A Novel Fuzzy Search Approach over Encrypted Data with Improved Accuracy and Efficiency

Jinkun Cao, Jinhao Zhu, Liwei Lin, Zhengui Xue, Ruhui Ma*, Haibing Guan
Shanghai Jiao Tong University, Shanghai, China
{*caojinkun, zhujinhao, llw02_02 , zhenguixue, ruhuima , hbguan* }*@sjtu.edu.cn*

***Abstract*—As cloud computing becomes prevalent in recent years, more and more enterprises and individuals outsource their data to cloud servers. To avoid privacy leaks, outsourced data usually is encrypted before being sent to cloud servers, which disables traditional search schemes for plain text. To meet both end of security and searchability, search-supported encryption is proposed. However, many previous schemes suffer severe vulnerability when typos and semantic diversity exist in query requests. To overcome such flaw, higher error-tolerance is always expected for search-supported encryption design, sometimes defined as 'fuzzy search'. In this paper, we propose a new scheme of multi-keyword fuzzy search over encrypted and outsourced data. Our approach introduces a new mechanism to map a natural language expression into a word-vector space. Compared with previous approaches, our design shows higher robustness when multiple kinds of typos are involved. Besides, our approach is enhanced with novel data structures to improve search efficiency. These two innovations can work well for both accuracy and efficiency. Moreover, these designs will not hurt the fundamental security. Experiments on a real-world dataset demonstrate the effectiveness of our proposed approach, which outperforms currently popular approaches focusing on similar tasks.**

***Index Terms*—Searchable encryption; Cloud computing; Fuzzy search**

I. Introduction

IN the age of "Big Data" and mobile Internet, the volume of data produced and processed through Internet expands fiercely. It brings a higher demand for storage and computation capacity. Since personal devices are incapable of handling it on many occasions, cloud service is endowed with more importance [1]–[3]. Meanwhile, cloud storage and computing bring new concern on privacy protection [4], [5].

To protect data privacy, outsourced data is usually encrypted in advance by data owners, after which the data query and search are performed. However, conventional search methods cannot be implemented in the ciphertext. Thus, search-supported encryption is proposed [6], with which relevance degree among encrypted text can be measured and thus search over encrypted data becomes possible. Furthermore, considering ambiguity, typos, grammar variance, and semantic variety, bias is common for text matching. Fuzzy search is thus proposed [7]–[9] to achieve more robust search performance with these noises involved (e.g. misspelling detection feature of search engines). Facing the same demand, fuzzy search is also developed on encrypted data. As usual scenes of data search, users care most about the accuracy and efficiency of searching. With a metric, there should be a ground truth rank between stored files and input query by their similarity. Correspondingly, a higher accuracy for search over encrypted data asks that the set of returned files should share a larger overlap with the ground truth most similar files. Efficiency is reflected by the time latency during search and obviously depends on the search algorithm performance. In this paper, we propose a novel multi-keyword fuzzy search design over encrypted cloud data. We promote performance on both accuracy and time efficiency with different innovations.

We conclude the pipeline of searchable encryption approaches into three steps as follows:

1) Represent: Keywords are extracted from outsourced files or received queries, and transfer into word-vectors, a combination of which builds the final representation of files or queries.
2) Encrypt and index: Files and queries are both encrypted to enhance security. They are suggested to be encrypted in heterogeneous ways. The encryption algorithm and key are usually provided by data owners. With some data structure, encrypted files are organized and stored for indexing.
3) Search: In practice, data users send queries and data holders perform some search algorithms on the query and stored encrypted data. Search consists of the calculation of relevance score and ranking by the score. The data user usually only asks for the top-k most relevant files with the query instead of all relevant files.

The first step is critical when designing a fuzzy search mechanism. By representing files in a certain structure, keyword information within files is mapped into a uniform representation space. Similar files, mostly containing a close composition of keywords, are expected to be mapped close in this space. The representation should be error-tolerant for common language bias, such as typos and synonyms, to return users' desired results. On the other hand, the search index, encryption techniques, and search algorithms should cooperate well to conduct fast and correct retrieval.

Even though, promising search accuracy is achieved by current fuzzy search schemes on some occasions, it may fail in many others. Typos, grammatical bias, semantic diversity often bring troubles. Non-perfect representation schemes even create new troubles. For example, anagrams (different words consisting of the same set of letters) will puzzle some

* = Corresponding author

schemes [8] to map different keywords into the same location in the representation space. To overcome similar flaws and promote search accuracy, we propose a novel text presentation scheme that generating keyword-vectors based on designed 'order-preserved uni-gram' (OPU). OPU outperforms popular 'uni-gram' [8] of 'n-gram' [9] in term of accuracy in many cases. The keyword pair 'silent'/'listen' (anagram) or 'keep'/'keap'(typo) has no chance to confuse our proposed mechanism as it can do in some other cases.

Because efficiency is another major concern during data search, accelerating search on without too much harm to accuracy is expected. We also propose an improvement on this aspect by renewing the design of data structure and search algorithm. Precisely, we propose an improved data organization scheme and design search algorithm based on it. A novel data clustering method is designed to gather similar files within a cluster. These clusters are some continuous areas in the aforementioned representation space. Furthermore, an index tree is built in a hierarchical manner by organizing those file clusters. Namely, we design a hierarchical index tree (HIT) for data organization. Compared with the previous designs [10]–[13], this design is expected to achieve better time efficiency with little harm to accuracy. Moreover, it is flexible enough to adapt to different cases with less hand-adjusted parameters. Moreover, such tree-based data organization brings extra convenience to do verification [14], [15] after data retrieval to ensure the freshness, correctness, and completeness of returned data.

At last and most importantly, security and privacy should be guaranteed in our proposed architecture, which is expected to be ensured under different popular threat models [16]. Therefore, focusing on the problem of fuzzy multi-keyword search over encrypted data, we summarize our contributions proposed in this paper in term of two aspects:

- We improve accuracy under many cases by designing a novel file representation scheme named 'order-preserved uni-gram' (OPU). It maps similar text to be close in representation space with kinds of noise involved.
- We improve the search efficiency by designing a new data structure for data organization and corresponding search algorithms. Hierarchical index tree (HIT) is adopted in our scheme to improve time efficiency during query with slight harm to accuracy. To organize outsourced data, an improved dynamic clustering algorithm is proposed which needs less pre-set parameters and thus more flexible.

These two innovations are involved in different stages of search on encrypted data and contribute to the final proposal differently. Of course, our proposal faces the trade-off between time efficiency and search accuracy: OPU brings slightly more computation overhead during the file processing and indexing stage than the word parsing based on naive "uni-gram"; construction of HIT brings slight negative effect on the search accuracy as well. However, both of the degradations are trivial enough that the overall design benefits both accuracy and time efficiency. We design various experiment to compare our proposed approach; with state-of-the-art schemes [8] on real-world linguistic dataset [17] and the result proves the effectiveness of our approach.

II. Related Work

A. Searchable Encryption

Curtmola et al. [18] proposed a security definition on searchable encryption which is followed by most popular mechanisms. Song et al. [19] proposed the first practical searchable encryption mechanism. Previous work focused on improving search accuracy and efficiency without harm to a necessary security guarantee. Wang et al. [20] made an important improvement by introducing novel file indexing techniques. Cao et al. [21] proposed a novel encrypted search scheme supporting multi-keyword matching by coordinate matching. Focusing on reaching similar demand, conjunctive keyword search [22], vector-space-based search models [23] and many other works were proposed. Many aforementioned studies focused on finding a higher efficient file representation design. To the end of encryption, "Secure kNN" algorithm [24] is adopted in most currently popular approaches [8], [20], [24], [25].

B. Quicker search

It is costly to traverse the whole content of all files to calculate their similarity with the query. To tackle the problem of quicker comparison on large-scale file storage, file indexing technique usually is adopted. As the first step, a file is represented by extracted keywords from it, which compresses the indexing volume to a great extent. Then it is common to use some data structures to organize these compressed file indices, on which previous encrypted search schemes propose many innovations. Linear search by a single thread on indices always provides the lower bound of time efficiency because all files are traversed. As an improvement, some studies [26], [27] try to improve latency through parallel computation and multi-task distribution. Some others focus on the improvement of the data structure for file indexing: hashing [13] and tree-like data structure [10]–[12] are widely researched for this topic. Recently, Chen et al. [28] introduced Hierarchical Clustering mechanism into encryption data organization and search algorithm. It is unavoidable in most cases that efficiency improvement from novel data structure brings harm to search accuracy because some files are skipped during the search to save time. To improve time efficiency without too much harm to accuracy is also expected in this area.

C. Fuzzy Search

Language bias such as misspellings and multiple-semantic expression is common and ought to be recognized to improve search accuracy. However, it is hard to distinguish many confusing pairs of words to be typos or different words. For example, how an automated system recognizes that received "catts eat mice" is a typo of "cats eat mice" and search for stored data relevant to the latter and wanted expression? Furthermore, encryption makes it even much more difficult

because a slight difference in original natural language expression could be heavily zoomed out after encryption. When calculating the similarity of different text, these recognized bias declines the search accuracy. To enhance the robustness for search, it comes to the issue of "fuzzy search" topic. Li et al. [7] first formalized and enabled the fuzzy search over encrypted data while maintaining the security guarantee with the help of a pre-defined fuzzy set. Under similar pre-defined dataset, Fu et al. [29] improved search accuracy when synonyms or antonyms exits in text. Without the assistance of extra information, fuzzy search usually is designed based on text representation and indexing schemes with high language-bias-tolerance. Wang et al. [9] relieved the effect of typos or grammatical diversity by transforming keywords into *'bi-grams'*. Furthermore, Fu et al. transform keywords into *'uni-grams'* in [8] to achieve better performance. But word parsing method based on *'uni-grams'* in [8] still have many defects such as not enough robust when anagrams, special character or some other kinds of language bias are involved in text materials.

III. Preliminaries

A. System Model

As popular designs [7]–[9], [21], the system model our scheme faces consists of three components: data owner, cloud server and data user. Data owners encrypt files before outsourcing them and build a data structure to index these outsourced files. The file indices are also encrypted. Outsourced data is only exposed to certified data users and trusted remote servers. Certified data users encrypt their queries and send them to remote servers. Search is operated on remote servers to leverage its computation and storage capacity. Relevance scores between queries and the stored data are computed with some algorithms. Remote servers then return $top-k$ most relevant files to data users. Data users would decipher these files with keys from the data owner.

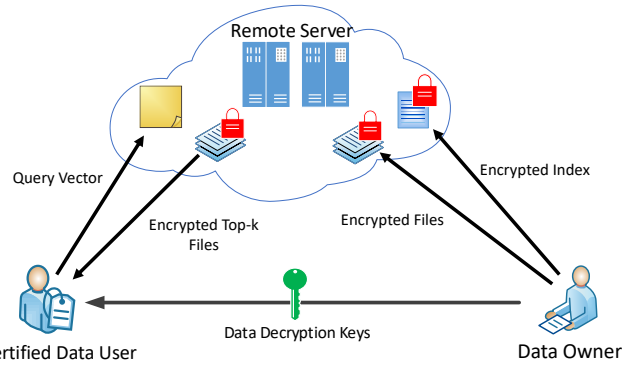

Fig. 1: System Model with three parties

B. Locality-Sensitive Hashing (LSH)

As a subclass of hash functions, locality-sensitive hashing functions have a significant feature. More similar items, which can be determined with some distance metric d (e.g. Euclidean distance), are more likely to be hashed into the same group. A LSH family $\mathcal{H}=\{h_1,h_2,...,h_l\}$ is defined as $(r_1,r_2,p_1,p_2)-sensitive$ if for each $h_q\in\mathcal{H}(1\leq q\leq l)$, two arbitrarily chosen points u,v satisfy:

$$if\ d(u,v)\leq r_1 : Pr[h_q(u)=h_q(v)]\geq p_1 \tag{1a}$$
$$if\ d(u,v)\geq r_2 : Pr[h_q(u)=h_q(v)]\leq p_2 \tag{1b}$$

where $d(u,v)$ is the distance between these two points to represent their similarity. $p-stable$ LSH [30] is a specific kind of LSH methods based on a $p-stable$ function family, which can be formulated as:

$$h_{a,b}(\overrightarrow{v})=\lfloor\frac{\overrightarrow{R}\cdot\overrightarrow{v}+b}{a}\rfloor \tag{2}$$

where $\overrightarrow{R}$, $\overrightarrow{v}$ are two vectors and a and $b\in[0,a]$ are two real numbers. $\overrightarrow{R}$, a and b are parameters and $\overrightarrow{v}$ is the variable to be hashed.

C. Bloom Filter

Bloom Filter [31] is a special data structure widely adopted to map a high-dimensional point into a space of lower dimensions. A D-dimensional point is transformed into a one-hot m-bit vector with a hash function. With a set of l independent hash functions $\mathcal{H}=\{h_1,h_2,...,h_l\}$, the point can be thus transformed into a m-bit vector with at most l nonzero bits. With the features of LSH, more similar keywords are expected to be mapped into the same position with higher probability by the same LSH function. Thus the finally generalized vectors are more likely to be similar or even the same. For a given set of points $\mathcal{P}=\{p_1,p_2,...,p_n\}$, the l independent LSH functions encode each $p_i(1\leq i\leq n)$ into at most different l bits of a m-bit vector $BF_{\mathcal{P}}$. This vector is thus called 'Bloom Filter'. To judge whether a point $p\in\mathcal{P}$, we simply generalize its corresponding bloom filter the same set of hash functions and test whether there is the same bloom filter for p_i found. As proved in [31], the chance to give a false positive through this method is approximately $(1-e^{-\frac{l\times n}{m}})^l$. The minimal rate of false positive is $(\frac{1}{2})^l$, which is achieved when $n\times l=m\times\ln 2$. Thus a better expected false positive is available with bigger l. But to set l small can keep produced Bloom Filter sparse, which is helpful to increase the accuracy of our scheme. This raises an important trade-off for application of bloom filters.

D. Hierarchical Clustering Tree

Clustering algorithms are adopted to divide items into different clusters through comparing their similarity, namely items are divided into the same cluster if they are adjacent enough in the vector space. Popular hierarchical clustering techniques, such as k-means [32], DBSCAN [33] and GMM [34], are widely adopted in data mining. Hierarchical clustering in [28] sets a maximum number of elements in a cluster and then begins with random sample points to divide adjacent items into one cluster. For data organization in search cases, the element to be clustered can be a file or a query or the center point of a sub-cluster, all of which are first mapped to a

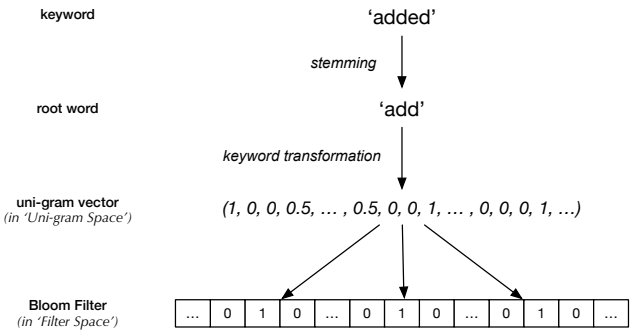

Fig. 2: Transformation from keyword to Bloom Filter

uniform space and represented by a point in the space. In hierarchical clustering, the clustering algorithm is performed recursively on original elements and the points representing sub-clusters. Finally, we could generalize a hierarchical clustering tree. Specifically, each outsourced file is a leaf node of the hierarchical clustering tree built through. To search on the tree to find some nodes, the general time complexity is simply $\mathcal{O}(log(n))$ instead of $\mathcal{O}(n)$ for linear search.

IV. Proposed Algorithms

In this section, we introduce our proposed algorithms in detail. Our main innovations include a novel keyword transformation scheme and a specially designed data structure for indexing. The former is named 'order-preserved uni-gram'(OPU) to show its difference with traditional 'uni-gram' [8] or 'n-gram' [20]. The designed data structure is called "Hierarchical Index Tree" (HIT). With the help of OPU, more information of the original natural language keywords can be reserved without privacy leaks, providing help for more accurate search. On the other hand, we propose an adaptive clustering algorithm to build HIT and a corresponding search algorithm to better balance the trade-off between search efficiency and accuracy.

A. "Order-Preserved Uni-gram" (OPU)

Misspelling occurs in three cases: *'misspelling of letters', 'order of letters reversed' and 'addition or missing of letters'*. To judge the similarity of two keywords, the popular mechanism usually splits keywords into some "pieces". The granularity of which keywords are decomposed influence the information it can memorize and the collision chance that different keywords are transformed into similar or even the same set of pieces. Wang et al. [9] adopted the *'n-gram'* method, which would split the word *'task'* into the set $\{ta, as, sk\}$ when $n = 2$. This method achieves a good result for the first case but fails for another two cases. Based on this scheme, Fu et al. [8] proposed *'uni-gram'* method, under which, for instance, the uni-gram set of keyword *'scheme'* is $\{s_1, c_1, h_1, e_1, m_1, e_2\}$. This scheme performs better for the other two cases. However, due to the lack of information on the order of letters, such a scheme is incapable of recognizing *'anagram'*, such as 'devil' and 'lived'.

To bring better fuzzy search, we propose a new method to transform a keyword into an 'order-preserved uni-gram' vector, which would show advantages compared with previous work. While keywords are of different length, the output OPU vectors are expected to be equally long. The construction of an OPU vector can be divided into three steps, decompose, encode and infect, introduced as follows:

1) Decompose: Given a set of keywords extracted from some text materials, we perform this operation on each of them respectively. First, all keywords should be stemmed [35] to eliminate the grammatical and other linguistic variations. Then all keywords would be dismembered into single letters. But different from the unordered set of letters for 'uni-gram', we also record the position of each letter in the original keyword for further use.

2) Encode: To relieve computation and storage overheads, at most first u letters in a keyword can be fully represented after transformation. The corresponding length of output OPU vector, V_{KW}, should be $(u \times 26 + 30)$, all bits of which are binary. Specifically, the vector consists of u "letter blocks"(LB), each of which has 26 bits, and one 'digit and symbol block'(DSB), whose length is 30 bits. Letters between 'a' to 'z' are mapped to $1^{st} - 26^{th}$ bit in each letter block. The i^{th} letter in KW corresponds to i^{th} letter block in V_{KW}.

For example, if the target keyword is "add", only $V_{KW}[1]$, $V_{KW}[26{+}4]$ and $V_{KW}[26{\times}2{+}4]$ are set to 1 with all other bits remaining 0 (because "a" and "d" are respectively the first and the fourth letter in alphabet). The last 30 bits in V_{KW} indicate whether 10 digits ('0'-'9') and 20 widely used symbols appear in KW. In practice, if a keyword is too long to fit the preset length of an OPU vector, it will be cast. As a trade-off, a longer vector can represent more complicated keywords but brings more overhead. On the other hand, a too short vector would make information loss normal, which brings severe hurt to representation rationality.

After encoding, we have vectors of uniform length representing keywords. The position of each letter in the original keyword is encoded in these vectors, making a critical difference with traditional methods.

3) Infect: A simple insight to realize fuzzy search is to raise the tolerance for letter dislocation when transforming words to a standardized representation (such as the uni-gram vector adopted in our scheme). To the tolerance of the produced uni-gram vector, we propose an 'Infect' mechanism, which makes the most difference between our proposed method and previous methods. Each nonzero bit of the vector after "Encode" would share its weights with neighboring bits and the relation is determined by an *Infection Function*. After *Infection*, bits of original representation vector may be transformed from binary to float. A typical kind of *Infection Functions* can be formulated as:

$$\delta w(d) = \begin{cases} \frac{1}{s^{d/26}} & d \leq 26u, d\%26 = 0 \\ 0 & otherwise \end{cases} \tag{3}$$

where d is the bit distance between two bits, s is a factor to adjust the infection strength. Note u is another to adjust how far the infection can spread and the farthermost distance

is $d_{max} = 26u$. Because same letters on the neighboring position of the original keyword are mapped into two 26-bit-far-away bits, only when $d\%26 = 0$, infection happens. Such a mechanism would weaken the negative effect brought by letter dislocation.

To explain that in detail, we study it with an example. A keyword "add" is already encoded into a OPU vector after "Decompose" and "Encode". If we set $s = 2$ and $u = 2$, *Infection* happens: $V_{KW}[1], V_{KW}[30]$ and $V_{KW}[56]$ are 1, others being 0. In the first wave of infection, value of $V_{KW}[27], V_{KW}[4], V_{KW}[56], V_{KW}[30]$ and $V_{KW}[82]$ increases by $\delta w = \frac{1}{2^1} = 0.5$. Then the second wave of infection follows: $V_{KW}[53], V_{KW}[82], V_{KW}[4]$ and $V_{KW}[108]$ are further increased by $\delta w = \frac{1}{2^2} = 0.25$. So far, infection finishes because $s = 2$. Finally the OPU vector generalized from the keyword "add" becomes V'_{KW} with $u = 2$:

$$V'_{KW}[i] = \begin{cases} 1.5 & i = 30, 56 \\ 1 & i = 1 \\ 0.75 & i = 4, 82 \\ 0.5 & i = 27 \\ 0.25 & i = 53, 108 \\ 0 & otherwise \end{cases} \tag{4}$$

4) Analysis of Order-preserved Unigram (OPU): We analyze the improvement of the proposed OPU over previous 'bi-gram' [9] and 'uni-gram' [8] design in this part. Comparison is performed through examples with different requirements for fuzzy search taken into consideration. The similarity of two keywords is simply quantified in Euclidean distance as:

$$Dist[V_1][V_2] = \sqrt{\sum_{i=1}^{N} (V_1[i] - V_2[i])^2}, \tag{5}$$

where V_1 and V_2 are respectively the representation vector of two keywords. N is the length of vectors. In fuzzy search, we want simple typo brings no severe decrease to similarity score. In other words, if V_1 is the correct form of a keyword and V_2 is a corresponding fuzzy form, their distance should be as short as possible so that they are considered to be largely similar during a search. On the other hand, if V_1 and V_2 are different wanted keywords and they are similar under some metric, a robust representation design can still distinguish them after vector representation. For instance, "add" and "dad" are not expected to be thought "similar" or even "the same" under a good scheme, which may bring huge bias into search results.

Considering various requirements for fuzzy search, three types of misspellings cases should be taken into consideration:

a) Letter misspelling: Letter misspelling indicates when letters in a word are replaced by some incorrect ones. For example, "beer" can be misspelled as "berr".

b) Wrong letter order: Wrong order of letters indicates when words are consist of a uniform set of letters, but some letters in them are arranged with the wrong order. For example, the word "bere" may be typed as "beer" for wrong letter order.

Fuzzy type	Correct	Wrong	Bi-gram	Uni-gram	OPU
Misspelling	add	aad	2	2	$\sqrt{3.125}$
	bear	beer	2	2	$\sqrt{3.25}$
Wrong Order	used	uesd	$\sqrt{6}$	0	$\sqrt{1.75}$
	pear	paer	4	0	1
Insertion/Absence	pen	pn	$\sqrt{3}$	1	$\sqrt{3.25}$
	pen	pean	$\sqrt{3}$	1	$\sqrt{2.75}$

TABLE I: The comparison of similarity scores under different schemes. The similarity is represented by Euclidean distance and parameters for OPU are set as: $u = 2$ and $s = 2$.

c) Insertion/Absence of letter: Insertion or absence of letter(s) in a word occurs frequently as well, causing typos in the text. For example, the word "pen" may be misspelled to be "pean" or "pn" in this case.

An ideal keyword decomposing approach should recognize the high similarity between a word and its typos. On the other hand, as an obvious trade-off, when a typo suffers from the severe difference from the correct form, the similarity should no longer be high, or the approach would become invalid to distinguish some different but similar words. For the three listed fuzzy cases, we calculate the relevance scores under different word decomposing approaches. The results are shown with examples in TABLE I.

Except for the three basic types of misspellings, OPU handles many other cases better than 'uni-gram' and 'bi-gram'. The most obvious shortcoming of traditional 'uni-gram' is that only the composition of letters in a word is recorded after decomposing, the information of their position is lost totally. Such a shortcoming makes it unable to distinguish different words in many cases. For example, for the traditional 'uni-gram' mechanism, anagrams like "listen" and "silent" produce the same keyword vector after transformation, which is incapable of satisfying users' demand. For example, it is apparently unacceptable to regard "Dad is silent" and "Dad is listen" as the same queries. While in our proposed OPU, the position information is also encoded into the final word representation, thus anagrams could no longer compromise our approach.

In summary, based on 'uni-gram', our proposed OPU not only inherits all its advantage but also encodes the position of letters in the original word into final representation, which enhances our scheme in many cases. We qualitatively compare the ability of three mentioned mechanisms to show the difference between different words in different cases and the result is shown in TABLE II. Note that "Ex-1" concludes the case with non-alphabet involved in a string.

B. "Hierarchical Index Tree" (HIT)

As the other main contribution, we exploit an efficient data structure to organize the file indices. Instead of the most naive linear organization, we design a tree-based index organization. To be precise, we build a hierarchical index tree to organize the file indices for faster file search. To relieve computation

Keyword #1	Keyword #2	bi-gram	uni-gram	OPU
listen	silent	Yes	No	Yes
2case	case2	No	No	Yes
exams	exam	Yes	Yes	Yes
running	run	No	Yes	Yes
useful	use	No	Yes	Yes
Ex-1	Ex-1	No	Yes	Yes

TABLE II: Comparison among three schemes

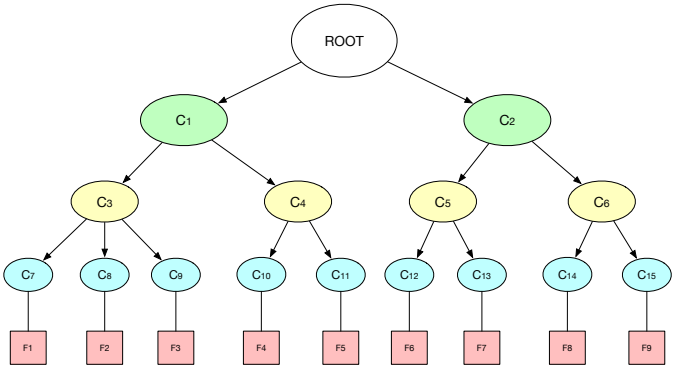

Fig. 3: Overview of Hierarchical Index Tree

overhead, we do not index a file straightforward through its full keyword vector, which usually is of high dimension. Instead, we map original word vectors into some intermediate representation of lower dimension and then perform indexing on it. As mentioned before, we choose "bloom filter" as the intermediate representation.

1) Construction of HIT: Given the file representation vectors, each of which encodes the keyword information of a file, we propose HIT to organize them. To build HIT, we need to divide bloom filters into clusters. Some nearby clusters may form a larger cluster standing on a higher layer in HIT. We propose an improved dynamic K-means Algorithm to plan these clusters of different levels, through linking which we could build the final HIT. We explain the algorithm in Algorithm 1. Instead of giving a fixed tightness factor to determine the final cluster number, we compare the average point distance and minimum point distance in a cluster to determine whether this cluster should be further subdivided.

To generalize clusters of higher level, the clusters of lower-level are represented by their center coordinates and thus regarded as "points". Algorithm 1 is therefore capable of clustering some tiny clusters into a larger one. Such operation is kept performing until a final cluster containing all points is generalized, and the construction of HIT is done.

2) Search in HIT: To search files in HIT, we need to calculate the relevance score between the representation vectors in HIT nodes and a given query vector, which usually is generalized from a query string. Only the leaf nodes in HIT represent real files and search process is expected to return k most similar stored files. To increase time efficiency, an intelligent search algorithm should not calculate the relevance score of every leaf between the target vector. Therefore,

Algorithm 1 HIT Construction

Input:
1. $\mathcal{P} = \{P_1, P_2, .., P_m\}$: the set of points to be clustered
2. e: a factor to adjust the desired tightness of clusters
3. D: function to calculate the distance of two points in $\mathcal{P}$

Output:
1. $\mathcal{C} = \{C_1, C_2, ..., C_n\}$: coordinates of built cluster centers
2. $\mathcal{L} = \{L_1, L_2, ..., L_m\}$: cluster index of each point

$n \leftarrow 1$
$Stable \leftarrow False$
while not Stable **do**
  Divide $\mathcal{P}$ into n clusters through K-means method
  Update $\mathcal{C}$ and $\mathcal{L}$
  $Stable \leftarrow True$
  for C_i in $\mathcal{C}$ **do**
    Calculate the distance matrix: $\mathcal{M}^i_{uv} = D(P_u, P_v)$, where $P_u, P_v \in C_i$
    $R_{avg} \leftarrow$ the average distance in M^i
    $R_{min} \leftarrow$ the minimum distance in M^i
    if $R_{min} < eR_{avg}$ **then**
      $n \leftarrow n + 1$
      $Stable \leftarrow False$
return $\mathcal{C}$ and $\mathcal{L}$

how to find reliable top-k most similar files without too much computation is the core problem. We thus design a search algorithm adapted to the HIT structure as explained in Algorithm 2, which could be notated as $Search(R, T, k, Rs)$.

Only to traverse all files can ensure that the literally "top-k most relevant files" are always found. Linear traversing requires a time complexity of $\mathcal{O}(n)$, where n is the scale of stored files, which is unacceptable in most cases. Through the proposed search algorithm, we try to find a proper trade-off between search accuracy and time efficiency. In other words, we look up required k files with $\mathcal{O}(log_n)$ time complexity and returned k files can be expected to be included in the global "top-k most relevant files". The designed experiment proves that HIT improves time efficiency to a high extent without bringing much damage on search accuracy.

V. Architecture Construction

In this section, we explain the complete pipelines of our proposed architecture in detail. Note that the scheme comprehensively leverages multiple algorithms and data structures in different steps dependent on each other. However the general architecture is still flexible with sub-modules replacement. For example, replacing bloom filter with other data structure or adopting a different encryption approach would not disable the architecture but only change fine-grained operations. The overview of the proposed architecture is visualized in Figure 4 in detail.

A. Keyword Extraction and Preprocessing

Given the set of files to be outsourced as $\mathcal{F} = \{F_1, F_2, ..., F_n\}$, the first step is to extract the keywords from them. Preposition, pronoun, auxiliary words and other

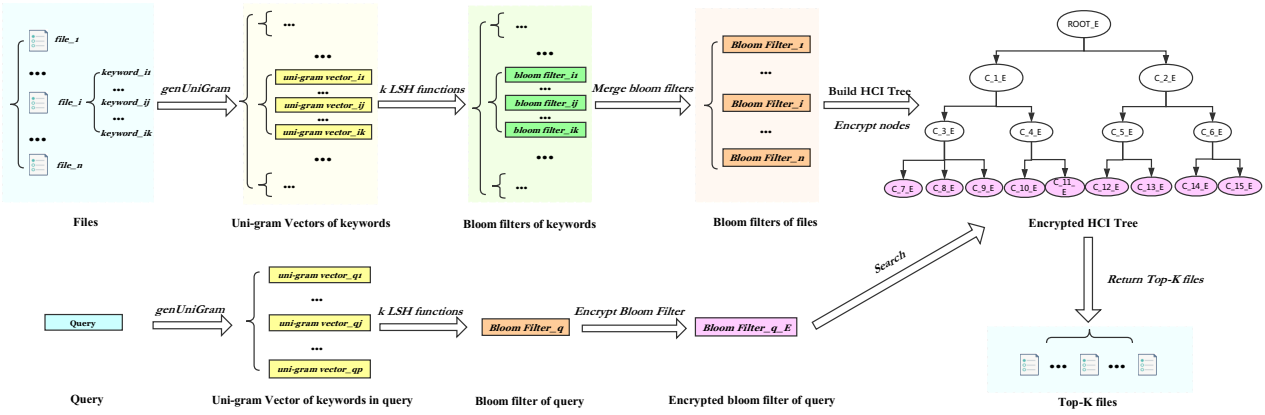

Fig. 4: Overview of the complete procedure of proposed scheme

Algorithm 2 Search in HIT

Input:

1. R: root node of current HIT
2. T: the representation vector of input query
3. k: the number of files to be returned
4. $Rs(A, B)$: function of relevance score between A and B

Output:

$\mathcal{F} = \{F_1, ..., F_k\}$: found top-k similar files

while children of R are not leaves **do**
  children nodes of R are $C(R) = \{C_1, ..., C_p\}$
  for $i \leftarrow 1$ to p **do**
    $S_i \leftarrow Rs(C_i, T)$
  find $S_k \leftarrow max\{S_i | 1 \leq i \leq p\}$
  $R \leftarrow C_k$
children of R are $C(R) = \{C_1, ..., C_q\}$
$\mathcal{S} = \{S_1, ..., S_p\}$ where $S_i = Rs(C_i, T)$
$R' \leftarrow R$
if $k \leq q$ **then**
  $\mathcal{C} = \{C'_1, ..., C'_k\} \leftarrow$ points ranking top-k in $\mathcal{S}$
  $\mathcal{F} = \{F_1, ..., F_k\}$: files of $C'_i (1 \leq i \leq k)$
else
  while $|C(R')| = 1$ or $|L(R')| < k$ **do**
    % $L(R')$ returns number of leaves under R'
    $R' \leftarrow$ father of R'
  $\mathcal{F}_1 \leftarrow$ files corresponding to points in C_R
  $\mathcal{F}_2 \leftarrow Search(R', T, k - p, Rs)$
  $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$
return $\mathcal{F}$

keywords without concrete semantic meaning are filtered out. Remaining keywords are noted as $\mathcal{K} = \{\mathcal{K}_1, ..., \mathcal{K}_n\}$, where $\mathcal{K}_i$ is the set of keywords extracted from F_i. We have $\cup_{i=1}^{n} \mathcal{K}_i = \{KW_1, ..., KW_N\}$. Then, we apply Stemming Algorithm [36] on these keywords to eliminate grammatical diversity. For example, "listening, "listen and "listened would be all stemmed into a uniform keyword 'listen'. After stemming, we then calculate the "term frequency" and "inverse

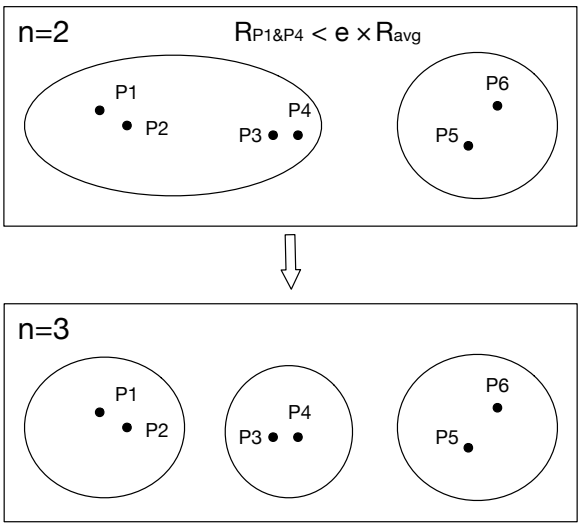

Fig. 5: Adjust cluster number in Algorithm 1

document frequency"(TF-IDF) [37] value of each keyword for further use.

Considering there are in total N legal keywords, the TF value of the i^{th} keyword KW_i, in the j^{th} file F_j, is formulated as:

$$TF_{i,j} = \frac{n_{i,j}}{\sum_{k=1}^{N} n_{k,j}}, \tag{6}$$

where $n_{i,j}$ is the frequency of KW_i in F_j. The IDF value is on the other calculated as:

$$IDF_i = \frac{|\mathcal{F}|}{|\{F_k | KW_i \in F_k\}|} \tag{7}$$

The TF-IDF value of KW_i in the file F_i is thus simply calculated by:

$$TFIDF_{i,j} = TF_{i,j} \times IDF_i \tag{8}$$

B. Generalization of representation vectors of files

To relieve computation overheads, we would transform the original keywords in natural language form into some numerical representation. For computation and storage convenience, they are in general transformed into vectors of uniform length. Then we build a structured representation of each file based on the representation vectors of all contained keywords in it. In our proposed scheme, we follow two steps to reach the goal.

1) Keyword representation: To represent a keyword in a uniform numerical form, we transform each keyword into a corresponding OPU vector as introduced in the aforementioned sections. Then, with l LSH functions, $\mathcal{H} = \{H_1, H_2, ..., H_l\}$, a keyword would be encoded into at most l bits in a bloom filter with else bits all zero.

2) File representation: All keywords in a file haven been represented already with bloom filter and the file can thus be represented a set of keyword bloom filters. For example, for a file F_i, we have $BF_{F_j} = \{bf_{1,j}, ..., bf_{u,j}\}$ where $bf_{k,j}$ is the bloom filter of the k^{th} keyword in F_j. Furthermore, we can represent the file in a bloom filter as well. The bloom filter of a file is of the same length with that of keywords. Through bit-wise addition and pre-calculated TF-IDF value of each keyword, the final representation of F_j is formulated as:

$$bf_{F_j} = \sum_{k=1}^{u} TFIDF_{k,j} \times bf_{k,j} \tag{9}$$

Till now, we've got the final representation of files to be outsourced. For queries, which would be involved in the architecture in the search stage, we transform them into a numerical representation as same as the process for files.

C. Construction of encrypted HIT

For search efficiency, we would outsource an aforementioned HIT for file indexing to the remote servers. Besides, for privacy protection, we have to encrypt built HIT before the outsourcing. In this section, we introduce the process to build and encrypt the HIT.

1) Building HIT: Before this step, each file has been represented in a bloom filter, which can also be regarded as a point in a $l_b - dimensional$ space, where l_b is the length of adopted bloom filters. As introduced in the aforementioned section and illustrated in Algorithm 1, we can build an HIT to organize the indices of files now.

2) Encryption of HIT and query: To ensure security and privacy, simply to transform data into bloom filters may be inadequate because such a transformation is determinative. Hence, we adopt secure kNN [19] algorithm to encrypt all nodes in the HIT as well. On the other hand, a query will also be encrypted in a corresponding manner to enable the calculation of relevance between files and query. This process is decomposed into steps as follows:

a) KeyGen(m): Given a security parameter m, this method generates a security key SK, which is a tuple (M_1, M_2, S). M_1 and M_2 are both m-dimensional invertible matrices and $S \in (0,1)^m$ is an m-dimensional vector.

b) EncIndex(I, SK): With secure kNN to encrypt an index vector I, which should be a bloom filter in our scheme and of length m, it should be first split into two vectors (I', I'') as follows:

$$I'[j] = \begin{cases} I[j], S[j] = 1 \\ \frac{1}{2} \times I[j] + r, S[j] = 0 \end{cases}, 1 \le j \le m \tag{10}$$

$$I''[j] = \begin{cases} I[j], S[j] = 1 \\ \frac{1}{2} \times I[j] - r, S[j] = 0 \end{cases}, 1 \le j \le m \tag{11}$$

where r is a small randomness introduced for security consideration. Finally, the encrypted expression of I is $T_F = \{M_1^T \cdot I', M_2^T \cdot I''\}$.

c) EncQuery(Q, SK): For queries in the search stage, it should also be encrypted to prevent information leak. Following the previous steps, a query has also been represented in an m-dimensional bloom filter, which is notated as Q. To encrypt it into a trapdoor, symmetric operations are operated to split it into two vectors (Q', Q'') as well:

$$Q'[j] = \begin{cases} Q[j], S[j] = 0 \\ \frac{1}{2} \times Q[j] + r, S[j] = 1 \end{cases}, 1 \le j \le m \tag{12}$$

$$Q'' = \begin{cases} Q[j], S[j] = 0 \\ \frac{1}{2} \times Q[j] - r, S[j] = 1 \end{cases}, 1 \le j \le m \tag{13}$$

Finally the trapdoor of the query vector is expressed as: $T_Q = \{M_1^{-1} \cdot Q', M_2^{-1} \cdot Q''\}$

D. Search

While once all data encrypted, we need to re-introduce the calculation of relevance score. Fortunately, secure kNN has a great feature that it allows invariant relevance score through naive inner production. Given an encrypted file index T_F and an encrypted query T_Q, we can obtain their relevance score as follows:

$$\begin{aligned} & Rs(T_F, T_Q) \\ &= T_F \cdot T_Q \\ &= \{M_1^T \cdot I', M_2^T \cdot I''\} \cdot \{M_1^{-1} \cdot Q', M_2^{-1} \cdot Q''\} \\ &= I' \cdot Q' + I'' \cdot Q'' \\ &= \sum_{j=1}^{m} (I'[j] \cdot Q'[j] + I''[j] \cdot Q''[j]) \\ &= \sum_{j=1, S[j]=0}^{m} (\frac{1}{2} I[j] + r) \cdot Q[j] + (\frac{1}{2} I[j] - r) \cdot Q[j] \\ &+ \sum_{j=1, S[j]=1}^{m} (\frac{1}{2} Q[j] + r) \cdot I[j] + (\frac{1}{2} Q[j] - r) \cdot I[j] \\ &= I \cdot Q \end{aligned} \tag{14}$$

Eq 14 proves that inner production on two vectors can produce the same result before and after the encryption by secure

kNN. So far, we can search for the top-k most relevant files through HIT referring to a given query. The required algorithm has been introduced in previous sections and concluded in Algorithm 2.

E. Verification

Similar as explained in [28], thanks to the construction of HIT, returned files can be further verified to confirm the correctness, completeness, and freshness. To support this mechanism, the data owner needs to build a signed tree in advance and outsource it to the cloud server together with the index tree. Once the search finishes, the cloud server returns top-k files together with the *signed sub-tree* along the search path to the data user. For example, referring to Fig 3, if files in C_7, C_8, C_9, C_{10} are returned as search result, the representation vectors stored in $C_1, C_3, C_4, C_7, C_8, C_9, C_{10}$ would be all returned as well because these nodes are gone through in search path. Here, we could adopt a signature algorithm such as RSA [38] to generate this signed hash tree. Then, with the received *signed sub-tree*, the data user calculates the signature of each node with its corresponding representation vector and compares the calculated signature with the returned signature. The returned files pass the test, and as claimed in [28], their correctness, completeness, and freshness could be verified without considering the real-time data update on remote servers.

VI. Privacy Analysis

In this section, we state the background settings and analysis of the security promise of our proposed architecture theoretically. The analysis is performed under two threat models, where privacy leakage resulting from untrusted data use is out of the discussion. The authorization of data users and remote servers is also not in the scope of this section. We assume all data users taken into consideration are already certified by the data owner.

A. Threat Model

The cloud server is considered as honest-but-curious [39], which means that it receives queries and executes the search as commanded, but it tries to derive sensitive information from queries and stored encrypted data at the same time. The requirement of privacy in our scheme is as same as that defined in [8], [9], [18]. Thus, there are two threat models asking for different levels of privacy protection:

a) Known Ciphertext Model: In this threat model, the cloud server is expected to gain only the content of encrypted files, encrypted file index and encrypted trapdoor of received queries.

b) Known Background Model: In this threat model, the cloud server is also interested in the statistical background information of encrypted data. It could perform a statistical attack to obtain more information about keywords [21], [40] involved in the search. Attach under this threat model also reveals unencrypted information through collecting and inference.

B. Security Objectives

Because we adopt the encryption-before-outsourcing schemes [6] in our proposed architecture, the privacy of file content is already guaranteed unless adopted encryption is broken. Then, there are other security objects that we need to take into consideration:

a) Keyword Unvisiability: File index and trapdoor of queries are both transformed from extracted keywords. The real content keywords should also be protected from being known by remote servers.

b) Trapdoor unlinkability: Remote servers should be prevented from learning the content of query/files from repeated search tasks. Thus, trapdoor generation from query/files should be fully deterministic. Thus, the same query string or file should not be always transformed into the same trapdoor.

We adopt bloom filter structure and secure kNN to ensure security and privacy. Because the generalization of bloom filters is deterministic with a certain family of LSH functions, it cannot reach the requirement of 'Trapdoor unlinkability'. However, during the encryption with secure kNN, randomness is introduced into all trapdoors, which helps our scheme to be capable of these security and privacy criteria. Detailed proof can be simply borrowed from that provided in [19], [41].

VII. Experiments and Evaluation

In this section, we design groups of experiments to estimate the performance of our scheme. We use the real-world natural language dataset '20NewsGroups' [17] as the raw plaintext file source. Main programs are written in Python2.7. All experiments were performed on a server having an 'Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz' and 16GB available DDR4 memory space. All experiments are run on Ubuntu 16.04 LTS operation system.

A. Evaluation Metrics

Since parameters setting matters a lot in experiments, we specify it at first. Unless declared otherwise, involved paramerts are set by default as:

- $s = 2, u = 2$ for "Infection" stage in generalizing OPU
- each query contains 5 keywords
- bloom filters have in total 8000 bits
- in total $l = 20$ LSH functions are used to build bloom filters
- each query asks for 20 returned files
- e in Algorithm 1 is set to be 0.4

Each text file in the dataset '20newsgroup' has a set of keywords whose scale varies a lot. To relieve the negative effect brought by this bias, we first filter the dataset by referring to how many keywords a file contains. We select 3583 files from '20newsgroup', each of which has at least 200 keywords and at most 400 keywords. In total 41558 raw keywords are extracted from these selected raw files. After stemming, keywords still sum to more than 3000. Instead of simply setting a threshold of relevance score to calculate search accuracy, we introduce a more practical accuracy metric for evaluation, which named *overlap rate of top-k files*. Because output return top-k files

affect what the data user gets from his query, this metric is expected to be better estimated the search accuracy in data users' real experience. The accuracy under such metric is formulated as:

$$Accuracy = \frac{|\{f|f \in FP_k, f \in FE_k\}|}{k}, \quad (15)$$

where FP_k is the set of top-k files returned through traversing search on plaintext files, and FE_k is the set of top-k files returned through encrypted search. For instance, given the same query, the search performed on encrypted data and plaintext search return the same set of top-k files. Then $Accuracy$ reaches its upper limit, i.e., $Accuracy = 1$. When $k = 3$, if the plaintext search returns a collection of files $FP_k = \{F_1, F_3, F_5\}$ and the search on encrypted data returns $FE_k = \{F_1, F_3, F_6\}$, the accuracy should be $\frac{2}{3}$. We compare our proposed scheme with the most popular scheme based on "uni-gram" keyword decomposing [8].

B. Non-fuzzy Search

As a basic requirement, fuzzy search approaches should definitely show convincing performance when no typo or other misleading information is involved. We test the search accuracy and time efficiency on text file datasets of different volume. The experiment result is shown in Figure 6. The search time for both of the schemes are tested under the naive linear search for variable control.

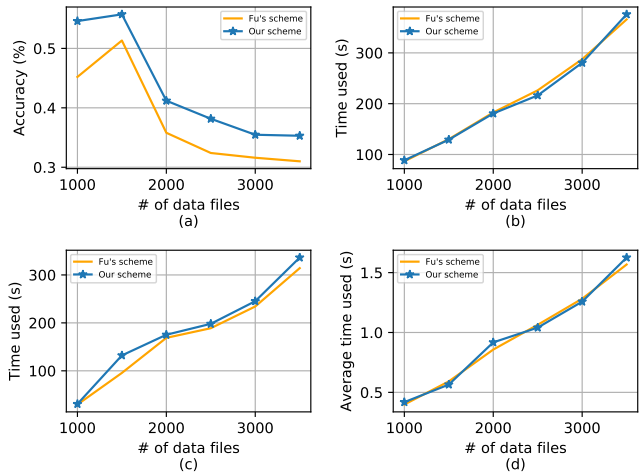

Fig. 6: Metrics under different volumes of dataset. a): accuracy of returned result; b): time used to generalize dictionary from raw dataset; c): time used to generalize bloomfilters of files; d): average time consumed to perform a query

For the different volume of the dataset, our proposed scheme achieves higher accuracy compared with Fu's scheme [8] with almost the same time performance. The main difference between these two schemes is how to represent a keyword, based on "uni-gram" of "order preserved uni-gram". Hence, the experiment proves that the introduction of OPU in our proposed scheme does not bring obvious extra computation overheads but it contributes to search accuracy to an evident extent.

In practice, the number of returned files, k, ought to be allowed fluctuant and adjusted by the data user. Therefore, a robust search scheme should achieve steady search accuracy when k changes. We control the value of k and maintain other variables unchanged in the experiment for comparison. The result is shown in Figure 7. Generally speaking, along with the increase of k (return files for a query), search accuracy increases and our scheme always achieves higher accuracy.

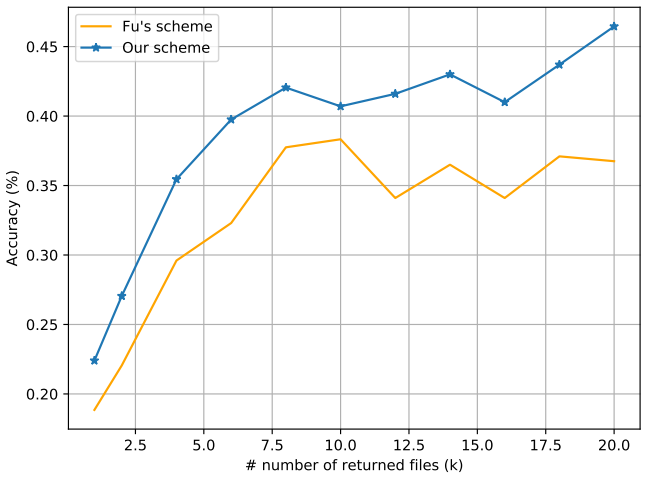

Fig. 7: Accuracy with different numbers of returned files (different values of k)

Similar to other searchable encryption mechanisms [8], [42], [43] based on Bloom Filter [31], another critical trade-off between time efficiency and accuracy is the size of bloom filter. A bigger bloom filter decreases the chance of hashing collision but it obviously increases the computation and memory storage overhead. To evaluate the influence of bloom filter size on our scheme, we set another group of experiments and the result is shown in Fig 8. Referring to the experiment, we find that our proposed scheme performs better in the set range of bloom filter size. The time efficiency is nearly the same as in that in [8]. On the other hand, the increased length of bloom filter brings no extra time consumption compared. It is because that OPU introduces no extra computation overheads into the bloom filter generalization stage.

At last, we also evaluate performance with different lengths of queries. The result is reported in Fig 9. Our proposed scheme performs steadily with different queries consisting of 1-10 keyword(s).

C. Fuzzy Search

During the generation of OPU, *"Infection"* is the most important mechanism to support fuzzy search for many different occasions as described in previous sections. It is one of the most important contributions in this paper. As the core of this step, "Infection Function" adopted is adjusted by two parameters: u and s in Eq 3. We change the value of u and s independently to evaluate their influence on search performance. The experiment result is reported in Fig 10, where two misspelled keywords are inserted into each query

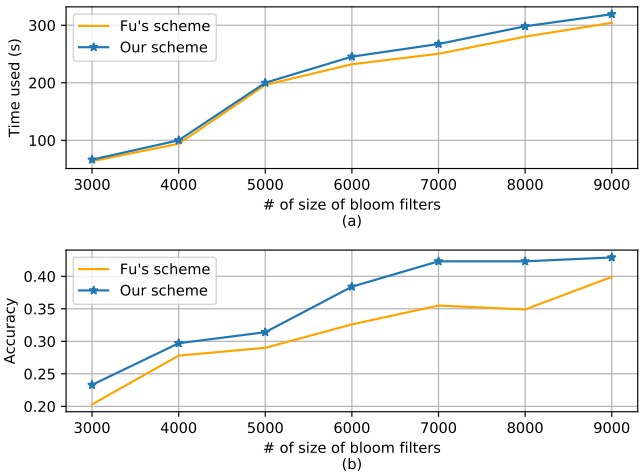

Fig. 8: Influence of the size of bloom filter (Len_{bf}) on the accuracy and time usage

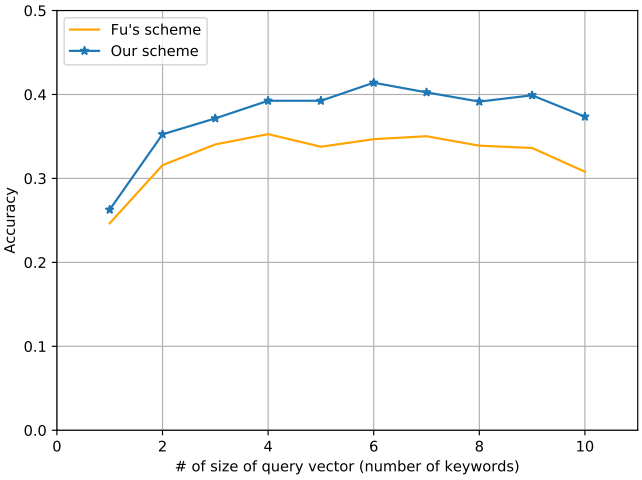

Fig. 9: Search accuracy with different size of query string (Len_{query})

string. The result argues that s has a very slight influence on accuracy when varies from 2 to 5. However, when u is set larger than 2, the accuracy reported in our scheme drops aggressively.

We provide an intuitive explanation for this phenomenon: a larger u allows the weight sharing between two more distant positions in a keyword but in practice, data user misspells keywords by exchanging letters on distant positions in a much lower possibility. For example, it is more likely for the keyword "listen" to be misspelled to be "lisetn" than "lestin". Hence, if u is set too larger, it may sacrifice the accuracy under most occasions to conform some rare extreme cases.

To compare time efficiency in fuzzy search, we set two groups of experiments, one for spelling typos and the other for anagrams. For misspelling, each query consists of Len_{query} keywords , and we select m keywords in the query. The selected keywords become mutants by three types of letter operation on a single letter: 1) letter replacement 2) neighboring letters exchange and 3) deletion or addition a letter. For example, the mutants of the word "search" in three cases respectively could be "seerch", "saerch" and "serch"/"searrch". With the keyword mutant involved in queries, we imitate the real fuzzy search scenes in practice. The experiment result is reported in Figure 11.

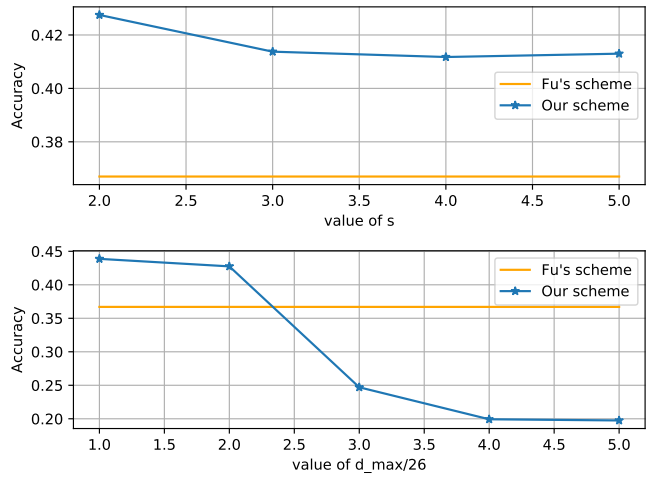

Fig. 10: Search accuracy under different value of s and u (= $d_{max}/26$) for infection

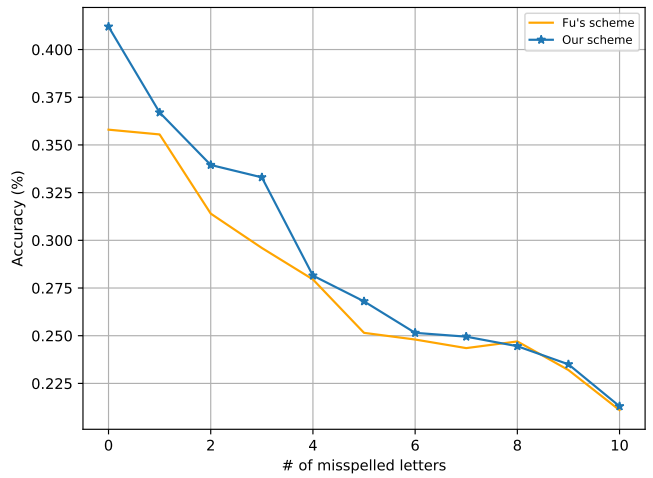

Fig. 11: Influence of misspellings on accuracy

Another interesting case in fuzzy search is when anagrams are involved in queries and files. As explained already, anagrams make many currently popular search schemes on encrypted data invalid. To evaluate the scheme performance with anagrams involved, we collect in a total of 500 pairs of anagrams from [44], and then insert N_A anagrams into files and queries in pair. Anagrams are inserted into F_A files. N_A and F_A are adjusted separately in two groups of experiments for variable control. Results of experiments are reported respectively in Fig 12 and in Fig 13.

Through these two groups of experiments, our scheme outperforms the scheme based on traditional 'uni-gram' more

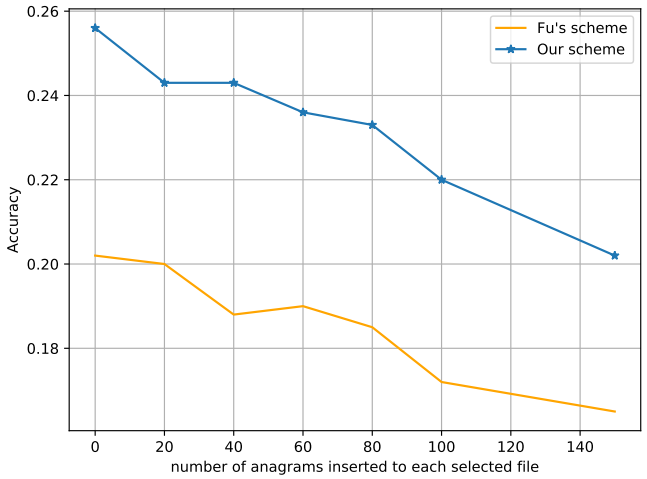

Fig. 12: Influence of number of file pairs to insert anagrams into (F_A) on accuracy

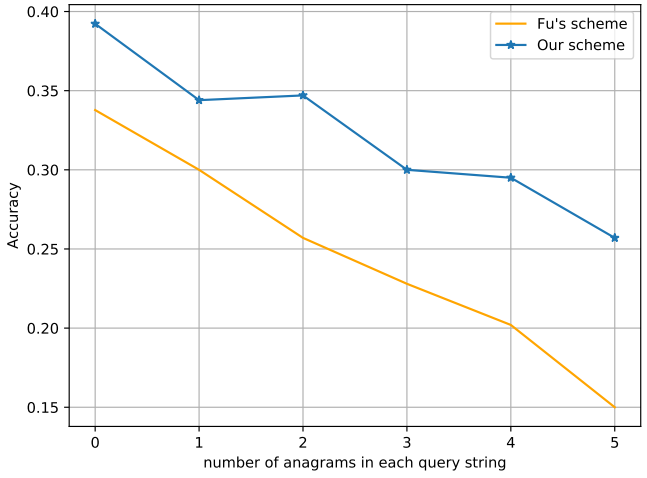

Fig. 13: Influence of number of anagrams in each query string (N_A) on accuracy

than for non-fuzzy search. We believe such improvement of accuracy obviously results from our proposed new mechanism to transform keywords in text to OPU.

D. Time efficiency

At last, we compare the search time consumption with HIT adopted and without it. From theoretical analysis, the time complexity is reduced from $\mathcal{O}(n)$ to $\mathcal{O}(log_n)$ and experiment result in Fig 14 conforms to the analysis well. Besides, comparing the result with previous experiments, we find HIT brings very slight harm to accuracy, which is acceptable in most practical occasions.

VIII. Conclusion

In this paper, we propose a novel scheme for multi-keyword search over outsource encrypted data, where a fuzzy search is well supported in many different cases. Innovative proposal in the paper does no harm to the privacy guarantee on outsourced data.

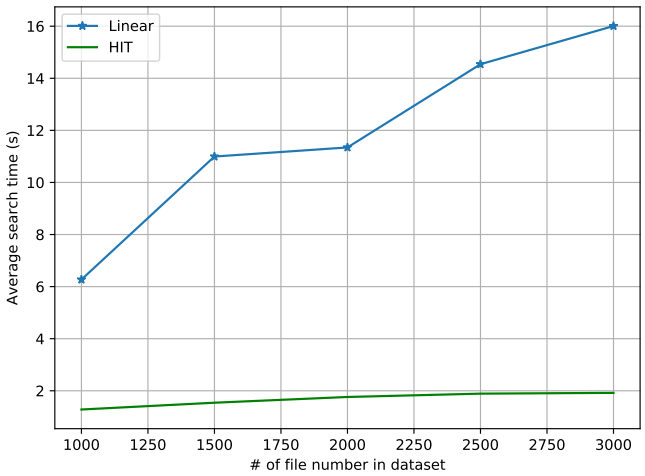

Fig. 14: Influence of the use of HIT on search time

Our contributions can be concluded into two aspects:

1) we propose a novel keyword decomposing scheme based on what is named "order-preserved uni-gram" (OPU), which eliminates many weaknesses of previous "uni-gram" and "n-gram" schemes.
2) we design a novel file indexing tree (HIT) which is based on hierarchical cluster tree while we improve the traditional K-means algorithm for data clustering. Thanks to the novel dynamic K-means algorithm, the indexing tree can be constructed more flexibleand with fewer parameters set manually.

Experiments on real-world data prove the effectiveness of our proposed architecture. OPU brings accuracy improvement to a great extent under various experiment settings, and the proposed HIT increases search time efficiency without obvious hurt on search accuracy,

References

[1] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless communications and mobile computing*, vol. 13, no. 18, pp. 1587–1611, 2013.

[2] S. S. Qureshi, T. Ahmad, K. Rafique *et al.*, "Mobile cloud computing as future for mobile applications-implementation methods and challenging issues," in *Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on*. IEEE, 2011, pp. 467–471.

[3] V. Namboodiri and T. Ghose, "To cloud or not to cloud: A mobile device perspective on energy consumption of applications," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a*. IEEE, 2012, pp. 1–9.

[4] Y. Sun, J. Zhang, Y. Xiong, and G. Zhu, "Data security and privacy in cloud computing," *International Journal of Distributed Sensor Networks*, vol. 10, no. 7, p. 190903, 2014.

[5] D. Chen and H. Zhao, "Data security and privacy protection issues in cloud computing," in *Computer Science and Electronics Engineering (ICCSEE), 2012 International Conference on*, vol. 1. IEEE, 2012, pp. 647–651.

[6] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *Infocom, 2010 proceedings IEEE*. Ieee, 2010, pp. 1–9.

[7] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," *International Journal of Engineering Research and Applications*, vol. 4, no. 7, pp. 1–5, 2014.

[8] 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*, vol. 11, no. 12, pp. 2706–2716, 2017.

[9] B. Wang, S. Yu, W. Lou, and Y. T. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *INFOCOM, 2014 Proceedings IEEE*, 2014, pp. 2112–2120.

[10] R. Brinkman, L. Feng, J. Doumen, P. H. Hartel, and W. Jonker, "Efficient tree search in encrypted data." *Information systems security*, vol. 13, no. 3, pp. 14–21, 2004.

[11] B. Wang, Y. Hou, M. Li, H. Wang, and H. Li, "Maple: scalable multi-dimensional range search over encrypted cloud data with tree-based index," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*. ACM, 2014, pp. 111–122.

[12] Z. Xia, X. Wang, X. Sun, and Q. Wang, "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data." *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 340–352, 2016.

[13] V. Gampala and S. Malempati, "A study on privacy preserving searching approaches on encrypted data and open challenging issues in cloud computing," *International Journal of Computer Science and Information Security*, vol. 14, no. 12, p. 294, 2016.

[14] 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 and Information Systems*, vol. 10, no. 2, pp. 667–684, 2013.

[15] Q. Zheng, S. Xu, and G. Ateniese, "Vabks: verifiable attribute-based keyword search over outsourced encrypted data," in *Infocom, 2014 proceedings IEEE*. IEEE, 2014, pp. 522–530.

[16] C. Wang, N. Cao, K. Ren, and W. Lou, "Enabling secure and efficient ranked keyword search over outsourced cloud data," *IEEE Transactions on parallel and distributed systems*, vol. 23, no. 8, pp. 1467–1479, 2012.

[17] K. Lang, "Newsweeder: Learning to filter netnews," in *Proceedings of the Twelfth International Conference on Machine Learning*, 1995, pp. 331–339.

[18] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *ACM Conference on Computer and Communications Security*, 2006, pp. 79–88.

[19] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *IEEE Symposium on Security and Privacy*, 2000, p. 44.

[20] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," in *IEEE International Conference on Distributed Computing Systems*, 2010, pp. 253–262.

[21] 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 and Distributed Systems*, vol. 25, no. 1, pp. 222–233, 2014.

[22] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *ACNS 04: International Conference on Applied Cryptography and Network Security*, 2004, pp. 31–45.

[23] H. Pang, J. Shen, and R. Krishnan, "Privacy-preserving similarity-based text retrieval," *ACM Transactions on Internet Technology (TOIT)*, vol. 10, no. 1, p. 4, 2010.

[24] Y. Elmehdwi, B. K. Samanthula, and W. Jiang, "Secure k-nearest neighbor query over encrypted data in outsourced environments," in *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*. IEEE, 2014, pp. 664–675.

[25] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis, "Secure knn computation on encrypted databases," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 2009, pp. 139–152.

[26] Z. Fu, X. Sun, Q. Liu, L. Zhou, and J. Shu, "Achieving efficient cloud search services: multi-keyword ranked search over encrypted cloud data supporting parallel computing," *IEICE Transactions on Communications*, vol. 98, no. 1, pp. 190–200, 2015.

[27] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *International Conference on Financial Cryptography and Data Security*. Springer, 2013, pp. 258–274.

[28] C. Chen, X. Zhu, P. Shen, J. Hu, S. Guo, Z. Tari, and A. Y. Zomaya, "An efficient privacy-preserving ranked keyword search method," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 4, pp. 951–963, 2016.

[29] 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*, vol. 27, no. 9, pp. 2546–2559, 2016.

[30] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, ser. SCG '04. New York, NY, USA: ACM, 2004, pp. 253–262. [Online]. Available: http://doi.acm.org/10.1145/997817.997857

[31] B. H. Bloom, *Space/time trade-offs in hash coding with allowable errors*. ACM, 1970.

[32] J. Macqueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of Berkeley Symposium on Mathematical Statistics and Probability*, 1967, pp. 281–297.

[33] M. Ester, H. P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," 1996.

[34] C. Biernacki, G. Celeux, and G. Govaert, "Assessing a mixture model for clustering with the integrated completed likelihood," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 7, pp. 719–725, 1998.

[35] J. B. Lovins, "Development of a stemming algorithm," *Mech. Translat. & Comp. Linguistics*, vol. 11, no. 1-2, pp. 22–31, 1968.

[36] P. Willett, "The porter stemming algorithm: then and now," *Program*, vol. 40, no. 3, pp. 219–223, 2006.

[37] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*. New York, NY, USA: McGraw-Hill, Inc., 1986.

[38] M. Calderbank, "The rsa cryptosystem: History, algorithm, primes," 2007.

[39] O. Goldreich, "Secure multi-party computation," *Manuscript. Preliminary version*, vol. 78, 1998.

[40] A. Swaminathan, Y. Mao, G. M. Su, H. Gou, A. L. Varna, S. He, M. Wu, and D. W. Oard, "Confidentiality-preserving rank-ordered search," in *ACM Workshop on Storage Security and Survivability, Storagess 2007, Alexandria, Va, Usa, October*, 2007, pp. 7–12.

[41] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li, "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," in *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*. ACM, 2013, pp. 71–82.

[42] S. M. Bellovin and W. R. Cheswick, "Privacy-enhanced searches using encrypted bloom filters." *IACR Cryptology ePrint Archive*, vol. 2004, p. 22, 2004.

[43] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *International Conference on Applied Cryptography and Network Security*. Springer, 2005, pp. 442–455.

[44] "www.manythings.org:everyday vocabulary anagrams," Published online, 2001. [Online]. Available: http://www.manythings.org/anagrams/