keyword Vectors
Query Keyword Vectors
$\vec{w}_1$
$\vec{q}_1$
{ sh , y }
{ i , ang }
{ 12 , 24 }
{ sh , i , 12 , y , ang , 24 }
Syllable Segmentation Set
Query Keywords q_1= 实样
Extracted Keyword w_2= 中文
Syllable Segmentation Set
{ zh , ong , 11 , w , en , 22 }
{ zh , w }
{ en , ong }
{ 11 , 22 }
$\vec{w}_2$
$\vec{q}_2$
{ zh , ong , 11 , w , en , 22 }
Query Keywords q_2= 中文
Fig. 4. An example of BCMS.

As shown in Fig. 4, assume that the user wants to query the keyword “实验”. Due to the spelling mistake, he enters the query keyword “实样” by substituting the mandarin vowel “an” with “ang” (In Chinese, the confusion of the former nasal vowel and the latter nasal vowel happens with high probability). In traditional hash algorithms, if a pinyin string of keyword is misspelled, the output of the hash function will be completely different from the original output. The vectors $\vec{w}_i$ and $\vec{q}_i$ in this scheme are processed by the LSH function to insert them into Bloom filter, which outputs the keyword index and query trapdoor, respectively. According to the characteristics of the LSH function, the hash values are equal with high probability when the inputs are similar. The Euclidean distance between the vector of keyword “实验” and the vector of keyword “实样” is only $\sqrt{2}$. Their hash values are equal with high probability after being processed by the LSH function. Although the data user inputs a misspelled query keyword “实样”, our scheme can still find the match result related to the keyword “实验”.

4.2 Scheme Construction

The concrete algorithms of BCMS are described as below. (Fig. 5 shows an example of keyword index and trapdoor construction.

(1) **Setup:** The keyword set $W = (w_1, w_2, \ldots, w_n)$ is extracted from the plaintext files $F = (f_1, f_2, \ldots, f_m)$.
(2) **KeyGen(λ):** On input a security parameter λ, this algorithm generates a vector S and two invertible matrices $\{M_1, M_2\}$, where $S \in \{0, 1\}^{\lambda}$ and $M_1, M_2 \in \{0, 1\}^{\lambda\times\lambda}$. The secret key SK consists of the triple $\{S, M_1, M_2\}$. Then, data owner generates a symmetric key sk to encrypt the documents.
(3) **BuildIndex(F, SK, l):** Data owner selects l LSH functions from the LSH family $H = \{h_{\sigma}: \{0, 1\}^{63} \to \{0, 1\}^{\lambda}\}$. For each document, build a λ-bit length Bloom filter as the keyword index. The keyword index set is denoted as $\mathcal{I} = (I_1, I_2, \ldots, I_m)$. The generation process is described as below: (a) Use Chinese keyword vector generation algorithm based on pinyin string to generate the keyword vector $\vec{w}_j$ for each keyword w_j. The keyword vector set of the document f_i is $\vec{W}_i = (\vec{w}_1, \vec{w}_2, \ldots)$; (b) Each keyword vector $\vec{w}_j$ is hashed using the LSH functions $h_{\sigma} \in H (1 \le \sigma \le l)$, which are inserted into Bloom filter I_i.
(4) **EncIndex($\mathcal{I}$, SK):** According to the vector S, the index I_i is split into I_i' and I_i''. If the jth bit of S is 0, set $I_i'[j] = I_i''[j] = I_i[j]$; if the jth bit of S is 1, $I_i'[j]$ and $I_i''[j]$ are set to be random values such that $I_i'[j] + I_i''[j] = I_i[j]$. Then, encrypt I_i' and I_i'' with key SK and obtain $Enc_{SK}(I_i) = \{M_1^T I_i', M_2^T I_i'', Enc_{SK}(\mathcal{I}) = (Enc_{SK}(I_1), Enc_{SK}(I_2), \ldots, Enc_{SK}(I_m))$. Finally, the data owner uploads the encrypted keyword index set $Enc_{SK}(\mathcal{I})$ to the cloud server.
(5) **EncFile (F, sk):** Data owner uses a symmetric encryption algorithm to encrypt the document set $F = (f_1, f_2, \ldots, f_m)$ to the ciphertext set $C = (c_1, c_2, \ldots, c_m)$, and uploads them to the cloud server.
(6) **Trapdoor(Q, SK, l):** When an authorized user makes a search query, he specifies a query keyword set $Q = (q_1, q_2, \ldots, q_{\eta})$. Then, generate a λ-bit Bloom filter trapdoor for Q, which is described as follows: (a) Use the Chinese keyword vector generation algorithm based on pinyin string to generate the query keyword vector $\vec{q}_j$ for each

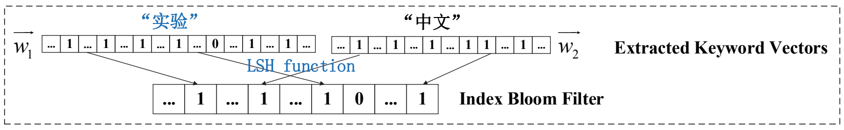
query keyword $q_j \in Q$; (b) Use the LSH functions $h_\sigma \in H$ $(1 \le \sigma \le l)$ to hash the query keyword vectors, which are inserted into Bloom filter T_Q.

(7) **EncTrapdoor(T_Q, SK):** According to the vector S, the trapdoor T_Q is split into T'_Q and T''_Q. If the jth bit of S is 0, the $T'_Q[j]$ and $T''_Q[j]$ are set to be random values such that $T'_Q[j] + T''_Q[j] = T_Q[j]$; if the jth bit of S is 1, set $T'_Q[j] = T''_Q[j] = T_Q[j]$. Then, encrypt T'_Q and T''_Q with key SK and obtain $Enc_{SK}(T_Q) = \{M_1^{-1}T'_Q, M_2^{-1}T''_Q\}$. Finally, the data user uploads the encrypted trapdoor $Enc_{SK}(T_Q)$ to the cloud.

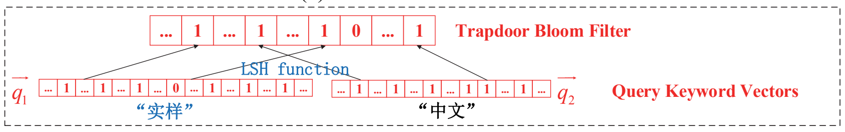
(8) **Query ($Enc_{SK}(\mathcal{I})$, $Enc_{SK}(T_Q)$, k):** According to the encrypted keyword index $Enc_{SK}(I_i)$ and the trapdoor $Enc_{SK}(T_Q)$, cloud server calculates the relevance scores of the encrypted documents, sorts all the relevance scores and returns top-k encrypted documents to the data user. The relevance score of document is calculated as

$$Enc_{SK}(I_i)\cdot Enc_{SK}(T_Q) = \{M_1^T I'_i, M_2^T I''_i\}\cdot\{M_1^{-1}T'_Q, M_2^{-1}T''_Q\} = I'_i\cdot T'_Q + I''_i\cdot T''_Q = I_i\cdot T_Q.$$

(9) Decrypt(C, sk): The authorized data user decrypts the returned top-k ciphertext using the symmetric key sk distributed by the data owner to recover the plaintext document.



(a) Construction of index



(b) Construction of trapdoor

Fig. 5. An example of index and trapdoor construction.

4.3 Analysis of the BCMS Scheme

(1) **Support dynamic document update:** The BCMS scheme uses a single Bloom filter as a keyword index for a document. It is not necessary to change the encrypted keyword indexes of the original dataset to add a new document (or delete an old document). The data owner calculates the term frequency and weighted zone score of the keywords in the new document, inserts encrypted index and ciphertext of the new document (or deletes the old document's encrypted index and ciphertext). Finally, the encrypted new document is uploaded to the cloud (or delete the encrypted old document) to achieve a dynamic document update.

(2) **Save storage and computation overhead:** In traditional keyword searchable encryption scheme based on pinyin string [13, 14], they use the wildcard or gram method to construct the fuzzy keyword sets to achieve fuzzy keyword search, which wastes a lot of storage space and computation resources. Our schemes are compared with these schemes [13, 14]. For the schemes using the wildcard based fuzzy keyword set construction method, assume that the length of the keyword w_i $(1 \le i \le n)$ is L, the sizes of the fuzzy set $S_{w_i,1}$, $S_{w_i,2}$ and $S_{w_i,3}$ are $2*L+1+1$, $C^1_{L+1} + C^1_L \cdot C^1_{L+1} \cdot 2C^2_{L+2}$

and $C_L^1 + C_L^3 + 2C_L^2 + 2C_L^2 \cdot C_L^1$ with editing distance d = 1, 2, 3, respectively. Therefore, the size of the keyword fuzzy set is $O(L^d)$ [14] with edit distance d and length L. The size of the fuzzy set $S'_{w_i,d}$ constructed by the gram method is $C_L^L + C_L^{L-1} + \ldots + C_L^{L-d}$ [13]. Although the storage cost of the gram method is less than that of the wildcard method, it still needs a large amount of storage space. The computation costs of the above two methods come from calculating the hash values for the fuzzy set when constructing encrypted keyword index and the trapdoor. With the increase of the edit distance d and length L, the size of fuzzy keyword set increases rapidly. As a result, the storage and computation overheads grow rapidly too.

Our scheme does not need to construct a fuzzy keyword set. The data owner generates a keyword vector for each keyword and the storage space is not affected by L and d. The storage space of each keyword is $O(1)$. To insert the keyword vector into the Bloom filter, l times LSH function calculations are required, which obviously has better efficiency than the above mentioned schemes.

4.4 Problem of the BCMS Scheme

In Table 2, the Euclidean distance between the original keyword vector and the query keyword vector is the same when the mandarin consonant, vowel, tone changes or a similar pronunciation occurs. In general, there are nine common pronunciation similar pairs in Chinese. Mandarin consonant pronunciation similar pairs are {/sh/and/s/, /ch/and /c/, /zh/and/z/, /b/and/p/, /d/and/t/, /f/and/h/ and /l/and/n/}, and the vowels pronunciation

Table 2. Query examples in the BCMS and ECMS scheme.

Change (consider a syllable)	Query word	Syllable segmentation set	Euclidian distance

original keyword	实验	BCMS: {sh, i, 12, y, an, 24}
		ECMS: {s1, f1/h1, i1, 12, 02, i2, a2, l2/n2, 24}
			0
			0

change of a consonant	实干	BCMS: {sh, i, 12, g, an, 24}
		ECMS: {s1, f1/h1, i1, 12, g2, a2, l2/n2, 24}
			$\sqrt{2}$
			$\sqrt{3}$

change of a vowel	实业	BCMS: {sh, i, 12, y, e, 24}
		ECMS: {s1, f1/h1, i1, 12, 02, i2, e2, 24}
			$\sqrt{2}$
			$\sqrt{3}$

change of a consonant and a vowel	实物	BCMS: {sh, i, 12, w, u, 24}
		ECMS: {s1, f1/h1, i1, 12, w2, u2, 24}
			2
			$\sqrt{6}$

change of a tone	试验	BCMS: {sh, i, 14, y, an, 24}
		ECMS: {s1, f1/h1, i1, 14, 02, i2, a2, l2/n2, 24}
			$\sqrt{2}$
			$\sqrt{2}$

similar pronunciation	实样	BCMS: {sh, i, 12, y, ang, 24}
		ECMS: {s1, f1/h1, i1, 12, 02, i2, a2, l2/n2, g2, 24}
			$\sqrt{2}$
			1

similar pronunciation and change of tone	石羊	BCMS: {sh, i, 12, y, ang, 22}
		ECMS: {s1, f1/h1, i1, 12, 02, i2, a2, l2/n2, g2, 22}
			2
			$\sqrt{2}$

original keyword	故事	BCMS: {g, u, 14, sh, i, 24}
		ECMS: {g1, u1, 14, s2, f2/h2, i2, 24}
			0
			0

change of position	事故	BCMS: {sh, i, 14, g, u, 24}
		ECMS: {s1, f1/h1, i1, 14, g2, u2, 24}
			0
			$\sqrt{10}$

similar pairs are {/an/and/ang/, /en/and/eng/, /in/and/ing/}. Therefore, this issue should be considered when we deal with Chinese keywords. The difference between pronunciation similar pairs should be less than the difference of changes of the mandarin consonant and vowel. BCMS scheme does not consider this issue nor the term weight of keywords such that the search result is not accurate enough.

5. ENHANCED CHINESE MULTI-KEYWORD FUZZY RANK SEARCH SCHEME

In BCMS scheme, the vectors of keywords “故事” and “事故” are the same. If data user queries the keyword “故事”, the encrypted documents containing both “故事” and “事故” are all returned to the data user, which is obviously not reasonable. In addition, we don’t take into consideration the similar consonant pairs like “b/p”, “d/t”. In order to solve these problems, we propose an **e**nhanced **C**hinese **m**ulti-keyword fuzzy rank **s**earch scheme (ECMS), which utilizes Chinese keyword vector generation algorithm based on unigram to convert a pinyin string into a keyword vector. It enlarges the difference of diverse change patterns in the pinyin string to make rank result more accurate. Moreover, the three-factor rank algorithm is designed, which combines the Euclidean distance of keyword vectors, keyword frequency and weighted zone score.

5.1 Chinese Keyword Vector Generation Algorithm based on Unigram

Unigram is a method in natural language processing to divide a word into single symbols. For example, the output of the English word “cloud” processed by the unigram method is the set {c, l, o, u, d}, and that of the Chinese keyword “可搜索加密” is the set {可, 搜, 索, 加, 密}. Before introducing the specific algorithm, we enumerate six special cases that may occur in practical applications and give out a way to deal with these situations: (1) The syllable “ya” is converted into zero-consonant and a vowel “ia”; (2) The syllable “yan” is converted into zero-consonant and a vowel “ian”; (3) The syllable “yang” is converted into zero-consonant and a vowel “iang”; (4) The syllable “yong” is converted into zero-consonant and a vowel “iong”; (5) The syllable “yao” is converted into zero-consonant and a vowel “iao”; (6) The syllable “ye” is converted into zero-consonant and a vowel “ie”. In addition, we define four similar consonant pairs like “b/p”, “d/t”, “f/h” and “l/n”. The unigram based Chinese keyword vector generation algorithm is introduced as follows.

Firstly, convert the Chinese keyword into pinyin string, and split the pinyin string using unigram method. The structure of a keyword vector is shown in Fig. 6. This scheme uses a 108-bit vector to represent a keyword. We utilize the 1st 23 bits to repre-

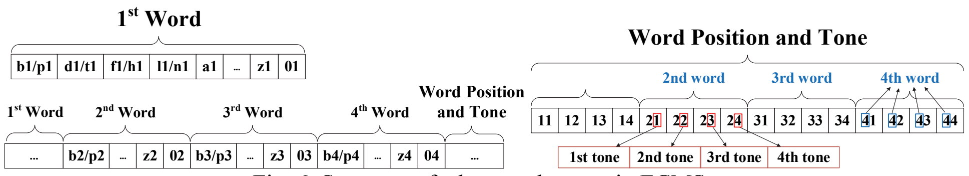
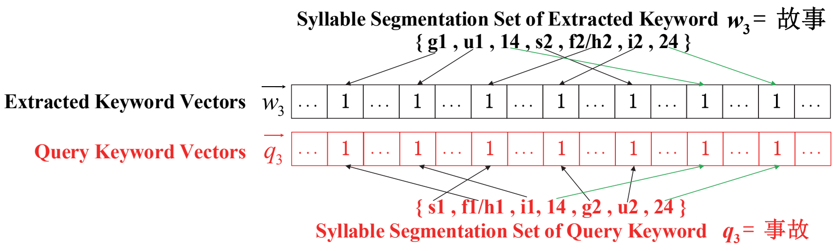


Fig. 6. Structure of a keyword vector in ECMS.

sent the 1st word in the keyword: (1) Put the similar pairs "b1" and "p1", "d1" and "t1", "f1" and "h1", "l1" and "n1" into one bit in the form of "b1/p1", "d1/t1", "f1/h1" and "l1/n1", respectively; (2) The next 18 bits represent the remaining letters from "a1" to "z1" in alphabetical order; (3) The 23rd bit is called "01" bit, where "1" represents the 1st word in the keyword and "0" represents that it (the 1st word) has no consonant. The bit vector structure of the 2nd word, 3rd word and 4th word in a keyword is same as that of the 1st word of the keyword. The last 16 bits (*i.e.*, 93rd-108th bits) represent the word position and tone (the same as that in BCMS). If the element exists in the pinyin string, the corresponding position in the bit vector is set to 1; otherwise, the position is set to 0.

For example, the syllable segmentation set of the keyword "故事" is {g1, u1, 14, s2, f2/h2, i2, 24} using unigram division method, where "g1" indicates that the English symbol "g" comes from the 1st word "故".



Syllable Segmentation Set of Extracted Keyword w_3 = 故事
{ g1 , u1 , 14 , s2 , f2/h2 , i2 , 24 }
Extracted Keyword Vectors $\overrightarrow{w_3}$
… 1 … 1 … 1 … 1 … 1 … 1 … 1 …
Query Keyword Vectors $\overrightarrow{q_3}$
… 1 … 1 … 1 … 1 … 1 … 1 … 1 …
{ s1 , f1/h1 , i1, 14 , g2 , u2 , 24 }
Syllable Segmentation Set of Query Keyword q_3 = 事故

Fig. 7. An example of ECMS.

"故事" and "事故" are completely different, but their syllable segmentation sets are the same in BCMS scheme and denoted as {g, u, 14, sh, i, 24}. As shown in Fig. 7, in ECMS, the syllable segmentation sets of "故事" and "事故" are {g1, u1, 14, s2, f2/h2, i2, 24} and {s1, f1/h1, i1, 14, g2, u2, 24}, respectively. Thus, we succeed to distinguish the disordered keywords.

In ECMS, the syllable segmentation set of "yan" in the keyword "实验" is s1={02, i2, a2, l2/n2, 24}, the syllable segmentation set of "gan" in the keyword "实干" is s2={g2, a2, l2/n2, 24}, the syllable segmentation set of "dian" in the keyword "小店" is s3={d2/t2, i2, a2, l2/n2, 24}, the syllable segmentation set of "pian" in the keyword "十片" is s4={b2/p2, i2, a2, l2/n2, 24} and the syllable segmentation set of "bian" in the keyword "十遍" is s5={b2/p2, i2, a2, l2/n2, 24}. It can be seen that the similarity of "bian" and "pian" is larger than that of "bian" and "dian", and the similarity of "dian" and "tian" is larger than that of "dian" and "bian", too.

5.2 Scheme Construction

ECMS scheme utilizes unigram based Chinese keyword vector generation algorithm to generate the Chinese keywords vector and three-factor rank algorithm to improve the accuracy of the rank results. The three-factor represents Euclidean distance between keyword vectors, weight of keyword information and weighted zone score. Therefore, the difference between ECMS scheme and BCMS scheme lies in the construction of keyword index and trapdoor. In order to simplify the description, this subsection only describes the difference between ECMS scheme and BCMS scheme.

(1) **BuildIndex(F, SK, l)**
• Calculate the keyword frequency weight $wf_{t,f}$. Sublinear scaling method [7] is used to calculate the keyword frequency weight: if $tf_{t,f} > 0$, $wf_{t,f} = 1 + \log tf_{t,f}$; if $tf_{t,f} = 0$, $wf_{t,f} = 0$, where $tf_{t,f}$ represents the term frequency of the keyword t in document f.
• Calculate weighted zone score Z_{ij}. Each document has three domains: title, abstract and text. Set the weight coefficients of these domains as g_1, g_2, g_3, respectively, which satisfy $g_1 > g_2 > g_3$. Let $\upsilon_i = 1$ denote that a keyword appears in the ith domain in the document; otherwise, $\upsilon_i = 0$. Then, calculate the weighted zone score of the keyword w_j. For instance, if a keyword w_j appears in the title and the text (not in abstract) of a document f_i, we set $\upsilon_1 = 1$, $\upsilon_2 = 0$, $\upsilon_3 = 1$. The weighted zone score of the keyword w_j in document f_i is $Z_{ij} = g_1 \times \upsilon_1 + g_2 \times \upsilon_2 + g_1 \times g_3$.
• BuildIndex: Data owner selects l LSH functions from LSH family $H = \{h_\sigma: \{0, 1\}^{63} \rightarrow \{0, 1\}^\lambda\}$. For each document, build a λ-bit length Bloom filter as the keyword index. The keyword index set is denoted as $\mathcal{I} = (I_1, I_2, \ldots, I_m)$. The generation process is described as below: (a) Use Chinese keyword vector generation algorithm based on unigram to generate the keyword vector $\vec{w}_j$ for each keyword w_j in the document f_i; (b) Each keyword vector $\vec{w}_j$ is hashed using the LSH functions $h_\sigma \in H (1 \leq \sigma \leq l)$, which are inserted into Bloom filter I_i. The hash value inserted into Bloom filter is $(Z_{ij} \cdot wf_{t,f})/l$, rather than the value 1 in BCMS scheme.

(2) **Trapdoor:** When an authorized user makes a search query, he first enters a query keyword set $Q = (q_1, q_2, \ldots, q_\eta)$. Then, generate a λ-bit Bloom filter trapdoor for Q and the generation process is described as follows: (a) Use the Chinese keyword vector generation algorithm based on unigram to generate the query keyword vector $\vec{q}_j$ for each query keyword $q_j \in Q$; (b) Use the LSH functions $h_\sigma \in H(1 \leq \sigma \leq l)$, to hash the query keyword vectors and insert them into Bloom filter T_Q.

(3) **Query:** According to the encrypted keyword index $Enc_{SK}(I_i)$ and the encrypted trapdoor $Enc_{SK}(T_Q)$, cloud server calculates the relevance scores of encrypted documents, sorts all the relevance scores and returns top-k encrypted documents to data user. The relevance score of document is calculated as

$$Enc_{SK}(I_i) \cdot Enc_{SK}(T_Q) = \{M_1^T I'_i, M_2^T I''_i\} \cdot \{M_1^{-1} T'_Q, M_2^{-1} T''_Q\} = I'_i \cdot T'_Q + I''_i \cdot T''_Q = I_i \cdot T_Q$$
$$= \sum_{j=1}^{n} (Z_{ij} \cdot wf_{t,f}).$$

5.3 Analysis of the ECMS Scheme

(1) **Support document dynamic update:** The ECMS scheme introduces weight information and the domain weighted scores of keywords to improve the accuracy of rank results. This ECMS scheme uses keyword frequency weight $wf_{t,f}$ to replace the $tf - idf$ weight used in most of the existing schemes [5, 6, 23-26]. The reason is that the $tf - idf$ weight method involves the inverse document frequency idf_t. It indicates the rareness of the keyword t in the whole document set and defined as $idf_t = \log(N/df_t)$, where N is the document number, df_t is the number of documents that contain the keywords t.
Obviously, when the number of the total documents changes, the inverse document frequency idf_t of the keyword t will change. Then, all the original encrypted keyword indexes should be rebuilt such that these existing schemes [5, 6, 23-26] cannot sup-

port dynamic document update. On the contrary, our ECMS scheme substitutes $tf-idf$ with keyword frequency weight $wf_{t,f}$, which is not influenced by the other inserted or deleted documents. Thus, the ECMS scheme supports dynamic document update.

(2) **Improvement of the rank accuracy:** In ECMS, the three-factor rank algorithm is designed to improve the rank accuracy. In BCMS, some of the Euclidian distances are the same shown in Table 2 and cannot distinguish the different change patterns in the pinyin string, such as the mandarin consonant, vowel or tone changes, or a similar pronunciation. In ECMS, the Euclidian distances are different shown in Table 2 such that the different changes in the pinyin string can be distinguished in the rank algorithm due to the characteristic of the LSH function. For example, the Euclidian distance between the vector of similar pronunciation and original word vector is reduced from $\sqrt{2}$ to 1. Then, the difference of Euclidian distance caused by the similar pronunciation is less than that of the variation of mandarin consonant or vowel. For another example, the Euclidian distance between the vector of position change is enlarged from 0 to $\sqrt{10}$. Then, the change of position is distinguished from the original keyword.

6. PERFORMANCE ANALYSIS

In this paper, we use 3000 Chinese research papers as the test dataset. Simulation experiments are implemented using Java language and TextRank algorithm [28], which is part of the HanLP natural language processing package. TextRank algorithm calculates and ranks the weights of keywords in documents, and selects top-100 keywords as the keyword set of the document. Chinese keyword pinyin conversion, mandarin consonant, vowel and tone recognition and word frequency statistics in this paper are also handled by HanLP toolkit. The LSH family with parameter $(\sqrt{3},2,0.56,0.28)$ is used in the simulation. We select $l = 30$ LSH functions and set the length of Bloom filter as $\lambda = 8000$. The experiments are conducted on a desktop computer with CPU i7-2600 3.4 GHz, 16 GB RAM running 64-bit Windows 8 operation system.

6.1 Functions Comparison

In Table 3, the functions of our schemes (BCMS and ECMS) are compared with the schemes in [10, 13, 14, 18, 19], which are anlyzed below; (1) **Multi-keyword** search is not supported in the schemes [13, 18, 19], while the schemes in [10, 14] and our schemes (BCMS and ECMS) have this function; (2) **Weighted zone score** is helpful to return more accurate rank results, which is realized in our ECMS scheme. However, all the other schemes [13, 14, 10, 18, 19] do not take this into consideration; (3) **Fuzzy search** is a frequently used method in the query phase. However, the scheme [10] does not support this search pattern; (4) **Chinese keyword search** is designed according to the characteristic of Chinese, which is useful for the Chinese query. Chen's scheme [18] and our schemes (BCMS and ECMS) realize this function using different methods; (5) **Rank** function enables the system to return the most relevant results to the users, which is considered in the schemes [10, 14, 19] and our schemes (BCMS and ECMS); (6) Yang's scheme [19] and our schemes (BCMS and ECMS) realize fuzzy keyword search **without pre-constructed fuzzy keyword set**, which greatly reduces the storage overhead at the cloud server. Unfortunately, the schemes [10, 13, 14, 18] do not have this desirable characteristic.

Table 3. Query examples in the BCMS and ECMS scheme.

Function	Li [13]	Li [14]	Cao [10]	Yang [19]	Chen [18]	BCMS	ECMS
Multi-keyword	×	✓	✓	×	×	✓	✓
Weighted zone score	×	×	×	×	×	×	✓
Fuzzy search	✓	✓	×	✓	✓	✓	✓
Chinese keyword search	×	×	×	×	✓	✓	✓
Rank	×	✓	✓	✓	×	✓	✓
without pre-constructed fuzzy keyword set	×	×	×	✓	×	✓	✓

6.2 Storage Overheads Comparison

We compare the storage overheads of keyword vectors in our schemes (BCMS and ECMS) and that of the wildcard (or gram) based scheme [13, 14] to demonstrate the efficiency of our scheme. In our system, the lengths of keyword vectors in the BCMS and ECMS schemes are 63 bits and 108 bits, respectively. The keyword vector storage overheads of these two schemes are shown in Fig. 8, which linearly increase with the number of keywords. In order to test the performance of Li's schemes [13, 14] and make a comparison, we construct the fuzzy keyword set of Chinese keyword's pinyin string using Li's wildcard (and gram) based methods. Then, each element in the constructed fuzzy keyword set is mapped into a vector.

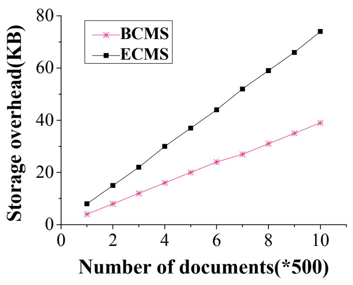


Fig. 8. Storage overhead of BCMS and ECMS.

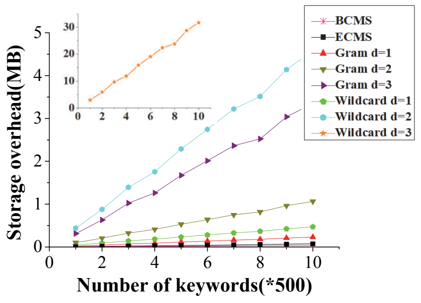


Fig. 9. Storage overhead.

Fig. 9 shows the keyword vector storage overhead comparison between BCMS, ECMS and Li's schemes [13, 14]. The implementations of Li's schemes [13, 14] consists of six colorful lines, which represent the gram based fuzzy keyword set method with edit distance $d = 1, 2, 3$, and wildcard based fuzzy keyword set method with edit distance $d = 1, 2, 3$, respectively. The pink line represents the storage overhead of our BCMS scheme, which has the least keyword vector storage overhead. The black line represents the storage overhead of our ECMS scheme, which has the second least keyword vector storage overhead. ECMS still has far less storage cost compared with Li's schemes [13, 14]. The subfigure in Fig. 9 shows the wildcard based fuzzy keyword set method with edit distance $d = 3$. Since its keyword vector storage space is much bigger than the other methods, we put it in a separate subfigure in Fig. 9 to make the comparison more clearly.

Table 4 gives out the keyword vector storage overhead comparison when the keyword numbers are 1000, 2000, 3000, 4000 and 5000, respectively, which indicates that BCMS scheme has the least storage overhead. It can be seen that the fuzzy set storage overheads of Li's schemes [13, 14] increase rapidly with the edit distance. When edit distance is fixed, the gram based fuzzy keyword set construction method requires less storage space than the wildcard based method. When $d = 3$ and the keyword number is 5000, the storage overhead of gram based method in Li's scheme [13] is about 86 times of that in the BCMS scheme, and the storage overhead of wildcard based method in Li's scheme [14] is about 813 times of that in the BCMS scheme.

Table 4. The storage overhead comparison.

Keyword Numbers / Storage(KB) / Schemes	1000	2000	3000	4000	5000
Gram $d = 1$ [13]	46	91	139	181	232
Gram $d = 2$ [13]	205	410	639	819	1065
Gram $d = 3$ [13]	631	1262	2012	2523	3348
Wildcard $d = 1$ [14]	92	184	282	368	471
Wildcard $d = 2$ [14]	878	1756	2744	3512	4577
Wildcard $d = 3$ [14]	5945	11890	19034	23779	31700
BCMS	8	16	24	31	39
ECMS	15	30	44	59	74

6.3 Computation Overheads Comparison

The trapdoor generation time (encryption time) is the same as the keyword index generation time (encryption time) when the keyword numbers are the same. The keyword index generation time (encryption time) in BCMS and ECMS is same. The reason is that the generation time equals the LSH function execution time and the encryption time is relevant to the length of Bloom filter. Both of the generation and encryption time depends on the length of the keyword vector. Thus, we only plot the keyword index generation time of BCMS scheme in Fig. 10, which increases with the keyword number.

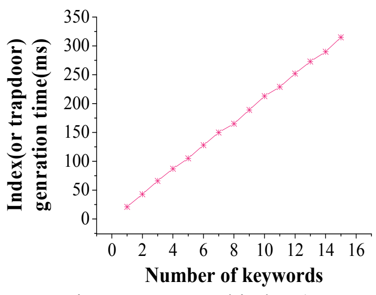


Fig. 10. Keyword index (or trapdoor) generation time.

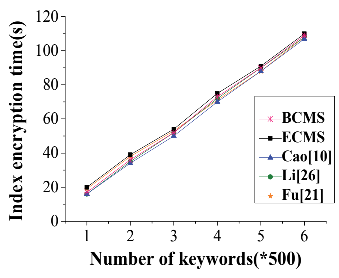


Fig. 11 (a). The index encryption time.

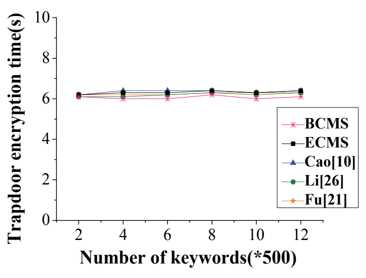


Fig. 11 (b). The trapdoor encryption time.

Fig. 11 (a) shows the keyword index encryption time of Cao's scheme [10], Li's scheme [26], Fu's scheme [21], BCMS and ECMS scheme, which linearly grows with the document number. Fig. 11 (b) plots the trapdoor encryption time of these schemes, which is a constant and does not grow with the number of query keywords. No matter how many keywords are queried, they are all inserted into the Bloom filter. The trapdoor encryption time using the KNN algorithm depends on the length of Bloom filter rather than the number of query keywords.

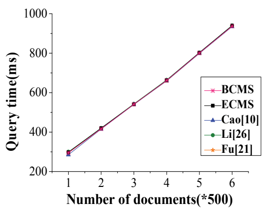


Fig. 12 (a). Query time varies with $|F|$.

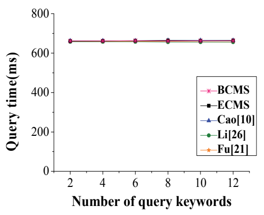


Fig. 12 (b). Query time varies with $|Q|$.

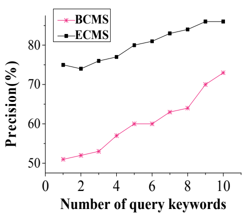


Fig. 13. Precision.

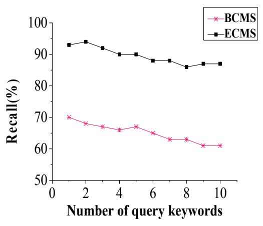


Fig. 14. Recall.

The query time in two schemes of this paper is the same since the query algorithm executed by KNN depends on the length of Bloom filter rather than the length of keyword vector. Fig. 12 (a) shows the query time of these schemes, which linearly grows with document number. Fig. 12 (b) plots the query time of these schemes, which is a constant and does not grow with the number of query keywords.

6.4 Accuracy Evaluation

Before analyzing the accuracy of the proposed scheme, we first introduce two concepts of LSH function and Bloom filter: false positive and false negative rates.

(1) False positive rate. When a query keyword q matches a keyword w and $d(w, q) > r_2$, the false positive situation occurs. Both Bloom filter and LSH function have false positive rate. The false positive rate of the Bloom filter using l hash function with m-bit length is $(1-1/m)^{nl}$, where n indicates that n elements are inserted into the Bloom filter. The false positive rate [20] is calculated by the following formula $(1-(1-p_2)^n(1-1/m)^{n(l-1)})^l$, where p_2 is the parameter of LSH function.
(2) False negative rate. If a query keyword q does not match a keyword w and $d(w, q) < r_1$, the false negative is generated. Bloom filter does not have false negative rate, but LSH function has. The false negative rate of LSH [20] is calculated as $1-(1-(1-p_1)(1-p_2)^{n-1}(1-1/m)^{n(l-1)})^l$.

In order to evaluate the accuracy of the query results, two fundamental notations: precision and recall are used in information retrieval field. Precision is the proportion of the relevant documents to all the returned results. Recall is the proportion of the relevant documents that are returned to all the relevant documents. The relationship among true/false positive and true/false negative are shown in Table 5.

Table 5. The relevance of the returned documents.

	Relevance	Irrelevance
Returned documents	true positive, *tp*	false positive, *fp*
Not returned documents	false negative, *fn*	true negative, *tn*

The precision and recall can be calculated by *Precision* = *tp*/(*tp*+*fp*) and *Recall* = *tp*/(*tp*+*fn*).

(1) **Precision:** Fig. 13 shows the precisions of BCMS and ECMS schemes, which increase with the number of query keywords. According to the precision analysis of the schemes [20, 21], the precision increases with the number of keywords. This is reasonable as the query keywords increase, the impact of false positive caused by the fuzzy keyword will decrease. Similarly, the precisions of two schemes in this paper increase with the number of query keywords. Precision of ECMS scheme is higher than that of BCMS scheme because ECMS scheme utilizes Chinese keyword vector generation algorithm based on unigram to deal with keywords such that different change patterns in the pinyin string can be distinguished, such as the mandarin consonant, vowel, tone changes or a similar pronunciation. And weight information, weighted zone score of keywords in documents and Euclidian Distance of keyword vectors are introduced into the rank algorithm to realize more accurate three-factor rank. In addition, ECMS scheme takes into consideration the word location information in the keyword vector generation algorithm such that the permutation of words in a keyword may result in different syllable segmentation set. Thus, the precision of ECMS scheme increases further. When the number of the query keywords is 10, the precisions of BCMS and ECMS schemes are 73% and 86%, respectively. The precision of ECMS scheme is 13% higher than that of BCMS scheme.

(2) **Recall:** Fig. 12 shows the recalls of BCMS and ECMS schemes, which increase with the number of query keywords. According to the recall analysis of the schemes [20, 21], the recall decreases when the number of keywords increases in the query. It is reasonable as the query keywords increase, the impact of false negative caused by the fuzzy keyword will increase. Similarly, the recalls of two schemes in this paper decrease with the number of query keywords. The recall of ECMS scheme is higher than that of BCMS scheme because ECMS scheme utilizes Chinese keyword vector generation algorithm based on unigram to deal with keywords such that different change patterns in the pinyin string can be distinguished， such as the mandarin consonant, vowel, tone changes or a similar pronunciation. Weight information, weighted zone score of keywords in documents and Euclidian Distance of keyword vectors are introduced into the rank algorithm to realize more accurate three-factor rank and return the most relevant documents. When the number of all relevant documents does not change, ECMS scheme will return more relevant documents. Moreover, ECMS scheme takes into consideration the word location information in the keyword vector generation algorithm such that the permutation of words in a keyword may result in different syllable segmentation set. Thus, the recall of ECMS scheme increases further. When the number of the query keywords is 10, the recalls of BCMS and ECMS schemes are 61% and 87%, respectively. The recall of ECMS scheme is 26% higher than that of BCMS scheme.

7. SECURITY ANALYSIS

Cloud server obtains the access pattern by recording the uploaded trapdoor and the query results of data users. In known ciphertext model, cloud server does not get any additional information except for the access patterns [29].

Theorem 1: BCMS scheme is secure in the known ciphertext model.
This article utilizes some notions in [29].

- *History* is $H = (\Delta, I, W, \vec{W})$, where Δ is a document set, I is an index set built from Δ, $W = (w_1, \ldots, w_k)$ is the query keyword set and $\vec{W} = (\vec{w}_1, \ldots, \vec{w}_k)$ is the query keyword vector set.
- *View* is $V(H) = (Enc_{sk}(\Delta), Enc_{SK}(I), Enc_{SK}(\vec{W}))$, where $Enc_{sk}(\Delta)$ is the encrypted document set (using the symmetric key sk), $Enc_{sk}(I)$ is the encrypted index set (using the key SK) and $Enc_{SK}(\vec{W})$ is the encrypted query keyword vector set (using the key SK). The contents in $V(H)$ is available for the cloud server.
- *Trace of a history* is the sensitive information learnt by the cloud server, such as the access pattern. The *trace of a history* is defined as $Tr(H) = \{Tr(w_1), \ldots, Tr(w_k)\}$, $Tr(w_i) = \{(\delta_j, s_j)_{w_i \subset \delta_j}, 1 \le j \le |\Delta|\}$, where s_j is the relevant score of the query keyword w_i in the file δ_j.

In the known ciphertext model, given two histories $\{H, H'\}$ with same trace, it generates $V(H)$ and $V'(H')$, respectively. If $V(H)$ and $V'(H')$ are not distinguishable, the cloud server (or the attacker) cannot obtain any additional information about the index or the document set except for the access patterns. Now we prove Theorem 1.

Proof: Denote *Sim* as a simulator. Given a history H, the simulator *Sim* can simulate a *View* $V'(H')$ such that the cloud server cannot distinguish $V(H)$ and $V'(H')$. To achieve this purpose, the simulator *Sim* operates as below.

- *Sim* randomly selects $\delta_i' \in \{0, 1\}^{|\delta_i|}$, $\delta_i \in \Delta$, $1 \le i \le |\Delta|$ and ouputs $\Delta' = \{\delta_i', 1 \le i \le |\Delta'|\}$, ranomly generates a vector $S' \in \{0, 1\}^{\lambda}$ and two invertible matrices $M_1, M_2 \in \{0, 1\}^{\lambda\times\lambda}$. The secret key is $SK' = \{M_1', M_2', S'\}$. Then *Sim* constructs $\vec{W}'$ and the trapdoor $Enc_{SK'}(\vec{W}')$ for each $\vec{w}_i \in \vec{W}'$, $1 \le i \le k$; (1) Generate a vector $\vec{w}_i' \in \{0, 1\}^{\lambda}$ such that the number of 1 in $\vec{w}_i'$ is same as that in $\vec{w}_i$. Then, output $\vec{W}' = \{\vec{w}_i', 1 \le i \le k\}$; (2) Generate the trapdoor for each $\vec{w}_i' \in \vec{W}'$ $(1 \le i \le k)$ and output $Enc_{SK'}(\vec{W}') = \{Enc_{SK'}(\vec{w}_1'), Enc_{SK'}(\vec{w}_2'), \ldots, Enc_{SK'}(\vec{w}_k')\}$.
- To generate $I(\Delta')$, *Sim* operates as below: (1) Generate a vector $I_{\delta_j'} \in \{0\}^{\lambda}$ as the index for $\delta_j' \in \Delta'(1 \le j \le |\Delta'|)$; (2) For each $w_i \in W(1 \le i \le k)$, if $w_i \subset \delta_j$, *Sim* sets $I_{\delta_j'} = I_{\delta_j'} + \vec{w}_i'$; (3) For each $I_{\delta_j'}(1 \le j \le |\Delta'|)$, *Sim* replaces the values of the elements that are bigger than 1 with 1 in the vector $I_{\delta_j'}$; (4) Set $I(\Delta') = I_{\delta_j'}, 1 \le j \le |\Delta'|\}$.
- *Sim* encrypts the index $I(\Delta')$ using the secret key SK' and output $Enc_{SK'}(I(\Delta')) = Enc_{SK'}(\{I_{\delta_j'}\}, 1 \le j \le |\Delta'|)$, and then sets $V' = (\Delta', Enc_{SK'}(I(\Delta')), Enc_{SK'}(\vec{W}'))$.

Through the above operations, the encrypted index $Enc_{SK'}(I(\Delta'))$ and the encrypted trapdoor $Enc_{SK'}(\vec{W}')$ generate the same trace as the one that cloud server has generated.

Cloud servers cannot distinguish $V(H)$ and $V'(H')$. Since the documents are encrypted by the symmetric encryption algorithm using the secret key sk, cloud server cannot distinguish $Enc_{sk}(\Delta)$ and Δ'. Moreover, the index $Enc_{SK'}(I(\Delta'))$, $Enc_{SK}(I)$, trapdoor $Enc_{SK'}(\vec{W})$, $Enc_{SK'}(\vec{W}')$ are encrypted by secure KNN algorithm, and the vector split process in the encryption process is random. If cloud server (or the attacker) does not have the secret keys SK and SK', they are not able to recover the plaintext index and query keyword. Thus, cloud server cannot get any additional information about the encrypted index or documents set except for the access patterns.

Theorem 2: ECMS scheme is secure in the known ciphertext model.

***Proof*:** Denote *Sim* as a simulator. Given a history H, the simulator *Sim* can simulate a *View* $V'(H')$ such that cloud server cannot distinguish $V(H)$ and $V'(H')$. To achieve this purpose, *Sim* executes the similar operations as in the proof of Theorem 1, the difference lies in the generation process of $I(\Delta')$. The specific operations are described below; (1) Generate a vector $I_{\delta_j'} \in \{0\}^{\lambda}$ as the index for $\delta_j' \in \Delta'(1 \leq j \leq |\Delta'|)$; (2) For each $w_i \in W(1 \leq i \leq k)$, if $w_i \subset \delta_j$, *Sim* calculates the number of 1 (set the number as μ) in the keyword vector $\vec{w}_i'$ and replaces all 1s in $\vec{w}_i'$ with s_j/μ. Then, set $I_{\delta_j'} = I_{\delta_j'} + \vec{w}_i'$; (3) Set $I(\Delta') = \{I_{\delta_j'}, 1 \leq j \leq |\Delta'|\}$.

According to the analysis in the proof of theorem 1, cloud server cannot distinguish $V(H)$ and $V'(H')$. Thus, cloud server cannot get any additional information about the encrypted index or documents set except for the access patterns.

8. CONCLUSION

A novel Chinese multi-keyword rank searchable encryption scheme is proposed in this paper utilizing locality-sensitive hashing (LSH) function and Bloom filter, which realizes efficient fuzzy keyword storage and supports dynamic document update. Weighted zone score, keyword weight and Euclidean distance are introduced to realize three-factor rank algorithm, which improves the accuracy of the rank results. Theoretical analysis and experimental results show that the proposed system not only realizes fuzzy Chinese multi-keyword rank searchable encryption function, but also reduces the storage cost of the keyword index. Thus, this system protects the privacy of data and realizes more accurate rank.

ACKNOWLEDGEMENT

The authors thank the editor-in-chief, associate editor and reviewers for their constructive and generous feedback. This work is supported by the National Natural Science Foundation of China (No. 61402112, 61702105); Open Fund Project of Fujian Provincial Key Laboratory of Information Processing and Intelligent Control (Minjiang University) (No. MJUKF201734).

REFERENCES

1. P. Mell and T. Grance, "The NIST definition of cloud computing," *Communications*

of the ACM, Vol. 53, 2010, p. 50.

2. B. Chen, Y. W. Chen, K. Y. Chen, *et al.*, "Enhancing query formulation for spoken document retrieval," *Journal of Information Science and Engineering*, Vol. 30, 2014, pp. 553-569.
3. D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceedings of IEEE Symposium on Security and Privacy*, 2000, p. 44.
4. Y. C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Proceedings of International Conference on Applied Cryptography and Network Security*, 2005, pp. 442-455.
5. C. Wang, N. Cao, K. Ren, *et al.*, "Enabling secure and efficient ranked keyword search over outsourced cloud data," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 23, 2012, pp. 1467-1479.
6. C. Wang, N. Cao, J. Li, *et al.*, "Secure ranked keyword search over encrypted cloud data," in *Proceedings of IEEE International Conference on Distributed Computing Systems*, 2010, pp. 253-262.
7. C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*, Cambridge University Press, Cambridge, 2008.
8. D. Liu and S. Wang, "Nonlinear order preserving index for encrypted database query in service cloud environments," *Concurrency and Computation: Practice and Experience*, Vol. 25, 2013, pp. 1967-1984.
9. A. Boldyreva, N. Chenette, Y. Lee, *et al.*, "Order-preserving symmetric encryption," in *Proceedings of the 28th Annual International Conference on Advances in Cryptology*, Vol. 5479, 2009, pp. 224-241.
10. N. Cao, C. Wang, M. Li, *et al.*, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 25, 2014, pp. 829-837.
11. I. H. Witten, A. Moffat, and T. C. Bell, "Managing gigabytes: compressing and indexing documents and images," *IEEE Transactions on Information Theory*, Vol. 41, 1995, pp. 79-80.
12. W. K. Wong, W. L. Cheung, B. Kao, *et al.*, "Secure kNN computation on encrypted databases," in *Proceedings of ACM Sigmod International Conference on Management of Data*, 2009, pp. 139-152.
13. J. Li, Q. Wang, C. Wang, *et al.*, "Enabling efficient fuzzy keyword search over encrypted data in cloud computing," *Computer Science and Information Systems*, Vol. 10, 2009, pp. 1-5.
14. J. Li, Q. Wang, C. Wang, *et al.*, "Fuzzy keyword search over encrypted data in cloud computing," in *Proceedings of IEEE INFOCOM*, 2010, pp. 1-5.
15. C. Wang, K. Ren, S. Yu, *et al.* Achieving usable and privacy-assured similarity search over outsourced cloud data," in *Proceedings of IEEE INFOCOM*, 2012, pp. 451-459.
16. M. Chuah and W. Hu, "Privacy-aware bedtree based solution for fuzzy multi-keyword search over encrypted data," in *Proceedings of the 31st International Conference on Distributed Computing Systems Workshops*, 2011, pp. 273-281.
17. J. Cao, X. J. Wu, Y. Q. Xia, *et al.*, "Pinyin-indexed method for approximate matching in Chinese," *Qinghua Daxue Xuebao / Journal of Tsinghua University*, Vol. 49,

2009, pp. 1328-1332 (in Chinese).
18. H. F. Chen, B. G. Lin, Y. Yang, *et al.*, "Chinese keyword fuzzy search over encrypted cloud data," *NETINFO SECURITY*, Vol. 7, 2014, pp. 69-74 (in Chinese).
19. Y. Yang, S. L. Yang, and M. Ke, "Ranked fuzzy keyword search based on simhash over encrypted cloud data," *Chinese Journal of Computers*, Vol. 2, 2017, pp. 431-444 (in Chinese).
20. B. Wang, S. Yu, W. Lou, *et al.*, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *Proceedings of the IEEE INFOCOM*, 2014, pp. 2112-2120.
21. Z. Fu, X. Wu, C. Guan, *et al.*, "Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement," *IEEE Transactions on Information Forensics and Security*, Vol. 11, 2016, pp. 2706-2716.
22. P. Indyk, R. Motwani, *et al.*, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the 30th ACM Symposium on Theory of Computing*, 1998, pp. 604-613.
23. C. Wang, K. Ren, S. C. Yu, *et al.*, "Achieving usable and privacy-assured similarity search over outsourced cloud data," in *Proceedings of the IEEE INFOCOM*, 2012, pp. 451-459.
24. Z. Xia, Y. Zhu, X. Sun, and L. Chen, "Secure semantic expansion based search over encrypted cloud data supporting similarity ranking," *Journal of Cloud Computing*. Vol. 3, 2014, pp. 1-11.
25. J. Wang, X. Yu, and M. Zhao, "Privacy-preserving ranked multi-keyword fuzzy search on cloud encrypted data supporting range query," *Arabian Journal for Science and Engineering*, Vol. 40, 2015, pp. 2375-2388.
26. H. W. Li, Y. Yang, *et al.*, "Enabling fine-grained multi-keyword search supporting classified sub-dictionaries over encrypted cloud data," *IEEE Transactions on Dependable and Secure Computing*, Vol. 13, 2016, pp. 312-325.
27. B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, Vol. 13, 1970, pp. 422-426.
28. R. Mihalcea and P. Tarau, "TextRank: Bringing order into texts," in *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2004, pp. 404-411.
29. R. Curtmola, J. Garay, S. Kamara, *et al.*, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, 2006, pp. 79-88.
30. C. Hong, Y. Li, M. Zhang, *et al.*, "Fast multi-keywords search over encrypted cloud data," in *Proceedings of International Conference on Web Information Systems Engineering*, 2016, pp. 433-446.

Yang Yang (杨旸) is an Associate Professor in the College of Mathematics and Computer Science of Fuzhou University, China. She received her Ph.D. degree from Xidian University in 2011. Her research interests include information security and privacy protection.

Yu-Chao Zhang (张煜超) is an M.S. candidate in the College of Mathematics and Computer Science of Fuzhou University. His current research interests include information security and searchable encryption.

Jia Liu (刘佳) is an M.S. candidate in the College of Mathematics and Computer Science of Fuzhou University. Her current research interests include information security and searchable encryption.

Xi-Meng Liu (刘西蒙) is a Lecture in the College of Mathematics and Computer Science of Fuzhou University, China. He received his Ph.D. degree from Xidian University in 2015. His research interests include information security and privacy protection.

Feng Yuan (袁峰) is an Assistant Researcher of Beijing Electronic Science and Technology Institute, China. He received his Ph.D. degree from the Xidian University in 2010. His research interests are in the area of information security.

Shang-Ping Zhong (钟尚平) is a Professor in the College of Mathematics and Computer Science of Fuzhou University, China. He received his Ph.D. degree from Chinese Academy of Sciences in 2005. His research interests include machine learning and information security.