Achieving Usable and Privacy-assured Similarity Search over Outsourced Cloud Data

Cong Wang[†], Kui Ren[†], Shucheng Yu[§], and Karthik Mahendra Raje Urs[†]
[†]Department of ECE, Illinois Institute of Technology, Chicago, IL 60616, USA
[§]Department of CS, University of Arkansas at Little Rock, AR 72204, USA
Email: [†]{cong, kren, karthik}@ece.iit.edu, [§]sxyu1@ualr.edu

***Abstract*—As the data produced by individuals and enterprises that need to be stored and utilized are rapidly increasing, data owners are motivated to outsource their local complex data management systems into the cloud for its great flexibility and economic savings. However, as sensitive cloud data may have to be encrypted before outsourcing, which obsoletes the traditional data utilization service based on plaintext keyword search, how to enable privacy-assured utilization mechanisms for outsourced cloud data is thus of paramount importance. Considering the large number of on-demand data users and huge amount of outsourced data files in cloud, the problem is particularly challenging, as it is extremely difficult to meet also the practical requirements of performance, system usability, and high-level user searching experiences.**

In this paper, we investigate the problem of secure and efficient similarity search over outsourced cloud data. Similarity search is a fundamental and powerful tool widely used in plaintext information retrieval, but has not been quite explored in the encrypted data domain. Our mechanism design first exploits a suppressing technique to build storage-efficient similarity keyword set from a given document collection, with edit distance as the similarity metric. Based on that, we then build a private trie-traverse searching index, and show it correctly achieves the defined similarity search functionality with constant search time complexity. We formally prove the privacy-preserving guarantee of the proposed mechanism under rigorous security treatment. To demonstrate the generality of our mechanism and further enrich the application spectrum, we also show our new construction naturally supports fuzzy search, a previously studied notion aiming only to tolerate typos and representation inconsistencies in the user searching input. The extensive experiments on Amazon cloud platform with real data set further demonstrate the validity and practicality of the proposed mechanism.

I. Introduction

In current era of information explosion, data being produced by both individuals and enterprises that need to be stored and searched is rapidly increasing [1]. Thus, it is getting more popular than ever for data owners to outsource the complex data management systems from local machines to cloud for the great flexibility and cost savings [2]. But, to protect data privacy and combat unsolicited accesses in cloud and beyond, sensitive data has to be encrypted by data owners before outsourcing [3], which makes deployment of traditional data utilization service (plaintext keyword search) a difficult task. The trivial solution of downloading all the data and decrypting locally is clearly impractical, due to the huge amount of bandwidth cost in cloud scale systems. Moreover, aside from eliminating the local storage management, storing data into the cloud serves no purpose unless they can be easily searched and utilized. Thus, exploring privacy-assured and effective search service over encrypted cloud data is of paramount importance.

Considering the large number of on-demand data users and huge amount of outsourced data files in the cloud, this problem is particularly challenging as it is extremely difficult to meet also the requirements of practical performance and acceptable system usability. As textual information is ubiquitous in data management systems, many applications have an increasing need to support similarity keyword searches on data collections, i.e., the discovery of similar keywords with respect to a given distance measure.[1] For example, in record linkage [4], it is often required to find records from a table that are similar to a given query, or find all similar pairs of the records from two tables. In biological databases [5], it is important to retrieve all similar protein sequences with respect to a given reference so as to identify biological clusters. In information retrieval, when a user's search input (e.g., "`spielberg`") might not match those interesting entries exactly (e.g., "`spielburg`"), it is suggested [4], [5] to return all most similar matched entries to the user so that the user could find possibly interesting results. All these applications amplify the crucial importance of enabling the similarity keyword search for outsourced cloud data management systems for the high system usability and overall user search experience. Though similarity search is actively studied in plaintext domain (see [4], [5] and references therein), it remains a challenging problem for outsourced cloud data because of the inherent security and privacy obstacles.

In the literature, searchable encryption (e.g. [6]–[10], to list a few) is a helpful technique that treats encrypted data as files and allows a user to securely search over it through keywords and retrieve files of interest. However, directly deploying these techniques for secure large-scale cloud data search services would not be necessarily adequate, as they are developed as crypto primitives without considering high service-level requirements and in particular the similarity search functionality in mind (detailed explanation in Section III). On a different front, researchers from database communities have been studying how to enable private query execution over outsourced database scenarios (see [11] for a good recent survey). This problem is quite different from the usable and private keyword search problem we investigate in this paper,

[1]A keyword denotes any sequence of letters drawn from a given alphabet.

and those techniques (not necessarily encryption based) are not suitable to be extended to our situation.

Hence, in this paper, we focus on enabling usable yet privacy-preserving similarity keyword search over outsourced cloud data. Similarity search has wide applications and greatly enhances system usability as well as user searching experience, by returning the closest possible matching files based on keyword similarity. To achieve the goals on both system security and usability, our mechanism design first leverages the edit distance as similarity metric and exploits a suppressing technique to construct storage-efficient similarity keyword set from a given document selection. As direct integration of similarity keyword set and existing searchable encryption technique does not provide good enough search efficiency, we then build a private and efficient trie-traverse searching index, and show it correctly achieves the defined similarity search functionality with constant search time complexity. To demonstrate the generality of our mechanism and further enrich the application spectrum, we also show our new construction naturally supports fuzzy search, a previously studied notion aiming only to tolerate typos and representation inconsistencies in the user searching input [12]. Formal security analysis and extensive experiments on Amazon EC2 cloud platform [13] show that the proposed scheme is highly efficient and provably secure. Our contribution can be summarized as follows:

1) We formalize the problem of usable and privacy-assured similarity keyword search over outsourced cloud data, and provide a complete mechanism design which fulfills the similarity search functionality while maintaining keyword privacy.

2) Our solution is highly efficient. For each trapdoor of searched keyword, the search cost at cloud server is only a constant in the proposed symbol-based trie-traverse searching scheme. The natural support of fuzzy search further enriches the application spectrum of proposed similarity search.

3) We give formal security proofs to rigorously justify the correctness and privacy-preserving guarantee of the proposed mechanism. Extensive experiments on Amazon cloud with real data set demonstrate the efficiency of the proposed solution.

The rest of paper is organized as follows: Section II introduces the system and threat model, our design goal, preliminaries, and notations. Section III and IV provide the detailed description of our proposed mechanism designs. Section V and VI present the security analysis and the experiment results, respectively. Section VII summarizes the related work, followed by the concluding remarks in Section VIII.

II. Problem Formulation

A. System and Threat Model

We consider a high-level architecture for cloud data utilization services illustrated in Figure 1. At its core, the architecture consists of three different entities: the *data owner*, the *user*, and the *cloud server*. Here the data owner may represent either the individual or the enterprise customer, who has a collection of n data files $\mathcal{C} = (F_1, F_2, \ldots, F_n)$ to be stored in the cloud server. A predefined set of distinct keywords in $\mathcal{C}$ is denoted as $W = \{w_1, w_2, ..., w_p\}$. To keep sensitive data confidential from unauthorized entities, cryptographic methods have to be applied to the data collection $\mathcal{C}$ by the owner before outsourcing. For effective cloud data utilization such service architecture should be equipped with similarity search capability over encrypted cloud data. To initialize the service, data owner will distribute search request (trapdoor) generation keys sk to authorized users, such as partners in a collaboration team or employees in the enterprise organization. Here, we assume the authorization between the data owner and users is appropriately done. To securely search the file collection for any interested typing input w, an authorized user uses the trapdoor generation key to generate a search request via some one-way function (similar as [7], [10], [14]), and submit it to the cloud. The latter then performs the search over the data collection $\mathcal{C}$ without decryption and sends back all encrypted files containing the specific keyword w, denoted as FID_w[2]. The similarity keyword search scheme returns the closest possible results based on pre-specified similarity metrics (formally defined in Section II-C).

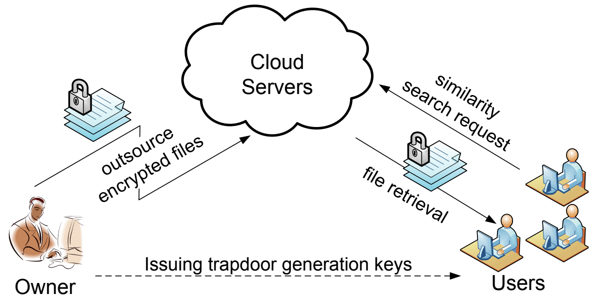

Cloud Servers
outsource encrypted files
similarity search request
file retrieval
Owner
Issuing trapdoor generation keys
Users

Fig. 1: Architecture of similarity keyword search over outsourced cloud data

Throughout this paper, we consider an "honest-but-curious" cloud server, which is consistent with most previous searchable encryption schemes. We assume the cloud server acts in an "honest" fashion and correctly follows the designated protocol specification, but is "curious" to infer and analyze the message flow received during the protocol so as to learn additional information. Note that when designing similarity keyword searching scheme, we follow the security definition deployed in the traditional searchable symmetric encryption [10]. Namely, it is required that nothing beyond the outcome and the pattern of search queries should be leaked from the remotely stored files and index (more details in Section V).

B. Design Goals

To enable secure and efficient similarity keyword search over encrypted cloud data under the aforementioned model, our mechanism should achieve the following design goals: 1) Similarity search functionality: to explore various mechanisms for designing efficient yet correct similarity search schemes over outsourced cloud data; 2) Security guarantee: to prevent cloud server from learning either the data files or the searched

[2]We do not differentiate between files and file IDs here.

keywords beyond the search pattern and access pattern; 3) Efficiency: to achieve above goals with as little as possible storage, communication, and computation overhead.

C. Notations and Preliminaries

- $\mathcal{C}$ – the file collection to be outsourced, denoted as a set of n data files $\mathcal{C} = (F_1, F_2, \ldots, F_n)$.
- W – the distinct keywords extracted from file collection $\mathcal{C}$, denoted as a set of m words $W = (w_1, w_2, ..., w_p)$.
- $\mathcal{I}$ – the index built for privacy-assured similarity search.
- T_w – the trapdoor generated by a user as a search request of input keyword w via some one-way transformation.
- $S_{w,d}$ – similarity keyword set of w, where d is the similarity threshold according to a certain similarity metrics.
- FID_{w_i} – the set of identifiers of files in $\mathcal{C}$ that contain keyword w_i.
- $f(key, \cdot), g(key, \cdot)$ – pseudorandom function (PRF), defined as: $\{0,1\}^* \times key \rightarrow \{0,1\}^l$.
- $\mathsf{Enc}(key, \cdot)$, $\mathsf{Dec}(key, \cdot)$ – symmetric key based semantic secure encryption/decryption function.

Edit Distance. Edit distance is one of the quantitative measurement of string similarity [15]. The *edit distance* $\mathsf{ed}(w_1, w_2)$ between two words w_1 and w_2 is the minimum number of primitive operations, including character insertion, deletion, and substitution, necessary to transform one of them into the other. Given a keyword w, we let $S_{w,d}$ denote its similarity set of words, such that any $w' \in S_{w,d}$ satisfies $\mathsf{ed}(w, w') \leq d$ for a certain integer d.

Similarity Keyword Search. Using edit distance, the definition of *Similarity Keyword Search* can be formulated as follows: Given n encrypted data files $\mathcal{C} = (F_1, F_2, \ldots, F_n)$, a predefined set of distinct keywords $W = \{w_1, w_2, ..., w_p\}$, a word w in the searching input and a specified edit distance d, the execution of similarity search should return a set of files $\{\mathsf{FID}_{w_i}\}$, where $\mathsf{ed}(w, w_i) \leq d$.

Searchable Encryption. It is a cryptographic primitive which allows a user to securely search over encrypted data via keyword(s) without first decrypting it. Most existing searchable encryption schemes employ the approach of building an encrypted searchable index, where its content is hidden to the server unless it is given appropriate trapdoors [1] for equality test. The trapdoors can only be generated via authorized secret key(s), and effective keyword search can be realized while both file content and keyword privacy are well-preserved.

III. The Overall Framework

To implement the secure and effective cloud data utilization service, we first explain the design challenges with regarding to the relevant building block, searchable encryption, and give a corresponding two-step framework of our proposed approach. To make the following presentation easier, we also give some details about the suppressing technique to be exploited for similarity keyword set construction. The questions on how to realize the efficient similarity search in a privacy-preserving manner and how to support a previously studied notion of fuzzy search is then described in the next section.

A. Challenges and Our Design Overview

As stated above, similarity search aims at retrieving all the similar results that are within the certain distance threshold from different user searching inputs. However, supporting this similarity matching over encrypted cloud data is quite challenging due to the inherent security and privacy obstacles: any two words that are approximately equal to each other would no longer be so after their one-way cryptographic transformation (e.g., by pseudorandom or hash functions), as required for encrypted keyword search. Thus, traditional searchable encryption schemes that conduct search only via equality test among the provided trapdoors and the encrypted searchable index do not support similarity search functionality.

To address this challenging problem, we propose to explore a two-step mechanism to circumvent the difficulty of conducting similarity match over encrypted data: 1) we build up a similarity keyword set that incorporates not only the exact keywords but also those ones that differ slightly due to format inconsistencies, minor typos, etc.; 2) based on the resulted similarity keyword set, we then design an efficient and secure similarity searching approach for effective file retrieval. In the following, we present how the secure similarity keyword search is implemented via this two-step mechanism with proposed techniques respectively.

Discussions. For the benefits of outsourcing, we believe the efficiency of considered operation is of first priority. Thus, in the following, we restrict ourselves to searchable symmetric encryption (SSE) framework [10]. As we focus on the similarity search design, we assume the pre-sharing of the trapdoor generation key between owner and users as well as adding/revoking users can be properly done via additional crypto tools such as broadcast encryption, as suggested in [10].

B. Building Similarity Keyword Sets

Building storage-efficient similarity keyword sets is our first task to achieve the similarity search defined in Section II-C. That is, given a keyword w_i and the similarity threshold d, we aim to generate $S_{w_i,d}$ such that any $w' \in S_{w_i,d}$ satisfies $\mathsf{ed}(w, w') \leq d$. The intuitive way to do so is by simply enumerating all possible words $\{w'_i\}$ satisfying the similarity criteria $\mathsf{ed}(w_i, w'_i) \leq d$. However, this approach does not work due to the impractical size of the resulting similarity keyword set. Consider the following 26 listed variants after only one substitution operation on the first character of keyword `SENSOR`: $\{$`AENSOR`, `BENSOR`, $\cdots$, `YENSOR`, `ZENSOR`$\}$. The total number of elements of such a set would be $13 \times 26 + 1$.

To address this problem, we resort to exploit a suppressing technique proposed in our preliminary work [12]. Different from the heuristic description in [12], we provide a fully formalized design in Algorithm 1 for building such a similarity keyword set. For completeness, we briefly overview the technique below through the same example of keyword `SENSOR`. The idea is to consider only the positions of the three primitive edit operations. Specifically, we use a wildcard '$\star$' to denote all three operations of character insertion, deletion and substitution at any position. For the keyword `SENSOR` with

Algorithm 1: CreateSimilaritySet(w_i, d)

```
Data: keyword $w_i$ and threshold distance $d$
Result: similarity keyword set $S_{w_i,d}$
begin
    if $d > 1$ then
1       CreateSimilaritySet($w_i$, $d-1$);
    if $d = 0$ then
2       set $S_{w_i,d} = \{w_i\}$;
    else
        for $k \leftarrow 1$ to $|S_{w_i,d-1}|$ do
            for $j \leftarrow 1$ to $2 \times |S_{w_i,d-1}[k]| + 1$ do
                if $j$ is odd then
3                   Set variant as $S_{w_i,d-1}[k]$;
4                   Insert $\star$ at position $\lfloor (j+1)/2 \rfloor$;
                else
5                   Set variant as $S_{w_i,d-1}[k]$;
6                   Replace $\lfloor j/2 \rfloor$-th character with $\star$;
                if variant is not in $S_{w_i,d-1}$ then
7                   Set $S_{w_i,d} = S_{w_i,d} \cup \{variant\}$;
```

the pre-set edit distance 1, its similarity keyword set can be constructed as $S_{\texttt{SENSOR},1} = \{$`SENSOR`, `⋆SENSOR`, `⋆ENSOR`, `S⋆ENSOR`, `S⋆NSOR`, $\cdots$, `SENSO⋆R`, `SENSO⋆`, `SENSOR⋆`$\}$. The total number of variants after one operation on `SENSOR` can now be reduced to only 13+1, far less than the exhaustive enumeration approach. Generally, the similarity keyword 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 $S_{w_i,k}$ denotes the set of words w_i' with k wildcards and $0 \leq k \leq d$.[3] The size of $S_{w_i,d}$ will be $\mathcal{O}(\ell^d)$, opposing to $\mathcal{O}(\ell^d \times 26^d)$ obtained in the straightforward approach. Thus, the larger the pre-set edit distance d is, the more storage overhead can be saved. Note that this technique, while suppressing the similarity keyword set, does not affect the search correctness, as described in the next section.

Discussions. Though our suppressing technique considerably reduces the size of similarity keyword set than the straightforward approach, the size still depends on the value of d. But in practice d does not have to be too large as it may on the other hand downgrades the search result accuracy, i.e., enforcing all data files in the collection retrieved. As shown in Section VI, when the average keyword length ℓ ranges from 4.78 to 5.6, setting $d = 2$ ensures a reasonable result in terms of both index size and search efficiency.

IV. Realizing Efficient Similarity Search

While the suppressing technique helps construct storage-efficient similarity keyword sets, it introduces yet another challenge: How to generate the search request and how to perform similarity keyword search? Answering the former is to ensure the correctness, and the latter is to ensure the efficiency and security of the similarity search over outsourced data.

A. Generating Search Request

According to Section II-C, for any w as a searching input, the execution of similarity search should return a set of files $\{\text{FID}_{w_i}\}$, where $\mathsf{ed}(w, w_i) \leq d$. But we should note that the similarity keyword set generated via aforementioned suppressing technique contains wildcard based variants in it. Therefore, for the correctness of the similarity search, the search request for input w must also contain wildcard based variants in order to perform the equality match. The **Theorem** 1 shows that as long as the search request for user's input w is w's similarity set $S_{w,d}$ generated via Algorithm 1, then the correctness of the similarity search can be achieved.

Theorem *1:* The intersection of the similarity sets $S_{w_i,d}$ and $S_{w,d}$ for keyword w_i and search input w is not empty if and only if $\mathsf{ed}(w, w_i) \leq d$.

Proof: First, we show the completeness, i.e., if $\mathsf{ed}(w, w_i) \leq d$, then $S_{w_i,d} \cap S_{w,d}$ is not empty. To prove this, it suffices to find a common element in $S_{w_i,d} \cap S_{w,d}$. Since $\mathsf{ed}(w, w_i) \leq d$, then at most we need d primitive operations to transform w into w_i (or vice versa). If we mark the positions of those d operations on w as $\star$, then this marked element, denoted as w^*, must be an element in $S_{w,d}$. Meanwhile, from w^*, we can reverse the d primitive operations on $\star$ and transform w^* back to w_i. This indicates that w^* must be an element in $S_{w_i,d}$, too. So w^* is at least the one common element in $S_{w_i,d} \cap S_{w,d}$, i.e., $S_{w_i,d} \cap S_{w,d}$ is non-empty.

Next, we prove the soundness, i.e, if $S_{w_i,d} \cap S_{w,d}$ is not empty, then $\mathsf{ed}(w, w_i) \leq d$. Again, we use w^* to denote the common element in $S_{w_i,d} \cap S_{w,d}$. If w^* does not contain any wildcard $\star$, then it must be the case where $w^* = w = w_i$. This is due to the definition of our similarity keyword set. Thus, $\mathsf{ed}(w, w_i) = 0 \leq d$. On the other hand, if w^* does contain some wildcards $\star$, then at most the number of $\star$ in w^* is d, due to $w^* \in S_{w,d} \cap S_{w_i,d}$. Note that for any $\star$ in w^*, we can always first reverse the underlying primitive operation on that $\star$ and change it back to the character in w and w_i, respectively. Denote the resulting variants as w'^* and $w_i'^*$, with both sharing $d-1$ different $\star$'s. Then we only need at most one primitive operation on that specific position to transform w'^* (or $w_i'^*$) to $w_i'^*$ (or w'^*). In other words, $\mathsf{ed}(w'^*, w_i'^*)$ is at most 1. After reverting all the d different $\star$'s from w^*, we get w and w_i accordingly. Thus, $\mathsf{ed}(w, w_i)$ is at most d. ■

The above proof showed how to ensure the similarity search to return all the correct results according to the definition. Next, we present a construction of secure and efficient searching index to achieve security and efficiency.

B. Performing Similarity Search — The Basic Scheme

For high-level performance of cloud data service system, the search time efficiency has to be taken into consideration. Based on the storage-efficient similarity keyword set, an intuitive way to realize similarity keyword search is to use the traditional listing approach. That is, we treat every wildcard variant in the

[3] $S_{w,0}$ denotes the user's original input.

similarity keyword set as a real keyword within an inverted index table as shown below. Here $f(key,\cdot), g(key,\cdot)$ denote pseudo-random functions introduced in Section II-C and τ denotes the maximum size of the similarity keyword set $S_{w_i,d}$ for $w_i \in W$, i.e., $\tau = \max\{|S_{w_i,d}|\}_{w_i \in W}$ where $|W| = p$.

In the `Preprocessing` phase:

1) The owner picks random keys x, y and builds index $\mathcal{I} = \{f(x, w'_i), \mathsf{Enc}(sk_{w'_i}, \mathrm{FID}_{w_i})\}_{w'_i \in S_{w_i,d}, 1 \le i \le p}$,[4] where secret key $sk_{w'_i} = g(y, w'_i)$.
2) He inserts extra $\tau|W| - |\mathcal{I}|$ dummy entries in $\mathcal{I}$, using random values of the same size as existing entries of $\mathcal{I}$.
3) He randomly shuffles the entries of the index $\mathcal{I}$ and then outsources $\mathcal{I}$ with encrypted file collection $\mathcal{C}$ to cloud.

In the `Searching` phase:

1) The user generates $S_{w,d}$ from his input w via Algorithm 1 and derives $T_{w'} = (f(x, w'), g(y, w'))$ for each $w' \in S_{w,d}$; he then generates $\tau - |S_{w,d}|$ dummy trapdoors, if any, from $\{f(x, j), g(y, j)\}_{1 \le j \le \tau|W| - |\mathcal{I}|}$ (by randomly choosing j) and sends a total of τ trapdoors to cloud.
2) Cloud server compares all received trapdoors $\{f(x, w')\}$ (and $\{f(x, j)\}$) with the index table $\mathcal{I}$, uses the corresponding $\{g(y, w')\}$ to decrypt the matched entries, and returns the union of file identifiers, $\{\mathrm{FID}_{w_i}\}_{\mathsf{ed}(w,w_i) \le d}$.
3) The user retrieves and decrypts the files of interest.

The result of **Theorem** 1 ensures the correctness of the above similarity search scheme. However, since the size of similarity keyword set has been expanded from $|W|$ to $\tau|W|$, it is easy to see the server side index storage and searching cost of this simple approach is $\mathcal{O}(\tau|W|)$. While such performance might be good enough for small size data collection, the linear searching cost towards the size of similarity keyword set is far from satisfactory when the system goes to cloud scale. Here the added dummy trapdoors ensure that all similarity sets have the same size τ, which further hides the searched keyword length information. For ease of presentation, we defer the detailed security analysis of this approach to Section V.

Discussions. One immediate improvement to the above basic searching scheme is to use Bloom Filters [16] to represent the index for similarity keyword set $S_{w_i,d}$ of each keyword w_i with edit distance d. Namely, the set $\{f(x, w'_i)\}_{w'_i \in S_{w_i,d}}$ is inserted into keyword w_i's Bloom filter as the index stored on the server. Now by binding the encrypted file identifiers $\mathsf{Enc}(sk_{w'_i}, \mathrm{FID}_{w_i})$ to w_i's Bloom filter, a per keyword index is generated to track the data files. Upon receiving the search request $\{T_{w'}\}_{w' \in S_{w,d}}$, the server just tests the member ship of each $f(x, w')$, $w' \in S_{w,k}$, against each Bloom filter. Compared to the listing scheme, we reduce the searching cost to $\mathcal{O}(|W|)$.

C. The Symbol-based Trie-Traverse Searching Scheme

To further enhance the search efficiency, we now propose a symbol-based trie-traverse searching scheme, where a multi-way tree is constructed for storing the similarity keyword elements $\{S_{w_i,d}\}_{w_i \in W}$ over a finite symbol set. The key idea behind this construction is that all trapdoors sharing a common prefix have a common node. 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 a trapdoor. All similar 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 $|\ \Delta\ | = 2^\theta$ and each symbol $\alpha_i \in \Delta$ is denoted by a θ-bit binary vector. Below, l is the output length of one-way function $f(key, \cdot)$.

[4]If $w'_i \in S_{w_i,d} \cap S_{w_j,d}$ for $j \ne i$, we keep a single entry of $f(x, w'_i)$ by merging FID_{w_i} and FID_{w_j}. Also, we assume certain paddings are done to keep the size of $\mathsf{Enc}(sk, \mathrm{FID}_{w_i})$ the same for $1 \le i \le p$.

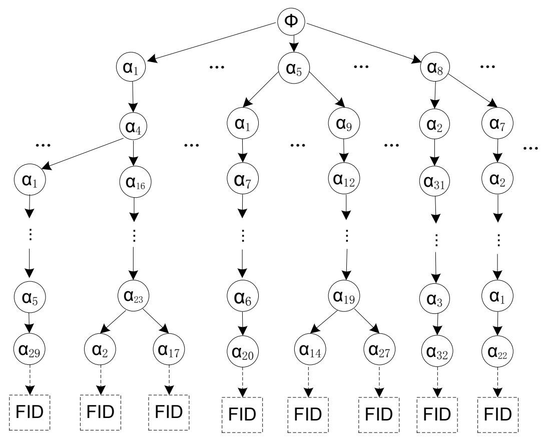


Fig. 2: An example of integrated symbol-based index for all words in the similarity keyword set.

In the `Preprocessing` phase:

1) The data owner computes $f(x, w'_i)$ for each $w'_i \in S_{w_i,d}$, $1 \le i \le p$ together with dummy entries as in the basic scheme. He then divides them into symbols as $\alpha_{i_1} \cdots \alpha_{i_{l/\theta}}$ from Δ, and builds up a trie G_W covering all the similarity keywords of $w_i \in W$;
2) He attaches $\{\mathsf{Enc}(sk_{w'_i}, \mathrm{FID}_{w_i})\}_{w'_i \in S_{w_i,d}, 1 \le i \le p}$ to G_W and outsources it with encrypted collection $\mathcal{C}$ to cloud.

In the `Searching` phase:

1) For search input w, the user sends a set of τ trapdoors as the search request, including $\{T_{w'}\}_{w' \in S_{w,d}}$ and $\tau - |S_{w,d}|$ dummy trapdoors as in the basic scheme.
2) The cloud server divides each $f(x, w')$ (or $f(x, j)$) into a sequence of symbols from Δ, performs the search over the trie G_W using Algorithm 2, decrypts matched entries via $g(y, w')$, and returns $\{\mathrm{FID}_{w_i}\}_{\mathsf{ed}(w,w_i) \le d}$ to the user.

Discussions. With the same input x and similarity keyword $w'_i \in S_{w_i,d}$, the output of function $f(key, \cdot)$ is deterministic. Thus, the division of the trapdoor into l/θ symbols $\alpha_{i_1} \cdots \alpha_{i_{l/\theta}}$ in Δ would be unique. Moreover, no information about w_i will be leaked from the output $\alpha_{i_1} \cdots \alpha_{i_{l/\theta}}$. Since the paths of trapdoors for different keywords can be integrated by merging all the paths with the same prefix into a single trie, efficient depth-first search can be finished in constant time. Specifically, given fixed l and θ, for each request, the search cost at the server side is only $\mathcal{O}(1)$ (a constant related to l/θ), which has nothing to do with the number of files n or the size of related keywords $|W|$ or $\tau|W|$. Here the encrypted file

Algorithm 2: SearchingTree($\{T'_w\}, G_w$)

```
Data: Searching Trapdoor set $\{T'_w\}$ and $G_w$
Result: Result ID set
begin
    for $i \leftarrow 1$ to $|\{T'_w\}|$ do
1       set $currentnode$ as $root$ of $G_w$;
        for $j \leftarrow 1$ to $l/\theta$ do
2           Set $\alpha$ as $\alpha_j$ in $f(x, w')$ within the $i$-th $T'_w$;
            if no child of $currentnode$ contains $\alpha$ then
3               break;
4           Set $currentnode$ as $child$ containing $\alpha$;
        if $currentnode$ is $leafnode$ then
5           Append $currentnode.FIDs$ to $resultIDset$;
            if $i = 1$ then
6               return $resultIDset$;
7   return $resultIDset$;
```

identifiers will be indexed according to its address or name and the index information will be stored at the ending node of the corresponding path. Such an example is given in Fig. 2.

D. Supporting Fuzzy Search

Fuzzy keyword search [12] is a recently proposed notion aiming to tolerate typos in the user searching input over encrypted cloud data. By following the formulation of similarity search problem in Section II-C, we reiterate fuzzy search problem as follows: given a collection $\mathcal{C}$ of encrypted files, a predefined set W of distinct keywords, a search input w, and a specified edit distance d, the execution of fuzzy search should: 1) return $\{\text{FID}_{w_i}\}$ if $w = w_i \in W$; 2) or return $\{\text{FID}_{w_i}\}$, where $\mathsf{ed}(w, w_i) \leq d$, if $w \notin W$.

Given the definitions of fuzzy search and similarity search, we can see that the notion of fuzzy search naturally satisfies similarity search, though the purposes of the two are not the same. Recall that in the fuzzy search mechanism, the trapdoor of user's original input (i.e., $S_{w,0}$) is always checked first by cloud server for exact match. If exact match exists, all the trapdoors for the wildcard variants of search input w will be discarded, and only the files for exact matches are returned. Based on this understanding, to support fuzzy search using the aforementioned similarity search mechanism, we can instruct the cloud server to always search the first trapdoor of all the received trapdoors for the search input and return exact match if possible. If not, then every match within the searchable index (including the exact matches) is to be returned.

V. Security Analysis

We now prove the security guarantee stated in Section II-B. Similar to existing SSE scheme [10], our searching mechanism always returns the same search results for the same search requests. Specifically, the cloud server can build up access patterns (from observation on retrieved encrypted data files) and search patterns (from observation on one-way transformed search requests) along the interactions with users, even if it cannot see the underlying plaintext. Thus, the security guarantee should ensure that nothing beyond the pattern and the outcome of a series of search requests be leaked. Here we adapt the simulation-based security model of [10] and prove our similarity search meets the non-adaptive semantic security guarantee. The non-adaptive attack model only considers adversaries (i.e., the cloud server) that cannot choose search requests based on the trapdoors and search outcomes of previous searches [10]. Note that this is acceptable to us since only users with authorized secret keys can generate search trapdoors. We leave the analysis under the stronger adaptive attack model as our future work. Next, we introduce some auxiliary notions used in [10] and adapt them for our private similarity keyword search scheme design.

History: an interaction between the user and the cloud server, determined by a file collection $\mathcal{C}$ and a set of keywords searched by the user, denoted as $H_q = (\mathcal{C}, w^1, \ldots, w^q)$.

View: given a history H_q under some secret key K, the cloud server can only see an encrypted version of the history, i.e., the view $V_K(H_q)$, including: the index $\mathcal{I}$ of $\mathcal{C}$; the trapdoors of the queried keywords $\{T_{w'}\}_{w' \in \{S_{w^1,d}, \cdots, S_{w^q,d}\}}$; and the encrypted file collection of $\mathcal{C}$, denoted as $\{e_1, \ldots, e_n\}$.

Trace: given a history H_q and an encrypted file collection $\mathcal{C}$, the trace of $Tr(H_q)$ captures the precise information to be learned by cloud server, including: the size of the encrypted files $(|F_1|, \ldots, |F_n|)$; the outcome of each search, $\{\text{FID}_{w_i}\}_{\mathsf{ed}(w_i, w^j) \leq d}$ for $1 \leq j \leq q$; and the pattern Π_q for each search. Here Π_q is a symmetric matrix where the entry $\Pi_q[i,j]$ stores the intersection $\{T_{w'}\}_{w' \in S_{w^i,d}} \cap \{T_{w'}\}_{w' \in S_{w^j,d}}$.

Informally, our security strength is captured by the statement that given two histories with the identical trace, the cloud server is not able to distinguish the views of the two histories. In other words, the cloud server cannot extract additional knowledge beyond the information we are willing to leak (i.e., the trace) and thus our mechanism is secure. The security result for our similarity search scheme is stated in the **Theorem** 2. Since we have shown in Section IV-D that the similarity search naturally supports fuzzy search, the following result applies to the instantiated fuzzy search as well.

Theorem *2:* Our similarity keyword search schemes meet the non-adaptive semantic security.

Proof: Due to space limitation, we only give the proof for the basic approach in Section IV-B. The proof of other schemes follow similarly. To prove the semantic security, we describe a simulator $\mathcal{S}$ such that, given $Tr(H_q)$, it can simulate a view V_q^* indistinguishable from cloud server's view $V_K(H_q)$ with probability negligibly close to 1, for any $q \in \mathbb{N}$, any H_q and randomly chosen K. Here the security parameter l of the PRF $f(key, \cdot)$, auxiliary information $\tau = \max\{|S_{w_i,d}|\}_{w_i \in W}$, $|W|$, and size of padded FID_{w_i} are known to $\mathcal{S}$.

For $q = 0$, $\mathcal{S}$ builds $V_0^* = \{\ e_1^*, e_2^*, \cdots, e_n^*, \mathcal{I}^*\}$ such that e_i^* is randomly chosen from $\{0,1\}^{|F_i|}$ for $1 \leq i \leq n$. Let $\mathcal{I}^* = (\mathsf{T}^*, \mathsf{C}^*)$, and we use $\mathsf{T}^*[i]$ and $\mathsf{C}^*[i]$ to denote the i-th row entry in T^* and C^* respectively, as generated below:

- To generate T^*, for $1 \le i \le \tau|W|$, $\mathcal{S}$ selects a random $t_i^* \in \{0,1\}^l$, and sets $\mathsf{T}^*[i] = t_i^*$.
- To generate C^*, for $1 \le i \le \tau|W|$, $\mathcal{S}$ selects a random $c_i^* \in \{0,1\}^{|\mathrm{FID}_{w_i}|}$ and sets $\mathsf{C}^*[i] = c_i^*$;

Due to the semantic security of the symmetric encryption, no probabilistic polynomial-time (P.P.T.) adversary can distinguish between e_i and e_i^*, or between $\mathsf{Enc}(sk_{w_i'}, \mathrm{FID}_{w_i})$ and c_i^*. Also due to the pseudo-randomness of the trapdoor function, no P.P.T. adversary can distinguish between $f(x, w_i')$ and a random string t_i^*. Thus, $V_K(H_0)$ and V_0^* are indistinguishable.

For $q \ge 1$, $\mathcal{S}$ builds the set $V_q^* = \{\ e_1^*,\ e_2^*,\ \cdots, e_n^*,\ \mathcal{I}^*,\ \{T_{w'}^*\}_{w' \in \{S_{w^1,d}, \cdots, S_{w^q,d}\}}\}$ such that e_i^* is still randomly drawn from $\{0,1\}^{|F_i|}$ for $1 \le i \le n$. Let $\mathcal{I}^* = (\mathsf{T}^*, \mathsf{C}^*)$. Note that the result $\{\mathrm{FID}_{w_i}\}_{\mathsf{ed}(w_i, w^j) \le d}$ for the j-th search request is the union of file identifiers in the matched entries of the index, we can rewrite it as $\cup_{k=1}^{\alpha_j} \mathrm{FID}_{w_{j,k}}$, where each $w_{j,k}$ meets $\mathsf{ed}(w_{j,k}, w^j) \le d$ and α_j denotes the number of matches for the j-th search. To build the first trapdoor set $\{T_{w'}^*\}_{w' \in \{S_{w^1,d}\}}$ for search input w^1, the simulator $\mathcal{S}$ does the following:

- Select α_1 random strings $t_{1,1}^*, \ldots, t_{1,\alpha_1}^* \in \{0,1\}^l$ and set them to α_1 non-assigned entries $\mathsf{T}^*[i_{1,1}], \ldots, \mathsf{T}^*[i_{1,\alpha_1}]$.
- Select α_1 random strings $\rho_{1,1}^*, \ldots, \rho_{1,\alpha_1}^* \in \{0,1\}^l$ and set $\mathsf{C}^*[i_{1,k}] = \mathsf{Enc}(\rho_{1,k}^*, \mathrm{FID}_{w_{i,k}})$ for $1 \le k \le \alpha_1$.
- Set remaining $\tau - \alpha_1$ trapdoors as random value pairs $(t_{1,k}^*, \rho_{1,k}^*) \in \{0,1\}^l \times \{0,1\}^l$ for $\alpha_1 < k \le \tau$.

For trapdoor simulation of w^j for $2 \le j \le q$, if $|\Pi_q[i,j]| = 0$ for all $i < j$, the simulator repeats the same process as simulating trapdoors for w^1. Otherwise, let β_j denotes the number of file identifier list in $\{\mathrm{FID}_{w_{j,k}}\}_{1 \le k \le \alpha_j}$, which have been assigned already in $\{\mathrm{FID}_{w_{i,k}}\}_{1 \le k \le \alpha_i, i<j}$. Next, $\mathcal{S}$ does:

- Choose β_j trapdoors from existing $\{T_{w'}^*\}_{w' \in \{S_{w^i,d}\}, i<j}$ that match to the common β_j file identifier list of $\{\mathrm{FID}_{w_{j,k}}\}_{1 \le k \le \alpha_j} \cap (\cup_{i=1}^{j-1} \{\mathrm{FID}_{w_{i,k}}\}_{1 \le k \le \alpha_i})$, and assign them to trapdoor simulation of w^j.
- If $\alpha_j > \beta_j$, the simulator $\mathcal{S}$ builds $\alpha_j - \beta_j$ entries in $\mathcal{I}^*$ via the same process as simulating trapdoors for w^1.
- $\mathcal{S}$ further checks $\Pi_q[i,j]$ for $i < j$, finds from already generated $\{T_{w'}^*\}_{w' \in \{S_{w^i,d}\}, i<j}$ the common γ_j trapdoors that do not have matched entries in index $\mathcal{I}^*$, and assigns them to the current trapdoor simulation of w^j.
- Set remaining $\tau - \alpha_j - \gamma_j$ trapdoors as random value pairs from $\{0,1\}^l \times \{0,1\}^l$.

The correctness of the constructed view is easy to demonstrate by searching on $\mathcal{I}^*$ via $\{T_{w'}^*\}_{w' \in S_{w^i,d}}$ for each i. We claim that there is no P.P.T. adversary can distinguish between V_q^* and $V_K(H_q)$. In particular, the simulated encrypted ciphertext is indistinguishable due to the semantic security of the symmetric encryption. The indistinguishability of index and trapdoors is based on the indistinguishability of the pseudo-random function output and a random string. Thus, we have proven the theorem. ■

VI. Scheme Evaluation

Table I compares our proposed schemes with the existing SSE scheme [10]. Note that all schemes reveal search patterns and search outcomes to server. Though introducing additional cost for owner's index generation and index size, both our schemes achieves similarity search and our trie-traverse searching scheme maintains the constant search cost on cloud. To verify the comparison, we conducted a thorough experimental evaluation of the proposed techniques on real data set: Request for comments database (RFC) [17], which contains $5,731$ plaintext files with total size 277 MB. Our experiment is implemented using C programming language on both local workstation and Amazon EC2 cloud instance. The local workstation runs with an Intel Core2 Duo CPU (E6550) running at 2.33GHz, 4GB of RAM and a 7200RPM disk drive. The cloud side process is conducted on Amazon Elastic Computing Cloud (EC2) with High-Memory instance type [13]. Note that in our experiment the dominant factor affecting the performance is the number of unique keywords to be indexed, not the file collection size. For example, it is possible for a file collection of 10 MB to have 4,000 unique key words and another collection of 1.2 MB to have more than 8,000 of words. Therefore, we present our result against different number of distinct keywords instead of file collection sizes. In particular, we first choose different file collections from the RFC data set, where the number of distinct keywords within each collection ranges from 1,000 to 10,000, and then conduct experiments over each collection multiple times to calculate the average.

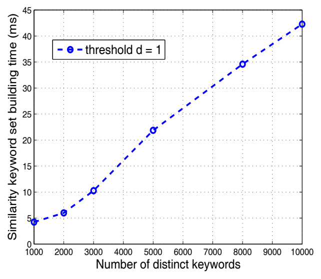
TABLE I: Comparison of SSE schemes. Here scheme-I and II denotes our basic and trie-traverse scheme respectively.

	SSE [10]	Our Scheme-I	Our Scheme-II
Preprocessing	$\mathcal{O}(\|W\|)$	$\mathcal{O}(\tau\|W\|)$	$\mathcal{O}(\tau\|W\|)$
Index size	$\mathcal{O}(\|W\|)$	$\mathcal{O}(\tau\|W\|)$	$\mathcal{O}(\tau\|W\|)$
Search cost	$\mathcal{O}(1)$	$\mathcal{O}(\tau\|W\|)$	$\mathcal{O}(1)$
Similarity search	No	Yes	Yes

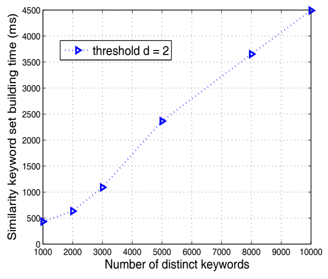
A. Preprocessing Evaluation

1) Cost For Generating Similarity Keyword Set: We first focus on the performance of similarity keyword set construction for all the keywords extracted from the document collection, which is introduced in Section III-B. Fig. 3 shows the similarity keyword set construction time with edit distance $d = 1$ and 2. We can see that in both cases, the construction time increases linearly with the number of keywords. The cost of constructing similarity keyword set under $d = 1$ is much less than the case of $d = 2$ due to the smaller set of possible wildcard-based words. Because the overall average keyword length in the RFC document collection is 4.78, we consider the case of $d = 2$ is quite sufficient for the similarity matching experiment and didn't try edit distance values larger than 2. In fact, setting larger distance threshold may eventually lead to the retrieval of all the document collection, which contradicts the goal of selectively retrieval.

2) Cost For Building Searchable Index: Given the constructed similarity keyword set, we next measure the time cost of index construction for the symbol-based trie-traverse approach. In our experiment, we choose $\theta = 4$ and use SHA-1

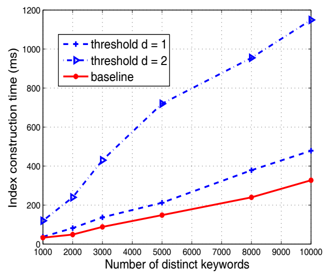


(a) $d = 1$.

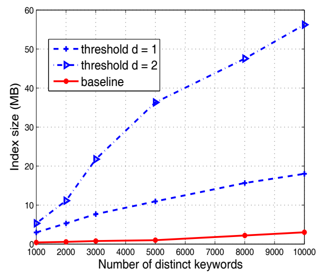


(b) $d = 2$.

Fig. 3: Similarity set construction time using wildcard-based approach with different choices of edit distance d.



(a) Index construction time.



(b) Index storage size.

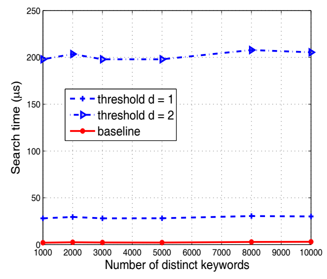
Fig. 4: Evaluation for searchable index construction with different choices of edit distance d.

as our hash function with output length of $l = 160$ bits. The resulted height of the searching tree is $l/\theta = 40$. Fig. 4-(a) shows the index construction time for edit distance $d = 1$ and $d = 2$. For completeness, we also include the index building time of existing SSE [10] as a baseline for comparison here. Similar to the similarity keyword set construction, the index construction time increases linearly with the number of distinct keywords. Compared to the baseline that only processes distinct keywords in the file collection, the index construction of our approach includes processing the similarity keywords and building the searching tree additionally. Though it takes more time than the baseline benchmark, we should note that the whole index construction is just a one-time cost and can be conducted off-line. As shown later in Section VI-B, it facilitates very fast searching performance. Fig. 4-(b) shows the index storage cost, which is the measured size of the index file stored in the raw binary format. Again, our approach consumes more storage space than the baseline due to the multi-way tree structure and the additional entries in the index corresponding to the similarity keywords, which can be deemed as the reasonable cost of supporting similarity search.

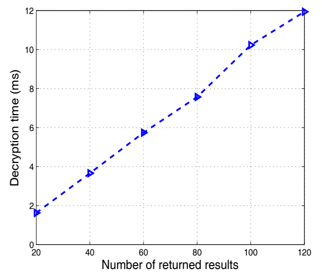
Another factor influencing the time and space cost of building searchable index is the "average keyword length" of the document collection considered, since the larger keyword length leads to more variants in the similarity keyword set to be processed (see Section III-B). As we mentioned, the average keyword length of the RFC collection is 4.78. But during the experiment, the keywords in the group of 3,000 and 5,000 are chosen in a way such that their average keyword length is 5.6. This change slightly increases the time and storage cost as seen from Fig. 4.

B. Searching Evaluation

1) Cost For Searching the Index: In our experiment, the searching operation is conducted on Amazon EC2 cloud. To test the similarity search capability, we choose to enter search keyword within edit distance of $d = 1$ and $d = 2$ to the actual existing keyword. To have a clear comparison, all the tested search keywords are of the same length, i.e., 6 characters. For example, with one test input as "KhZnYa", we attempt to retrieve through similarity search all files containing "Khanna", the surname of an author of a number of RFCs found in the collection. Fig. 5 shows the cost of similarity search for the two cases $d = 1$ and $d = 2$. With the powerful EC2 instance we choose, the search time is in the order of microseconds. Both results confirm the constant search time property of the proposed trie-traverse similarity searching mechanism. Note that the difference between the two cases is primarily due to the different number of trapdoors contained within $S_{\texttt{KhZnYa},1}$ and $S_{\texttt{KhZnYa},2}$. For completeness, we again include the search time of existing SSE [10] as a baseline here. Since essentially the baseline scheme only needs one equality match within the index after receiving the search trapdoor, it always performs faster than our mechanism. However, considering the effective support of similarity search over encrypted cloud data, our proposed mechanism can still be deemed quite efficient.



(a) Cloud side search time.



(b) User side decryption cost.

Fig. 5: Evaluation for search cost and file retrieval.

2) Cost For Results Retrieval and Decryption: The search returns encrypted results set, which has to be decrypted by the user who performed the search. This operation requires the least processing capabilities, and doesn't depend on index size or document collection. It is plainly determined by the number of retrieved results. For completeness, we show in Figure 5 the time cost to decrypt the results, with result counts varying between 20 to 120 in steps of 20. The linear dependency on the number of retrieved results is clearly verified.

VII. Related Work

Searchable encryption (e.g., [6]–[10], [14], [18] and references therein) has been widely studied as a cryptographic primitive, with a focus on security definition formalizations and efficiency improvements. In the symmetric key setting, Song et al. [6] first introduced the notion of searchable encryption. They proposed to encrypt each word in the file independently, thus with the searching cost comparable to scanning the whole file collection. Later, Goh [7] and Chang et al. [9] both developed per-file searchable index schemes, which reduce the searching cost proportional to the number of files in the collection. To further enhance search efficiency, Curtmola et al. [10] proposed a per-keyword approach, where an encrypted hash table index is built for the entire file collection, with each entry consisting of the trapdoor of a keyword and an set of correspondingly encrypted file identifiers. In the public-key setting, Boneh et al. [8] gave the first searchable encryption construction, where anyone with the public key can write to the data stored on the server but only users with the private key can search. As an attempt to enrich search predicates, conjunctive keyword search [19], [20], subset query [18] and range query over encrypted data [18], [21], have also been proposed in the public-key setting. Very recently, secure ranked search over outsourced cloud data, which further facilitates search result relevance ranking, has been studied in the settings of single keyword [22] and multi-keyword search [23], respectively. Note that all these schemes support only exact keyword search, and can not be applied to similarity search scenario.

Our earlier work [12] argues for the importance of fuzzy search over encrypted data, and gives an initial attempt on the idea of similarity keyword set construction. In this paper, we have taken several significant steps further along the direction. Firstly, we focus on a more general problem of similarity search, which naturally supports fuzzy search as a special case. Secondly, we provide a complete solution to realize the efficient similarity keyword search mechanism, including detailed algorithms for both similarity keyword set construction and efficient trie-traverse searching. Thirdly, we conduct thorough correctness and security analysis, and validate the practicality with large data set on real cloud platform.

VIII. Conclusion

In this paper, motivated by achieving practical system usability and high-level user searching experience, we study the problem on secure and efficient similarity search over outsourced cloud data. With edit distance as the similarity metric, our mechanism design first exploits a suppressing technique to build storage-efficient similarity keyword set from a given document collection. Using that keyword set as a basis, we then propose a new symbol-based trie-traverse searching mechanism, and show it correctly achieves the defined similarity search with constant search time complexity. We demonstrate the generality of our construction by showing its natural support of fuzzy search, a previously studied notion aiming only to tolerate typos and representation inconsistencies in the user searching input. We formally prove the privacy-preserving guarantee and the correctness of the proposed mechanism under rigorous security treatment. The extensive experiments on Amazon cloud platform with real data set further demonstrate the validity and practicality of the proposed mechanism.

As our ongoing work, we will continue to research on usable and secure mechanisms for the effective utilization over outsourced cloud data. Interesting problems may include designs that further take into consideration conjunction of keywords, sequence of keywords, and even the complex natural language semantics to produce highly relevant search results.

Acknowledgement

This work is supported in part by the US National Science Foundation under grants CNS-1054317 and CNS-1116939, and by an Amazon Web service research grant.

References

[1] S. Kamara and K. Lauter, "Cryptographic cloud storage," in *Proc. of Financial Cryptography Workshops*, Jan. 2010.
[2] M. Armbrust et al., "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, April 2010.
[3] Cloud Security Alliance, "Security guidance for critical areas of focus in cloud computing," http://www.cloudsecurityalliance.org, Dec. 2009.
[4] C. Li, B. Wang, and X. Yang, "Vgram: Improving performance of approximate queries on string collections using variable-length grams," in *Proc. of VLDB*, 2007, pp. 303–314.
[5] Z. Zhang, M. Hadjieleftheriou, B. C. Ooi, and D. Srivastava, "Bed-tree: an all-purpose index structure for string similarity search based on edit distance," in *Proc. of SIGMOD*, 2010, pp. 915–926.
[6] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. of IEEE Symposium on Security and Privacy*, 2000, pp. 44–55.
[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 EUROCRYPT*, 2004.
[9] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Proc. of ACNS*, 2005.
[10] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proc. of ACM CCS*, 2006, pp. 79–88.
[11] P. Samarati and S. De Capitani di Vimercati, "Data protection in outsourcing scenarios:issues and directions," in *Proc. of ASIACCS*, 2010.
[12] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *Proc. of IEEE INFOCOM, Mini-Conference*, 2010, pp. 441–445.
[13] Amazon.com, "Amazon elastic compute cloud," http://aws.amazon.com/ec2/, 2011.
[14] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in *Proc. of CRYPTO*, 2007, pp. 535–552.
[15] 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.
[16] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.
[17] RFC, "Request For Comments Database," http://www.ietf.org/rfc.html.
[18] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proc. of TCC*, 2007, pp. 535–554.
[19] P. Golle, J. Staddon, and B. R. Waters, "Secure Conjunctive Keyword Search over Encrypted Data," in *Proc. of ACNS*, 2004, pp. 31–45.
[20] L. Ballard, S. Kamara, and F. Monrose, "Achieving efficient conjunctive keyword searches over encrypted data," in *Proc. of ICICS*, 2005.
[21] E. Shi, J. Bethencourt, T.-H. H. Chan, D. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *Proc. of IEEE Symposium on Security and Privacy*, 2007, pp. 350–364.
[22] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," in *Proc. of ICDCS*, 2010.
[23] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," in *Proc. of INFOCOM*, 2011, pp. 829–837.