Enabling Efficient Fuzzy Keyword Search over Encrypted Data in Cloud Computing

Jin Li[1], Qian Wang[1], Cong Wang[1], Ning Cao[2], Kui Ren[1], and Wenjing Lou[2]

[1] Department of ECE, Illinois Institute of Technology
{jli25, qwang, cwang, kren2}@iit.edu
[2] Department of ECE, Worcester Polytechnic Institute,
{ncao, wjlou}@ece.wpi.edu

Abstract. As Cloud Computing becomes prevalent, more and more sensitive information are being centralized into the cloud. For the protection of data privacy, sensitive data usually have to be encrypted before outsourcing, which makes effective data utilization a very challenging task. Although traditional searchable encryption schemes allow a user to securely search over encrypted data through keywords and selectively retrieve files of interest, these techniques support only *exact* keyword search. That is, there is no tolerance of minor typos and format inconsistencies which, on the other hand, are typical user searching behavior and happen very frequently. This significant drawback makes existing techniques unsuitable in Cloud Computing as it greatly affects system usability, rendering user searching experiences very frustrating and system efficacy very low. In this paper, for the first time we formalize and solve the problem of effective fuzzy keyword search over encrypted cloud data while maintaining keyword privacy. Fuzzy keyword search greatly enhances system usability by returning the matching files when users' searching inputs exactly match the predefined keywords or the closest possible matching files based on keyword similarity semantics, when *exact* match fails. In our solution, we exploit edit distance to quantify keywords similarity and develop two advanced techniques on constructing fuzzy keyword sets, which achieve optimized storage and representation overheads. We further propose a brand new symbol-based trie-traverse searching scheme, where a multi-way tree structure is built up using symbols transformed from the resulted fuzzy keyword sets. Through rigorous security analysis, we show that our proposed solution is secure and privacy-preserving, while correctly realizing the goal of fuzzy keyword search. Extensive experimental results demonstrate the efficiency of the proposed solution.

1 Introduction

Cloud Computing, the new term for the long dreamed vision of computing as a utility [1], enables convenient, on-demand network access to a centralized pool of configurable computing resources (e.g., networks, applications, and services) that can be rapidly deployed with great efficiency and minimal management overhead [2]. The amazing advantages of Cloud Computing include: on-demand self-service, ubiquitous network access, location independent resource pooling, rapid resource elasticity, usage-based pricing, transference of risk, etc. [2, 3]. Thus, Cloud Computing could easily benefit its users in avoiding large capital outlays in the deployment and management of both software and hardware. Undoubtedly, Cloud Computing brings unprecedented paradigm shifting and benefits in the history of IT.

As Cloud Computing becomes prevalent, more and more sensitive information are being centralized into the cloud, such as emails, personal health records, private videos and photos, company finance data, government documents, etc. By storing their data into the cloud, the data owners can be relieved from the burden of data storage and maintenance so as to enjoy the on-demand high quality data storage service. However, the fact that data owners and cloud server are not in the same trusted domain may put the oursourced data at risk, as the cloud server may no longer be fully trusted in such a cloud environment due to a number of reasons: the cloud server may leak data information to unauthorized entities or be hacked. It follows that sensitive data usually should be encrypted prior to outsourcing for data privacy and combating unsolicited accesses.

However, data encryption makes effective data utilization a very challenging task given that there could be a large amount of outsourced data files. Moreover, in Cloud Computing, data owners may share their outsourced data with a large number of users. The individual users might want to only retrieve certain specific data files they are interested in during a given session. One of the most popular ways is to selectively retrieve files through keyword-based search instead of retrieving all the encrypted files back which is completely impractical in cloud computing scenarios. Such keyword-based search technique allows users to selectively retrieve files of interest and has been widely applied in plaintext search scenarios, such as Google search [4]. Unfortunately, data encryption restricts user's ability to perform keyword search and thus makes the traditional plaintext search methods unsuitable for Cloud Computing. Besides this, data encryption also demands the protection of keyword privacy since keywords usually contain important information related to the data files. Although encryption of keywords can protect keyword privacy, it further renders the traditional plaintext search techniques useless in this scenario.

To securely search over encrypted data, searchable encryption techniques have been developed in recent years [5–13]. Searchable encryption schemes usually build up an index for each keyword of interest and associate the index with the files that contain the keyword. By integrating the trapdoors of keywords within the index information, effective keyword search can be realized while both file content and keyword privacy are well-preserved. Although allowing for performing searches securely and effectively, the existing searchable encryption techniques do not suit for cloud computing scenario since they support only *exact* keyword search. That is, there is no tolerance of minor typos and format inconsistencies which, on the other hand, are typical user searching behavior and happen very frequently. As common practice, users may search and retrieve the data of their respective interests using any keywords they might come up with. It is quite common that users' searching input might not exactly match those pre-set keywords due to the possible typos, such as `Illinois` and `Ilinois`, representation inconsistencies, such as `PO BOX` and `P.O. Box`, and/or her lack of exact knowledge about the data. To give a concrete example, statistics from Google [4] shows that only less than 77% of the users' searching input exactly matched the name of `Britney`, detected in their spelling correction system within a three-month period. In other words, searching based on exact keyword match would return unnecessary failures for more than 23% search requests of `Britney`, making the searching system ineffective with low usability. This significant drawback of existing schemes signifies the important need for new techniques that support searching flexibility, tolerating both minor typos and format inconsistencies. That is, secure fuzzy search capability is demanded for achieving enhanced system usability in Cloud Computing. Although the importance of fuzzy search has received attention recently in the context of plaintext searching by information retrieval community [14–17], it is still being overlooked and remains to be addressed in the context of encrypted data search.

In this paper, we focus on enabling effective yet privacy-preserving fuzzy keyword search in Cloud Computing. To the best of our knowledge, we formalize for the first time the problem of effective fuzzy keyword search over encrypted cloud data while maintaining keyword privacy. Fuzzy keyword search greatly enhances system usability by returning the matching files when users' searching inputs exactly match the predefined keywords or the closest possible matching files based on keyword similarity semantics, when *exact* match fails. More specifically, we use edit distance to quantify keywords similarity and develop two novel techniques, i.e., an wildcard-based technique and a gram-based technique, for the construction of fuzzy keyword sets. Both techniques eliminate the need for enumerating all the fuzzy keywords and the resulted size of the fuzzy keyword sets is significantly reduced. Based on the constructed fuzzy keyword sets, we further propose an advanced symbol-based trie-traverse searching scheme, where a multi-way tree structure is built up using symbols transformed from the fuzzy keywords. Through rigorous security analysis, we show that the proposed solution is secure and privacy-preserving, while correctly realizing the goal of fuzzy keyword search. Extensive experimental results demonstrate the efficiency of the proposed solution.

The rest of paper is organized as follows: Section 2 introduces the system model, threat model, our design goal and briefly describes some necessary background for the techniques used in this paper. Section 3 summarizes the features of related work. Section 4 and 5 provide the detailed description of our proposed schemes. Section 6 and 7 present the security and performance analysis, respectively. Finally, Section 8 concludes the paper.

2 Related Work

Plaintext fuzzy keyword search. Recently, the importance of fuzzy search has received attention in the context of plaintext searching in information retrieval community [15–17]. They addressed this problem in the traditional information-access paradigm by allowing user to search without using try-and-see approach for finding relevant information based on approximate string matching. The approximate string matching algorithms among them can be classified into two categories: on-line and off-line. The on-line techniques, performing search without an index, are unacceptable for their low search efficiency, while the off-line approach, utilizing indexing techniques, makes it dramatically faster. A variety of indexing algorithms, such as suffix trees, metric trees and q-gram methods, have been presented. At the first glance, it seems possible for one to directly apply these string matching algorithms to the context of searchable encryption by computing the trapdoors on a character base within an alphabet. However, this trivial construction suffers from the dictionary and statistics attacks and fails to achieve the search privacy.

Searchable encryption. Traditional searchable encryption [5–13] has been widely studied in the context of cryptography. Among those works, most are focused on efficiency improvements and security definition formalizations. The first construction of searchable encryption was proposed by Song et al. [6], in which each word in the document is encrypted independently under a special two-layered encryption construction. Goh [7] proposed to use Bloom filters to construct the indexes for the data files. For each file, a Bloom filter containing trapdoors of all unique words is built up and stored on the server. To search for a word, the user generates the search request by computing the trapdoor of the word and sends it to the server. Upon receiving the request, the server tests if any Bloom filter contains the trapdoor of the query word and returns the corresponding file identifiers. To achieve more efficient search, Chang et al. [10] and Curtmola et al. [11] both proposed similar "index" approaches, where a single encrypted hash table index is built for the entire file collection. In the index table, each entry consists of the trapdoor of a keyword and an encrypted set of file identifiers whose corresponding data files contain the keyword. As a complementary approach, Boneh et al. [8] presented a public-key based searchable encryption scheme, with an analogous scenario to that of [6]. In their construction, anyone with the public key can write to the data stored on the server but only authorized users with the private key can search. As an attempt to enrich query predicates, conjunctive keyword search, subset query and range query over encrypted data, have also been proposed in [12, 18]. Note that all these existing schemes support only exact keyword search, and thus are not suitable for Cloud Computing.

Others. Private matching [19], as another related notion, has been studied mostly in the context of secure multiparty computation to let different parties compute some function of their own data collaboratively without revealing their data to the others. These functions could be intersection or approximate private matching of two sets, etc. [20]. The private information retrieval [21] is an often-used technique to retrieve the matching items secretly, which has been widely applied in information retrieval from database and usually incurs unexpectedly computation complexity.

3 Problem Formulation

3.1 System Model

In this paper, we consider a cloud data system consisting of data owner, data user and cloud server. Given a collection of n encrypted data files $\mathcal{C} = (\mathrm{F}_1, \mathrm{F}_2, \ldots, \mathrm{F}_\mathrm{N})$ stored in the cloud server, a predefined set of distinct keywords $W = \{w_1, w_2, ..., w_p\}$, the cloud server provides the search service for the authorized users over the encrypted data $\mathcal{C}$. We assume the authorization between

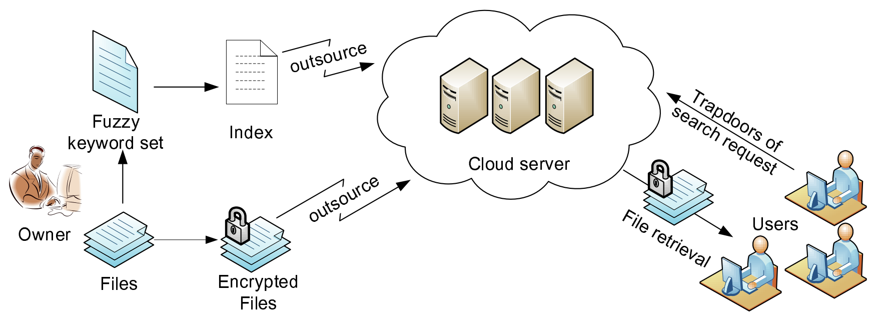


Fig. 1: Architecture of the fuzzy keyword search

the data owner and users is appropriately done. An authorized user types in a request to selectively retrieve data files of his/her interest. The cloud server is responsible for mapping the searching request to a set of data files, where each file is indexed by a file ID and linked to a set of keywords. The fuzzy keyword search scheme returns the search results according to the following rules: 1) if the user's searching input exactly matches the pre-set keyword, the server is expected to return the files containing the keyword[3]; 2) if there exist typos and/or format inconsistencies in the searching input, the server will return the closest possible results based on pre-specified similarity semantics (to be formally defined in section 3.4). An architecture of fuzzy keyword search is shown in the Fig. 1.

3.2 Threat Model

We consider a semi-trusted server. Even though data files are encrypted, the cloud server may try to derive other sensitive information from users' search requests while performing keyword-based search over $\mathcal{C}$. Thus, the search should be conducted in a secure manner that allows data files to be securely retrieved while revealing as little information as possible to the cloud server. In this paper, when designing fuzzy keyword search scheme, we will follow the security definition deployed in the traditional searchable encryption [11]. More specifically, it is required that nothing should be leaked from the remotely stored files and index beyond the outcome and the pattern of search queries.

3.3 Design Goals

In this paper, we address the problem of supporting efficient yet privacy-preserving fuzzy keyword search services over encrypted cloud data. Specifically, we have the following goals: i) to explore different mechanisms for constructing storage-efficient fuzzy keyword sets; ii) to design efficient and effective fuzzy search schemes based on the constructed fuzzy keyword sets; iii) to validate the security and evaluate the performance by conducting extensive experiments.

3.4 Preliminaries

Edit Distance There are several methods to quantitatively measure the string similarity. In this paper, we resort to the well-studied edit distance [22] for our purpose. The edit distance $\mathsf{ed}(w_1, w_2)$ between two words w_1 and w_2 is the number of operations required to transform one of them into the other. The three primitive operations are 1) Substitution: changing one character to another in a word; 2) Deletion: deleting one character from a word; 3) Insertion: inserting a single character into a word. Given a keyword w, we let $S_{w,d}$ denote the set of words w' satisfying $\mathsf{ed}(w, w') \leq d$ for a certain integer d.

[3] Note that we do not differentiate between files and file IDs in this paper.

Fuzzy Keyword Search Using edit distance, the definition of fuzzy keyword search can be formulated as follows: Given a collection of n encrypted data files $\mathcal{C} = (F_1, F_2, \ldots, F_N)$ stored in the cloud server, a set of distinct keywords $W = \{w_1, w_2, ..., w_p\}$ with predefined edit distance d, and a searching input (w, k) with edit distance k $(k \leq d)$, the execution of fuzzy keyword search returns a set of file IDs whose corresponding data files possibly contain the word w, denoted as FID_w: if $w = w_i \in W$, then return FID_{w_i}; otherwise, if $w \notin W$, then return $\{FID_{w_i}\}$, where $\mathsf{ed}(w, w_i) \leq k$. Note that the above definition is based on the assumption that $k \leq d$. In fact, d can be different for distinct keywords and the system will return $\{FID_{w_i}\}$ satisfying $\mathsf{ed}(w, w_i) \leq \min\{k, d\}$ if exact match fails.

Trapdoors of Keywords Trapdoors of the keywords can be realized by applying a one-way function f, which is similar as [5,7]: Given a keyword w_i and a secret key sk, we can compute the trapdoor of w_i as $T_{w_i} = f(sk, w_i)$.

4 Constructions of Effective Fuzzy Keyword Search in Cloud

The key idea behind our secure fuzzy keyword search is two-fold: 1) building up fuzzy keyword sets that incorporate not only the exact keywords but also the ones differing slightly due to minor typos, format inconsistencies, etc.; 2) designing an efficient and secure searching approach for file retrieval based on the resulted fuzzy keyword sets. In this section, we will focus on the first part, i.e., building storage-efficient fuzzy keyword sets to facilitate the searching process.

4.1 The Straightforward Approach

Before introducing our constructions of fuzzy keyword sets, we first propose a straightforward approach that achieves all the functions of fuzzy keyword search, which aims at providing an overview of how fuzzy search scheme works.

Assume $\mathit{\Pi}$=($\mathsf{Setup}(1^\lambda)$, $\mathsf{Enc}(sk, \cdot)$, $\mathsf{Dec}(sk, \cdot)$) is a symmetric encryption scheme, where sk is a secret key, $\mathsf{Setup}(1^\lambda)$ is the setup algorithm with security parameter λ, $\mathsf{Enc}(sk, \cdot)$ and $\mathsf{Dec}(sk, \cdot)$ are the encryption and decryption algorithms, respectively. The scheme goes as follows: We can begin by constructing the fuzzy keyword set $S_{w_i,d}$ for each keyword $w_i \in W$ $(1 \leq i \leq p)$ with edit distance d. The intuitive way to construct the fuzzy keyword set of w_i is to enumerate all possible words w'_i that satisfy the similarity criteria $\mathsf{ed}(w_i, w'_i) \leq d$, that is, all the words with edit distance d from w_i are listed. For example, the following is the listing variants after a substitution operation on the first character of keyword `CASTLE`: {`AASTLE`, `BASTLE`, `DASTLE`, $\cdots$, `YASTLE`, `ZASTLE`}. Based on the resulted fuzzy keyword sets, the fuzzy search is conducted as follows: 1) To build an index for w_i, the data owner computes trapdoors $T_{w'_i} = f(sk, w'_i)$ for each $w'_i \in S_{w_i,d}$ with a secret key sk shared between data owner and authorized users. The data owner also encrypts FID_{w_i} as $\mathsf{Enc}(sk, \mathrm{FID}_{w_i} \| w_i)$. The index table $\{(\{T_{w'_i}\}_{w'_i \in S_{w_i,d}}, \mathsf{Enc}(sk, \mathrm{FID}_{w_i} \| w_i))\}_{w_i \in W}$ and encrypted data files are outsourced to the cloud server for stroage; 2) To search with w, the authorized user computes the trapdoor T_w of w and sends it to the server; 3) Upon receiving the search request T_w, the server compares it with the index table and returns all the possible encrypted file identifiers $\{\mathsf{Enc}(sk, \mathrm{FID}_{w_i} \| w_i)\}$ according to the fuzzy keyword definition in section III-D. The user decrypts the returned results and retrieves relevant files of interest.

This straightforward approach apparently provides fuzzy keyword search over the encrypted files while achieving search privacy using the technique of secure trapdoors. However, this approach has serious efficiency disadvantages. The simple enumeration method in constructing fuzzy keyword sets would introduce large storage complexities, which greatly affect the usability. Recall that in the definition of edit distance, substitution, deletion and insertion are three kinds of operations in computation of edit distance. The numbers of all similar words of w_i satisfying $\mathsf{ed}(w_i, w'_i) \leq d$ for $d = 1, 2$ and 3 are approximately $2k \times 26$, $2k^2 \times 26^2$, and $\frac{4}{3}k^3 \times 26^3$, respectively. For example, assume there are 10^4 keywords in the file collection with average keyword length 10 and $d = 2$. The output length of hash function is 160 bits. The resulted storage cost for the index will be 30GB. Therefore, it brings forth the demand for fuzzy keyword sets with smaller size.

Algorithm 1 Wildcard-based Fuzzy Set Construction

```
1: procedure CreateWildcardFuzzySet($w_i, d$)
2:     if $d > 1$ then
3:         Call CreateWildcardFuzzySet($w_i, d-1$);
4:     end if
5:     if $d = 0$ then
6:         Set $S'_{w_i,d} = \{w_i\}$;
7:     else
8:         for ($k \leftarrow 1$ to $|S'_{w_i,d-1}|$) do
9:             for $j \leftarrow 1$ to $2 * |S'_{w_i,d-1}[k]| + 1$ do
10:                if $j$ is odd then
11:                    Set fuzzyword as $S'_{w_i,d-1}[k]$;
12:                    Insert $\star$ at position $\lfloor (j+1)/2 \rfloor$;
13:                else
14:                    Set fuzzyword as $S'_{w_i,d-1}[k]$;
15:                    Replace $\lfloor j/2 \rfloor$-th character with $\star$;
16:                end if
17:                if fuzzyword is not in $S'_{w_i,d-1}$ then
18:                    Set $S'_{w_i,d} = S'_{w_i,d} \cup \{\text{fuzzyword}\}$;
19:                end if
20:            end for
21:        end for
22:    end if
23: end procedure
24: end procedure
```

4.2 Advanced Techniques for Constructing Fuzzy Keyword Sets

To provide more practical and effective fuzzy keyword search constructions with regard to both storage and search efficiency, we now propose two advanced techniques to improve the straightforward approach for constructing the fuzzy keyword set. Without loss of generality, we will focus on the case of edit distance $d = 1$ to elaborate the proposed advanced techniques. For larger values of d, the reasoning is similar. Note that both techniques are carefully designed in such a way that while suppressing the fuzzy keyword set, they will not affect the search correctness, as will be described in section 5.

Wildcard-based Fuzzy Set Construction In the above straightforward approach, all the variants of the keywords have to be listed even if an operation is performed at the same position. Based on the above observation, we proposed to use an wildcard to denote edit operations at the same position. The wildcard-based fuzzy set of w_i with edit distance d is denoted as $S_{w_i,d}$=$\{S'_{w_i,0}, S'_{w_i,1}, \cdots, S'_{w_i,d}\}$, where $S'_{w_i,\tau}$ denotes the set of words w'_i with τ wildcards. Note each wildcard represents an edit operation on w_i. The procedure for wildcard-based fuzzy set construction is shown in Algorithm 1. For example, for the keyword `CASTLE` with the pre-set edit distance 1, its wildcard-based fuzzy keyword set can be constructed as $S_{\texttt{CASTLE},1} = \{$`CASTLE`, `*CASTLE`, `*ASTLE`, `C*ASTLE`, `C*STLE`, $\cdots$, `CASTL*E`, `CASTL*`, `CASTLE*`$\}$. The total number of variants on `CASTLE` constructed in this way is only $13 + 1$, instead of $13 \times 26 + 1$ as in the above exhaustive enumeration approach when the edit distance is set to be 1. Generally, for a given keyword w_i with length ℓ, the size of $S_{w_i,1}$ will be only $2\ell + 1 + 1$, as compared to $(2\ell + 1) \times 26 + 1$ obtained in the straightforward approach. The larger the pre-set edit distance, the more storage overhead can be reduced: with the same setting of the example in the straightforward approach, the proposed technique can help reduce the storage of the index from 30GB to approximately 40MB.

Gram-based Fuzzy Set Construction Another efficient technique for constructing fuzzy set is based on grams. The gram of a string is a substring that can be used as a signature for efficient approximate search [17]. While gram has been widely used for constructing inverted list

Algorithm 2 Gram-based Fuzzy Set Construction

```
1: procedure CreateGramFuzzySet($w_i, d$)
2:     if $d > 1$ then
3:         Call CreateGramFuzzySet($w_i, d - 1$);
4:     end if
5:     if $d = 0$ then
6:         $S'_{w_i,d} = \{w_i\}$;
7:     else
8:         for ($k \leftarrow 1$ to $|S'_{w_i,d-1}|$) do
9:             for $j \leftarrow 1$ to $2 * |S'_{w_i,d-1}[k]| + 1$ do
10:                Set fuzzyword as $S'_{w_i,d-1}[k]$;
11:                Delete the $j$-th character;
12:                if fuzzyword is not in $S'_{w_i,d-1}$ then
13:                    Set $S'_{w_i,d} = S'_{w_i,d} \cup \{\text{fuzzyword}\}$
14:                end if
15:            end for
16:        end for
17:    end if
18: end procedure
19: end procedure
```

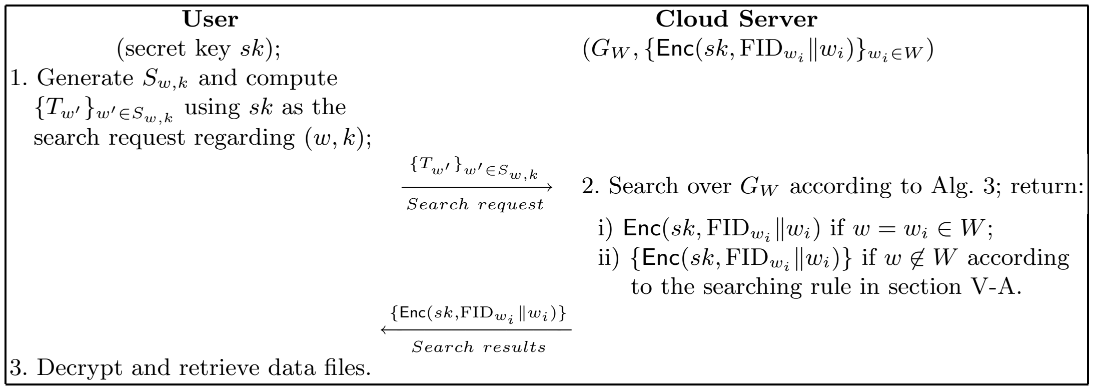
for approximate string search [23–25], we use gram for the matching purpose. We propose to utilize the fact that any primitive edit operation will affect at most one specific character of the keyword, leaving all the remaining characters untouched. In other words, the relative order of the remaining characters after the primitive operations is always kept the same as it is before the operations. With this significant observation, the fuzzy keyword set for a keyword w_i with ℓ single characters supporting edit distance d can be constructed as $S_{w_i,d} = \{S'_{w_i,\tau}\}_{0 \le \tau \le d}$, where $S'_{w_i,\tau}$ consists of all the $(\ell - \tau)$-gram from w_i and with the same relative order (we assume that $d \le \ell$). For example, the gram-based fuzzy set $S_{\texttt{CASTLE},1}$ for keyword `CASTLE` can be constructed as $\{$`CASTLE`, `CSTLE`, `CATLE`, `CASLE`, `CASTE`, `CASTL`, `ASTLE`$\}$. Generally, given a keyword w_i with ℓ single characters, the size of $S'_{w_i,\tau}$ is $C_\ell^{\ell-\tau}$, and the size of $S_{w_i,d}$ is $C_\ell^\ell + C_\ell^{\ell-1} + \cdots + C_\ell^{\ell-d}$. Compared to wildcard-based construction, gram-based construction can further reduce the storage of the index from 40MB to approximately 10MB under the same setting as in the wildcard-based approach. The procedure for gram-based fuzzy set construction is shown in Algorithm 2.

5 Efficient Fuzzy Searching Schemes

As shown in section 4, the size of fuzzy keyword set is greatly reduced using the proposed advanced techniques. However, the above constructions introduce another challenge: How to generate the search request and how to perform fuzzy keyword search? In the straightforward approach, because the index is created by enumerating all of fuzzy words for each keyword, there always exists matching words for the search request as long as the edit distance between them is equal or less than d. To design fuzzy search schemes based on the fuzzy keyword sets constructed from wildcard-based or gram-based technique, we compute the searching request regarding (w, k) as $\{T_{w'}\}_{w' \in S_{w,k}}$, where $S_{w,k} = \{S'_{w,0}, S'_{w,1}, \cdots, S'_{w,k}\}$ is generated in the same way as in the fuzzy keyword set construction. In this section, we will show how to achieve fuzzy keyword search based on the fuzzy sets constructed from the proposed advanced techniques. For simplicity, we will only consider the fixed d in our scheme designs. In this section, we start with some intuitive solutions, the analysis of which will motivate us to develop more efficient ones.

5.1 The Intuitive Solutions

Based on the storage-efficient fuzzy keyword set constructed as above, an efficient way to realize fuzzy keyword search is to use the traditional listing approach. Specifically, the scheme goes as



User (secret key sk);

1. Generate $S_{w,k}$ and compute $\{T_{w'}\}_{w' \in S_{w,k}}$ using sk as the search request regarding (w, k);

$\xrightarrow[Search\ request]{\{T_{w'}\}_{w' \in S_{w,k}}}$

Cloud Server $(G_W, \{\mathsf{Enc}(sk, \mathrm{FID}_{w_i} \| w_i)\}_{w_i \in W})$

2. Search over G_W according to Alg. 3; return:
   i) $\mathsf{Enc}(sk, \mathrm{FID}_{w_i} \| w_i)$ if $w = w_i \in W$;
   ii) $\{\mathsf{Enc}(sk, \mathrm{FID}_{w_i} \| w_i)\}$ if $w \notin W$ according to the searching rule in section V-A.

$\xleftarrow[Search\ results]{\{\mathsf{Enc}(sk, \mathrm{FID}_{w_i} \| w_i)\}}$

3. Decrypt and retrieve data files.

Fig. 2: Protocol for the symbol-based trie-traverse fuzzy keyword search

follows: (1) In the index building phase, for each keyword $w_i \in W$, the data owner computes trapdoors $T_{w_i'} = f(sk, w_i')$ for all $w_i' \in S_{w_i,d}$ with secret key sk. Then he computes $\mathsf{Enc}(sk, \mathrm{FID}_{w_i} \| w_i)$ and outsources the index table $\{\{T_{w_i'}\}_{w_i' \in S_{w_i,d}}, \mathsf{Enc}(sk, \mathrm{FID}_{w_i} \| w_i)\}$ together with the encrypted data files to the cloud server; (2) Assume an authorized user types in w as the searching input, with the pre-set edit distance k. The searching input is first transformed to a fuzzy set $S_{w,k}$. Then, the trapdoors $\{T_{w'}\}_{w' \in S_{w,k}}$ for each element $w' \in S_{w,k}$ are generated and submitted as the search request to the cloud server; (3) Upon receiving the search request, the server first compares $\{T_{w'}\}_{w' \in S'_{w,0}}$ with the index and returns the search result as $\mathsf{Enc}(sk, \mathrm{FID}_w \| w)$ if there exists an exact match. Otherwise, the server will compare all the elements of $\{T_{w'}\}_{w' \in S'_{w,\tau}}$ $(1 \le \tau \le k)$ with the index for the file collection and return all of the matched results $\{\mathsf{Enc}(sk, \mathrm{FID}_{w_i} \| w_i)\}$. The user now can obtain $\{\mathrm{FID}_{w_i} \| w_i\}$ through decryption and retrieve files of interest.

In fact, the correctness of the search result is based on the following observation: Assume the search request $\{T_{w'}\}_{w' \in S_{w,k}}$ for w and the index $\{T_{w_i'}\}_{w_i' \in S_{w_i,d}}$ for keyword w_i are computed with edit distance k and d, respectively. As long as the edit distance satisfies $\mathsf{ed}(w, w_i) \le k$, there would always exist at least one match between the elements in $S_{w,k}$ and the ones in $S_{w_i,d}$. Therefore, the search correctness is still maintained according to the fuzzy keyword search definition in 3.4 (The security proof of the listing approach will be given in section 6 for both the wildcard-based and the gram-based fuzzy keyword set constructions). The scheme also supports for variable d, and the results in section 6 will still hold by replacing the condition $\mathsf{ed}(w, w_i) \le k$ with $\mathsf{ed}(w, w_i) \le \min\{k, d\}$. To further hide the keyword length information, dummy trapdoors can be added such that all of the fuzzy sets have the same size. In this listing approach, both the searching cost and the storage cost at the server side are $O(\tau |W|)$, where $\tau = \max\{\{|S_{w_i,d}|\}_{w_i \in W}\}$.

Another solution is to explore Bloom filter [26] to represent the fuzzy keyword set $S_{w_i,d}$ for each keyword w_i with edit distance d, namely, the trapdoor set $\{T_{w_i'}\}_{w_i' \in S_{w_i,d}}$ is inserted into keyword w_i's Bloom filter as the index stored on the server. Now by binding the encrypted file identifiers $\mathsf{Enc}(sk, \mathrm{FID}_{w_i} \| w_i)$ to w_i's Bloom filter, a per keyword index is generated to track the data files. Upon receiving the search request $\{T_{w'}\}_{w' \in S_{w,k}}$, the server tests which Bloom filters contain 1's in all r locations for each element $w' \in S_{w,k}$ and returns the search results, assuming there are r independent hash functions $h_1, \ldots, h_r$ used in the construction of Bloom filter. In this solution, the server will only need to store a bit array of m bits instead of the trapdoor information for all fuzzy set regarding w_i. Compared to the listing scheme, both storage cost and searching cost are now $O(|W|)$. However, due to the property of Bloom filter, there exists probability of falsely recognizing an unrelated word w_i' as in $\{T_{w_i'}\}_{w_i' \in S_{w_i,d}}$. For a keyword w_i and its corresponding Bloom filter with a bit array of m bits, the probability of a false positive is then $f = (1 - (1 - 1/m)^{r|S_{w_i,d}|})^r \approx (1 - e^{-r|S_{w_i,d}|/m})^r$. While the Bloom filter above is built for each keyword $w_i \in W$, it can also be built based on each file. The intuition behind this idea is to insert fuzzy set $S_{w_i,d}$ of all the keywords belonging to the same file into a single Bloom filter, a search request $\{T_{w'}\}_{w' \in S_{w,k}}$ for (w, k) is conducted by testing all the words in $\{T_{w'}\}_{w' \in S_{w,k}}$ through each

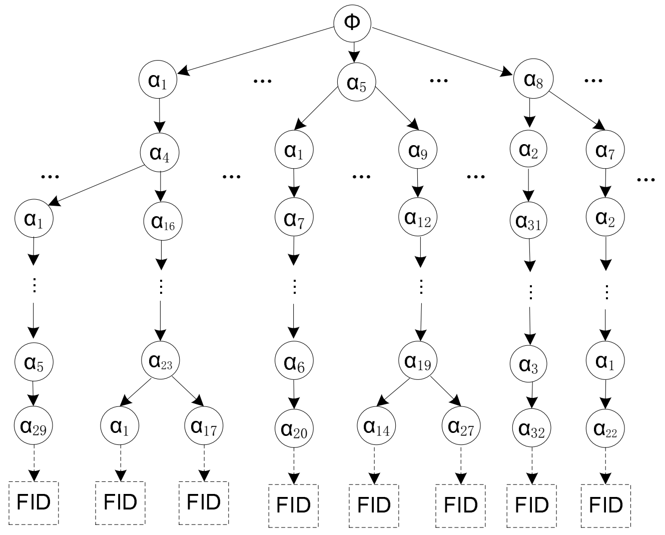


Fig. 3: An example of integrated symbol-based index for all words in the fuzzy keyword set.

file's Bloom filter. Note that the search cost associated with this solution is $O(|N|)$, where N is the number of data files.

5.2 The Symbol-based Trie-Traverse Search Scheme

To enhance the search efficiency, we now propose a symbol-based trie-traverse search scheme, where a multi-way tree is constructed for storing the fuzzy keyword set $\{S_{w_i,d}\}_{w_i \in W}$ over a finite symbol set. The key idea behind this construction is that all trapdoors sharing a common prefix may have common nodes. The root is associated with an empty set and the symbols in a trapdoor can be recovered in a search from the root to the leaf that ends the trapdoor. All fuzzy words in the trie can be found by a depth-first search. Assume $\Delta = \{\alpha_i\}$ is a predefined symbol set, where the number of different symbols is $\mid \Delta \mid = 2^n$, that is, each symbol $\alpha_i \in \Delta$ can be denoted by n bits. The scheme, as described in Fig. 2, works as follows:

(1) Assume the data owner wants to outsource the file collection $\mathcal{C}$ with keyword set W, he computes $T_{w'_i}$ as $\alpha_{i_1} \cdots \alpha_{i_{l/n}}$ for each $w'_i \in S_{w_i,d}$ $(i = 1, \cdots, p)$, where l is the output length of one-way function $f(x)$. A tree G_W covering all the fuzzy keywords of $w_i \in W$ is built up based on symbols in Δ. The data owner attaches the $\mathsf{Enc}(sk, \mathrm{FID}_{w_i} \| w_i)$ to G_W for $i = 1, \ldots, p$ and outsources these information to the cloud server;

(2) To search files containing w with edit distance k, the user computes $T_{w'}$ for each $w' \in S_{w,k}$ and sends $\{T_{w'}\}_{w' \in S_{w,k}}$ to the server;

(3) Upon receiving the request, the server divides each $T_{w'}$ into a sequence of symbols, performs the search over G_W using Algorithm 3 and returns $\{\mathsf{Enc}(sk, \mathrm{FID}_{w_i} \| w_i)\}$ to the user.

Note that by dividing the keying hash value into l/n parts, each n-bit of the hash value represents a symbol in Δ. The hash value of each fuzzy word $w'_i \in S_{w_i,d}$ is deterministic because with the same input sk and w'_i, the output $\alpha_{i_1} \cdots \alpha_{i_{l/n}}$ is unique. Moreover, no information about w_i will be leaked from the output $\alpha_{i_1} \cdots \alpha_{i_{l/n}}$. In this scheme, the paths of trapdoors for different keywords are integrated by merging all the paths with the same prefix into a single trie to support more efficient search. The encrypted file identifiers will be indexed according to its address or name and the index information will be stored aat the ending node of the corresponding path. Such an example of symbol-based trie is given in Fig. 3. With the returned search results $\{\mathsf{Enc}(sk, \mathrm{FID}_{w_i} \| w_i)\}$, the user may retrieve the files of his interest after decrypt and obtain $\{\mathrm{FID}_{w_i} \| w_i\}$. For each request, the search cost only $O(l/n)$ at the server side, which has nothing to do with the number of files or the size of related keywords.

Algorithm 3 SearchingTree

```
1: procedure SearchingTree($\{T'_w\}$)
2:     for $i \leftarrow 1$ to $|\{T'_w\}|$ do
3:         set $currentnode$ as $root$ of $G_w$;
4:         for $j \leftarrow 1$ to $l/n$ do
5:             Set $\alpha$ as $\alpha_j$ in the $i$-th $T'_w$;
6:             if no $child$ of $currentnode$ contains $\alpha$ then
7:                 break;
8:             end if
9:             Set $currentnode$ as $child$ containing $\alpha$;
10:        end for
11:        if $currentnode$ is $leafnode$ then
12:            Append $currentnode.FIDs$ to $resultIDset$;
13:            if $i = 1$ then
14:                return $resultIDset$;
15:            end if
16:        end if
17:    end for
18:    return $resultIDset$;
19: end procedure
20: end procedure
```

5.3 Supporting Multiple Users

In this section, we consider a natural extension from the previous single-user setting to multi-user setting, where a data owner stores a file collection on the cloud server and allows an arbitrary group of users to search over his file collection. Let $\mathsf{BE} = (\mathsf{KeyGen}_{BE}, \mathsf{Enc}_{BE}, \mathsf{Dec}_{BE})$ be a broadcast encryption scheme providing revocation-scheme security against a coalition of all revoked users [27]. Additionally, let π be a pseudo-random permutation.

The index computation is almost the same as the single-user setting except for each trapdoor T_w, a pseudo-random permutation $\pi(\xi, \cdot)$ is applied with a secret key ξ which is encrypted with the broadcast encryption scheme and stored on the server.

To search with (w, k), an authorized user computes trapdoors $\{\pi(\xi, T_{w'})\}_{w' \in S_{w,k}}$ with a secret key ξ which is distributed by the data owner.

Upon receiving the request, the server recovers the trapdoors by computing π^{-1} $(\xi,\pi(\xi,T_{w'}))$. Because the key ξ currently used is only known by the server and the set of currently authorized users, the search request is valid only if the user is not revoked. Each time a user is revoked, the data owner picks a new ξ and stores it on the server encrypted such that only non-revoked users can decrypt it. After the update, the server will use the new ξ to compute $\pi^{-1}(\xi, \cdot)$ for following search requests. Furthermore, the revoked users cannot recover the current ξ and thus, their requests will not yield valid trapdoors after the server applies $\pi^{-1}(\xi, \cdot)$.

6 Security Analysis

In this section, we analyze the correctness and security of the proposed fuzzy keyword search schemes. At first, we show the correctness of the schemes in terms of two aspects, that is, completeness and soundness.

Theorem 1. *The wildcard-based schemes satisfy both completeness and soundness. Specially, upon receiving the request of w, all of the keywords $\{w_i\}$ will be returned if and only if $\mathsf{ed}(w, w_i) \leq k$.*

The proof can be derived based on the following Lemma:

Lemma 1. *The intersection of the fuzzy sets $S_{w_i,d}$ and $S_{w,k}$ for w_i and w is not empty if and only if $\mathsf{ed}(w, w_i) \leq k$.*

Proof. First, we show that $S_{w_i,d} \cap S_{w,k}$ is not empty when $\mathsf{ed}(w, w_i) \leq k$. To prove this, it is enough to find an element in $S_{w_i,d} \cap S_{w,k}$. Let $w = a_1 a_2 \cdots a_s$ and $w_i = b_1 b_2 \cdots b_t$, where all these a_i and b_j are single characters. After $\mathsf{ed}(w, w_i)$ edit operations, w can be changed to w_i according to the definition of edit distance. Let $w^* = a_1^* a_2^* \cdots a_m^*$, where $a_i^* = a_j$ or $a_i^* = *$ if any operation is performed at this position. Since the edit operation is inverted, from w_i, the same positions containing wildcard at w^* will be performed. Because $\mathsf{ed}(w, w_i) \leq k$, w^* is included in both $S_{w_i,d}$ and $S_{w,k}$, we get the result that $S_{w_i,d} \cap S_{w,k}$ is not empty.

Next, we prove that $S_{w_i,d} \cap S_{w,k}$ is empty if $\mathsf{ed}(w, w_i) > k$. The proof is given by reduction. Assume there exists an w^* belonging to $S_{w_i,d} \cap S_{w,k}$. We will show that $\mathsf{ed}(w, w_i) \leq k$, which reaches a contradiction. First, from the assumption that $w^* \in S_{w_i,d} \cap S_{w,k}$, we can get the number of wildcard in w^*, which is denoted by n^*, is not greater than k. Next, we prove that $\mathsf{ed}(w, w_i) \leq n^*$. We will prove the inequality with induction method. First, we prove it holds when $n^* = 1$. There are nine cases should be considered: If w^* is derived from the operation of deletion from both w_i and w, then, $\mathsf{ed}(w_i, w) \leq 1$ because the other characters are the same except the character at the same position. If the operation is deletion from w_i and substitution from w, we have $\mathsf{ed}(w_i, w) \leq 1$ because they will be the same after at most one substitution from w_i. The other cases can be analyzed in a similar way and are omitted. Now, assuming that it holds when $n^* = \gamma$, we need to prove it also holds when $n^* = \gamma + 1$. If $\hat{w}^* = a_1^* a_2^* \cdots a_n^* \in S_{w_i,d} \cap S_{w,k}$, where $a_i^* = a_j$ or $a_i^* = *$. For an wildcard at position t, cancel the underlying operations and revert it to the original characters in w_i and w at this position. Assume two new elements w_i^* and w^* are derived from them respectively. Then perform one operation at position t of w_i^* to make the character of w_i at this position be the same with w, which is denoted by w_i'. After this operation, w_i^* will be changed to w^*, which has only k wildcards. Therefore, we have $\mathsf{ed}(w_i', w) \leq \gamma$ from the assumption. We know that $\mathsf{ed}(w_i', w) \leq \gamma$ and $\mathsf{ed}(w_i', w_i) = 1$, based on which we know that $\mathsf{ed}(w_i, w) \leq \gamma + 1$. Thus, we can get $\mathsf{ed}(w, w_i) \leq n^*$. It renders the contradiction $\mathsf{ed}(w, w_i) \leq k$ because $n^* \leq k$. Therefore, $S_{w_i,d} \cap S_{w,k}$ is empty if $\mathsf{ed}(w, w_i) > k$.

The following Theorem says that in the gram-based search schemes, the satisfied keywords will be returned, as well as some keywords which are not desired. In concrete, the returned keywords may also include keyword w_i as the answer for the request of (w, k) even if $\mathsf{ed}(w, w_i) > k$. For example, to search with the request $(\mathtt{CAT}, 1)$, the keyword `CASTLE` will be returned if $(\mathtt{CASTLE}, 3)$ is stored in the index, even if the edit distance of `CAT` and `CASTLE` is greater than 1. Thus, with the returned results, the user should filter the keyword set by further computing the edit distance.

Theorem 2. *The gram-based fuzzy keyword search schemes satisfy the completeness. Specially, upon receiving the request of (w, k), all of the keywords w_i will be returned if $\mathsf{ed}(w, w_i) \leq k$.*

The proof of the Theorem can be obtained through the following Lemma:

Lemma 2. *Assume $S_{w_i,d}$ and $S_{w,k}$ are built with edit distance d and k for w_i and w, respectively. The set $S_{w_i,d} \cap S_{w,k}$ is not empty when $\mathsf{ed}(w_i, w) \leq \min\{d, k\}$.*

Proof. Let $\mathsf{ed}(w_1, w_2) = d$, after at most d operations on the characters of w_1, it can be transformed to w_2. Without loss of generality, assume $|w_1| \geq |w_2|$. It means that the remaining $|w_1| - d$ characters in w_1 are untouched and they are equal to a $(|w_1| - d)$-character sequence in w_2, which belongs to $S_{w_1,k_1} \cap S_{w_2,k_2}$ when $d \leq \min\{k_1, k_2\}$.

For the security analysis, we will use the security model of [11] by using the simulation-based proof technique. There are two kinds of attacks defined by [11], that is, non-adaptive attack and adaptive attack. In the non-adaptive attack, it only considers adversaries that make search queries without taking into account the trapdoors and search outcomes of previous searches. The adversaries in the adaptive attack, however, can choose their queries as a function of previously obtained trapdoors and search outcomes. In this paper, we show the security proof against the non-adaptive attack. To achieve the adaptive security, the technique of [11] can be applied in our constructions similarly. We first introduce some auxiliary notions and definitions used in [11] and adapt some of them for our fuzzy keyword search encryption scheme.

History: an interaction between the data user and the cloud server, which is determined by a file collection $\mathcal{D} = (F_1, F_2, \ldots, F_n)$ and a set of keywords searched by the client, denoted as $H_q = (\mathcal{D}, w_1, \ldots, w_q)$.

View: what the cloud server actually sees during the interaction of a given history H_q under some secret key K, including the index $\mathcal{I}^*$ of $\mathcal{D}$, the trapdoors of the queried keywords $\{T_{w'}\}_{w' \in \{S_{w_1}, \cdots, S_{w_q}\}}$, the number of files, their ciphertexts $(C_1, \cdots, C_n)$, denoted as $V_K(H_q)$.

Trace: the precise information leaked about the history H_q, including the file identifiers of keyword w_i, which is denoted as $\{FID_{w_i}\}_{1 \le i \le q}$ (outcome of each search), the equality pattern $\mathit{\Pi}_q$ for each search (search pattern), and the size of the encrypted files, denoted as $Tr(H_q)$, where $\mathit{\Pi}_q$ is regarded as a symmetric binary matrix where $\mathit{\Pi}_q[i,j] = 1$ if any element in $\{T_{w'}\}_{w' \in S_{w_i}}$ matches an element in $\{T_{w'}\}_{w' \in S_{w_j}}$, and $\mathit{\Pi}_q[i,j] = 0$ otherwise, for $1 \le i, j \le q$.

We have the security result for our search schemes as follows:

Theorem 3. *Both our fuzzy keyword search schemes meet the semantic security.*

Proof. Due to the space limitation, we only give the proof for the fuzzy keyword search scheme in Section V-A. The proof of other schemes follow similarly, and thus omitted here.

To prove the semantic security of our fuzzy keyword searchable encryption scheme, it is equivalent to describe a simulator $\mathcal{S}$ such that, given $Tr(H_q)$, it can simulate the adversary's view of $V_K(H_q)$ with probability negligibly close to 1, for any $q \in \mathbb{N}$, any H_q and $K \overset{R}{\leftarrow} KeyGen(1^k)$. Note that $Tr(H_q) = (\mathrm{FID}(w_1), \ldots, \mathrm{FID}(w_q), \{|C_i|\}_{1 \le i \le n}, \mathit{\Pi}_q)$, and $V_K(H_q) = (C_1, \ldots, C_n, \mathcal{I}^*, \{T_{w'}\}_{w' \in \{S_{w_1}, \cdots, S_{w_q}\}})$. We will show that the simulator S can generate a view V_q^* with trace $Tr(H_q)$, which is indistinguishable from $V_K(H_q)$. Further, note that the security parameters of the PRF $f(\cdot)$, hash functions $\pi(\cdot)$ and encryptions $\mathsf{Enc}(\cdot)$ are known to $\mathcal{S}$. Without loss of generality, we denote the identifier of individual file as $\mathrm{id}(F_i) = i$ where $\{1 \le i \le n\}$.

For $q = 0$, S builds the set $V_0^* = \{1, 2, \cdots, n, e_1^*, e_2^*, \cdots, e_n^*, \mathcal{I}^*, C_1^*, \ldots, C_n^*,\}$ such that $e_i^* \leftarrow \{0,1\}^{|F_i|}$ and $\mathcal{I}^* = (\mathsf{T}^*, \mathsf{C}^*)$, where T^* and C^* are generated as follows:

- To generate T^*, for $1 \le i \le \sum_{i=1}^{n} \mid S_{w_i,d} \mid$, S selects a random t_i with the same length of trapdoor, and sets $\mathsf{T}^*[i] = t_i$;
- To generate C^*, for $1 \le i \le n$, S selects a random e'_i with the same length of $|FID_{w_i}|$ and sets $\mathsf{C}^*[i] = e'_i$;
- To generate $\{C_i\}_{1 \le i \le n}$, S chooses a random $e_i^* \in \{0,1\}^{|F_i|}$ and sets $C_i^* = e_i^*$.

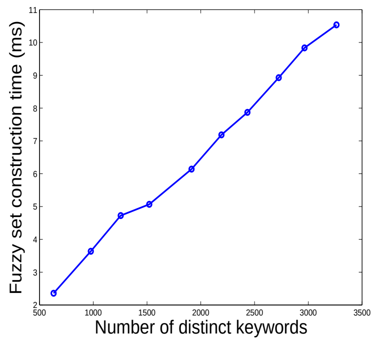
Built in this way, we claim that no probabilistic polynomial-time adversary can distinguish between V_0^* and $V_K(H_0)$. Otherwise, an algorithm can be built to distinguish between at least one of the elements of V_0^* and $V_K(H_0)$, which is impossible because of the semantic security of the symmetric encryption and the pseudo-randomness of the trapdoor.

For $q \ge 1$, S builds the set $V_q^* = \{1, 2, \cdots, n, e_1^*, e_2^*, \cdots, e_n^*, \mathcal{I}^*, \{T_{w'}^*\}_{w' \in \{S_{w_1}, \cdots, S_{w_q}\}}$ such that $e_i^* \leftarrow \{0,1\}^{|F_i|}$ and $\mathcal{I}^* = (\mathsf{T}^*, \mathsf{C}^*)$, where T^* and C^* are generated as follows:

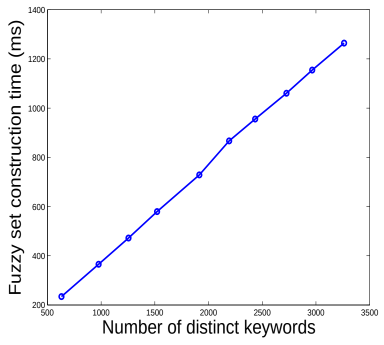
- For $1 \le i \le \sum_{i=1}^{n} \mid S_{w_i,d} \mid$, S selects a random t_i and sets $\mathsf{T}^*[i] = t_i$ as the simulation of T^*;
- For $1 \le i \le n$, S selects a random $e'_i \in Z_p$ and sets $\mathsf{C}^*[i] = e'_i$ as the simulation of C^*. Then, attach $C^*[i]$ behind $T^*[i]$;
- To generate $\{T_{w'}^*\}_{w' \in \{S_{w_i}\}}$, it computes $\{f(K, w')\}$ and attaches an encrypted FID_{w_i} from $Tr(H_q)$.

It also simulates $\{C_i^*\}_{1 \le i \le n}$ by choosing a random $e_i \in \{0,1\}^{|F_i|}$ and sets $C_i^* = e_i$. Built in this way, we claim that no probabilistic polynomial-time adversary can distinguish between V_q^* and $V_K(H_q)$. Otherwise, an algorithm can be built to distinguish between at least one of the elements of V_q^* and $V_K(H_q)$, which is impossible because of the semantic security of the symmetric encryption and the pseudo-randomness of the trapdoor.

As a result, the simulator generates the view of $V_q^* = \{1, 2, \cdots, n, e_1^*, e_2^*, \cdots, e_n^*, \mathcal{I}^*, \{T_{w'}\}_{w' \in \{S_{w_1}, \cdots, S_{w_n}\}}\}$. The correctness of the constructed view is easy to demonstrate by searching



(a)



(b)

Fig. 4: Fuzzy set construction time using wildcard-based approach: (a) $d = 1$, (b) $d = 2$.

on $\mathcal{I}^*$ via $\{T^*_{w'}\}_{w' \in S_{w_i}}$ for each i. We claim that there is no probabilistic polynomial-time adversary can distinguish between V^*_q and $V_K(H_q)$. Otherwise, based on a standard hybrid argument, the adversary could distinguish at least one of the elements of V^*_q and $V_K(H_q)$. This is impossible because each element in V^*_q is indistinguishable from its counterpart in $V_K(H_q)$. More specifically, the simulated encrypted ciphertext is indistinguishable because of the semantic security of the symmetric encryption. The indistinguishability of index is based on the assumption that no one could tell the difference between the output of pseudo-random function and a random string.

Based on the above analysis, we have proven the result of this theorem.

7 Performance Analysis

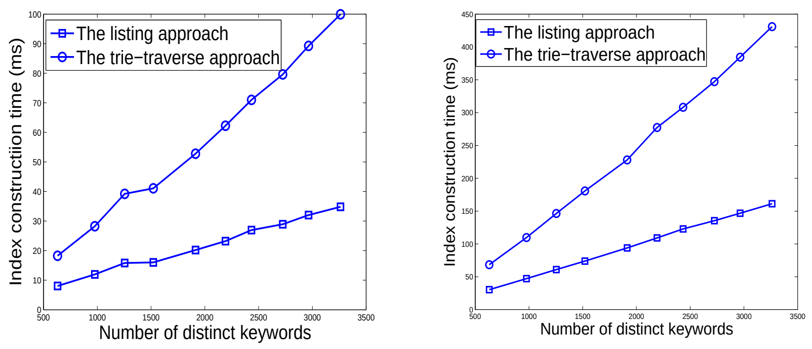
We conducted a thorough experimental evaluation of the proposed techniques on real data set: the recent ten years' IEEE INFOCOM publications. The data set includes about $2,600$ publications. We extract the words in the paper titles to construct the core keyword set in our experiment. The total number of keywords is $3,262$ and their average word length is 7.44. Our experiment is conducted on a Linux machine with an Intel Core 2 processor running at 1.86GHz and 2G DDR2-800 memory. The performance of our scheme is evaluated regarding the time cost of fuzzy set construction, the time and storage cost of index construction, the search time of the listing approach and the symbol-based trie-traverse approach.

7.1 Performance of Fuzzy Keyword Set Construction

In section 4, we propose two advanced techniques for the construction of fuzzy keyword sets, which both can be employed in our proposed fuzzy search schemes. In our experiment, we only focus on the wildcard-based fuzzy set construction because it provides the sound results compared to the gram-based fuzzy set construction as discussed in section 6. Fig. 4 shows the fuzzy set construction time by using the wildcard-based approach with edit distance $d = 1$ and 2. We can see that in both cases, the wildcard-based approach is very efficient and the construction time increases linearly with the number of keywords. The cost of constructing fuzzy keyword set under $d = 1$ is much less than the case of $d = 2$ due to the smaller set of possible wildcard-based words.

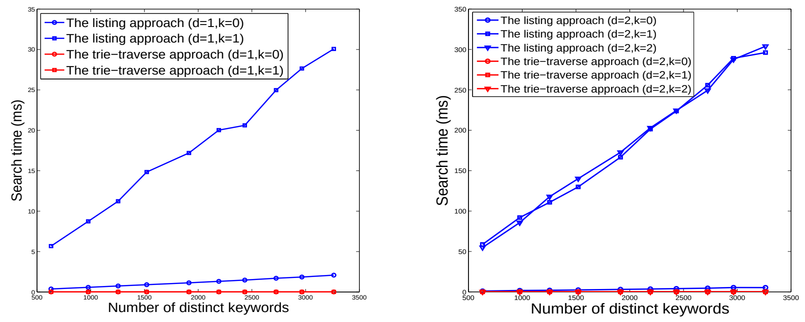
7.2 Performance of Fuzzy Keyword Search

Efficiency of Index Construction Given the fuzzy keyword set constructed using wildcard-based technique, we measure the time cost of index construction for the listing approach and symbol-based trie-based approach. In our experiment, we choose $n = 4$ and use SHA-1 as our hash function with output length of $l = 160$ bits. The resulted height of the searching tree is $l/n = 40$.



(a) (b)

Fig. 5: (a) Index construction time for edit distance $d = 1$. (b) Index construction time for edit distance $d = 2$.



(a) (b)

Fig. 6: (a) Searching time for edit distance $d = 1$. (b) Searching time for edit distance $d = 2$.

Fig. 5 shows the index construction time for edit distance $d = 1$ and $d = 2$. Similar to the fuzzy keyword set construction, the index construction time also increases linearly with the number of distinct keywords. Compared to the listing approach, the index construction of the trie-traverse approach includes the process of building the searching tree additionally, thus its time cost is larger than that of listing approach. However, the whole index construction process is conducted off-line, thus it will not affect the searching efficiency. Table 1 shows the index storage cost of the two approaches. The symbol-based trie-traverse approach consumes more storage space than the listing approach due to its multi-way tree structure. This additional storage cost, however, is not a main issue in our setting, as such index information only take up a small amount of storage space on the cloud server.

Efficiency of Search We evaluate the search performance as the number of keywords increases. Fig. 6 shows the comparisons of average search time between the listing approach and the symbol-based trie-traverse approach. According to the definition of fuzzy keyword search, the types of searching inputs (e.g., $k = 0$ or $k = 1$) may affect the searching result. The experimental results show that for the fixed value of d, better search efficiency can be achieved when the search input exactly matches some predefined keyword. This is because $S'_{w,0}$ is always searched first during the searching process. Note that for both $d = 1$ and $d = 2$, the trie-based search approach is much more efficient than the listing approach. These results validate our analysis and show that our proposed solution is very efficient and effective in supporting fuzzy keyword search over encrypted cloud data.

Table 1: The index storage cost.

	Index size ($d = 1$)	Index size ($d = 2$)
The listing approach	1.6MB	6.6MB
The trie-traverse approach	13.0MB	48.8MB

8 Conclusion

In this paper, for the first time we formalize and solve the problem of supporting efficient yet privacy-preserving fuzzy search for achieving effective utilization of remotely stored encrypted data in Cloud Computing. We design two advanced techniques (i.e., wildcard-based and gram-based techniques) to construct the storage-efficient fuzzy keyword sets by exploiting two significant observations on the similarity metric of edit distance. Based on the constructed fuzzy keyword sets, we further propose a brand new symbol-based trie-traverse searching scheme, where a multi-way tree structure is built up using symbols transformed from the resulted fuzzy keyword sets. Through rigorous security analysis, we show that our proposed solution is secure and privacy-preserving, while correctly realizing the goal of fuzzy keyword search. Extensive experimental results demonstrate the efficiency of our solution.

As our ongoing work, we will continue to research on security mechanisms that support 1) search semantics that takes into consideration conjunction of keywords, sequence of keywords, and even the complex natural language semantics to produce highly relevant search results. and 2) search ranking that sorts the searching results according to the relevance criteria.

References

1. D. Parkhill, "The challenge of the computer utility," Addison-Wesley Educational Publishers Inc., US, 1966.
2. P. Mell and T. Grance, "Draft nist working definition of cloud computing," Referenced on June. 3rd, 2009 Online at http://csrc.nist.gov/groups/SNS/cloud-computing/index.html, 2009.
3. M. Armbrust and et.al, "Above the clouds: A berkeley view of cloud computing," Tech. Rep., Feb 2009. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html
4. Google, "Britney spears spelling correction," Referenced online at http://www.google.com/jobs/britney.html, June 2009.
5. M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in *Proceedings of Crypto 2007, volume 4622 of LNCS*. Springer-Verlag, 2007.
6. D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. of IEEE Symposium on Security and Privacy'00*, 2000.
7. E.-J. Goh, "Secure indexes," Cryptology ePrint Archive, Report 2003/216, 2003, http://eprint.iacr.org/.
8. D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. of EUROCRYP'04*, 2004.
9. B. Waters, D. Balfanz, G. Durfee, and D. Smetters, "Building an encrypted and searchable audit log," in *Proc. of 11th Annual Network and Distributed System*, 2004.
10. Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Proc. of ACNS'05*, 2005.
11. R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proc. of ACM CCS'06*, 2006.
12. D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proc. of TCC'07*, 2007, pp. 535–554.
13. F. Bao, R. Deng, X. Ding, and Y. Yang, "Private query on encrypted data in multi-user settings," in *Proc. of ISPEC'08*, 2008.
14. X. Yang, B. Wang, and C. Li, "Cost-based variable-length-gram selection for string collections to support approximate queries efficiently," in *Proc. of ACM SIGMOD'08*, 2008.
15. C. Li, J. Lu, and Y. Lu, "Efficient merging and filtering algorithms for approximate string searches," in *Proc. of ICDE'08*, 2008.

16. A. Behm, S. Ji, C. Li, , and J. Lu, "Space-constrained gram-based indexing for efficient approximate string search," in *Proc. of ICDE'09*.
17. S. Ji, G. Li, C. Li, and J. Feng, "Efficient interactive fuzzy keyword search," in *Proc. of WWW'09*, 2009.
18. E. Shi, J. Bethencourt, T.-H. H. Chan, D. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *IEEE Symposium on Security and Privacy*, 2007.
19. J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss, and R. N. Wright, "Secure multiparty computation of approximations," in *Proc. of ICALP'01*.
20. K. N. A. Beimel, P. Carmi and E. Weinreb, "Private approximation of search problems," in *Proc. of 38th Annual ACM Symposium on the Theory of Computing*, 2006, pp. 119–128.
21. R. Ostrovsky, "Software protection and simulations on oblivious rams," Ph.D dissertation, Massachusetts Institute of Technology, 1992.
22. V. Levenshtein, "Binary codes capable of correcting spurious insertions and deletions of ones," *Problems of Information Transmission*, vol. 1, no. 1, pp. 8–17, 1965.
23. S. Chaudhuri, V. Ganti, and R. Motwani, "Robust identification of fuzzy duplicates," in *Proc. of ICDE'05*.
24. A. Arasu, V. Ganti, and R. Kaushik, "Efficient exact set-similarity joins," in *Proc. of VLDB'06*, 2006, pp. 918–929.
25. K. Chakrabarti, S. Chaudhuri, V. Ganti, and D. Xin, "An efficient filter for approximate membership checking," in *Proc. of SIGMOD'06*.
26. B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.
27. A. Fiat and M. Naor, "Broadcast encryption," in *Proc. CRYPTO'93*.