This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2765323, IEEE Transactions on Services Computing

EliMFS: Achieving Efficient, Leakage-resilient, and Multi-keyword Fuzzy Search on Encrypted Cloud Data

Jing Chen, Kun He, Lan Deng, Quan Yuan, Ruiying Du, Yang Xiang, and Jie Wu

***Abstract*—Motivated by privacy preservation requirements for outsourced data, keyword searches over encrypted cloud data have become a hot topic. Compared to single-keyword exact searches, multi-keyword fuzzy search schemes attract more attention because of their improvements in search accuracy, typo tolerance, and user experience in general. However, existing multi-keyword fuzzy search solutions are not sufficiently efficient when the file set in the cloud is large. To address this, we propose an Efficient Leakage-resilient Multi-keyword Fuzzy Search (EliMFS) framework over encrypted cloud data. In this framework, a novel two-stage index structure is exploited to ensure that search time is independent of file set size. The multi-keyword fuzzy search function is achieved through a delicate design based on the Gram Counting Order, the Bloom filter, and the Locality-Sensitive Hashing. Furthermore, considering the leakages caused by the two-stage index structure, we propose two specific schemes to resist these potential attacks in different threat models. Extensive analysis and experiments show that our schemes are highly efficient and leakage-resilient.**

***Index Terms*—Cloud security, searchable encryption, multi-keyword fuzzy search.**

I. Introduction

OUTSOURCING data to the cloud has become a prevalent trend in recent years because of the benefits it brings to data owners including convenient access, decreased costs, and flexible data management. With the impetus of IT giants like Amazon, Google, and Microsoft, more enterprises and individuals incline to upload their data to cloud storage [1]. However, data leakages occur frequently, causing huge losses to both cloud providers and cloud users. The lack of data security and privacy protection has become the biggest obstacle for the development of cloud storage technologies. To protect owners' data privacy, all data, especially the sensitive information, must be encrypted before outsourcing, which unfortunately makes traditional plaintext-search-based data utilization service a challenging task.

Searchable Symmetric Encryption (SSE) is a useful cryptographic primitive for searching encrypted data using particular keywords without leaking private information to cloud servers [2]. To support ciphertext searches on encrypted cloud data, researchers have proposed a number of SSE schemes. Considering the huge amount of outsourced documents in the cloud, one important direction is a *multi-keyword* search [3]–[5], which supports multiple (conjunctive or disjunctive) keywords search, e.g., "cloud" and "security", at one time. Compared to a *single-word* search [6], [7], a multi-keyword search can efficiently improve search result accuracy since single-keyword searches usually return results that are far too coarse. Note that most existing multi-keyword search techniques are less friendly towards keyword spelling since they *only* support exact keyword matching [8], [9]. Considering that users may occasionally perform searches with typos in their search keywords, *fuzzy search* is proposed to achieve approximate keyword matches to a certain extent. However, they usually exploit edited distances to quantify keywords' similarities for single-word searches *only*.

To enhance the user-searching experience and protect data privacy, achieving efficient, leakage-resilient, and multi-keyword fuzzy search services on encrypted cloud data becomes a significant challenge, especially with the increasing number of on-demand data users and the exploding quantities of outsourced cloud data. In [10], the authors implemented a search scheme that satisfies both multi-keyword and fuzzy requirements simultaneously by treating several phrases in a predefined dictionary as one keyword for a search; this approach lacks flexibility in reality. [11] tackled the privacy-preserving multi-keyword fuzzy search problem using a *forward index* structure. However, forward index based schemes need to store a list of keywords for each file and must check every file in the file set when searching [12]; this is unaffordable when a file set is large.

In this paper, we propose an Efficient Leakage-resilient Multi-keyword Fuzzy Search (EliMFS) framework for encrypted cloud data. Unlike existing multi-keyword fuzzy search schemes, we design a novel two-stage index structure, consisting of a *forward index* and an *inverted index* to improve search efficiency. Different from a forward index, the inverted index stores a list of file identifiers for each keyword, and a user can directly find the files for each keyword in a query [13]. By building up the two-stage index and designing the corresponding search algorithm, our search complexity is only determined by the number of the files associated with one keyword in the query rather than the entire file-set size. Furthermore, note that *there is a trade-off in the complex index structure which leads to a greater search*

J. Chen, K. He, and L. Deng are with the State Key Laboratory of Software Engineering, Computer School, Wuhan University, China, 430072. E-mail: chenjing, kunhe, landeng@whu.edu.cn

R. Du is with the Collaborative Innovation Center of Geospatial Technology Wuhan University. E-mail: duraying@whu.edu.cn

Q. Yuan is with the Computer School, University of Texas-Permian Basin, TX, USA. E-mail: yuan_q@utpb.edu

Y. Xiang is with the School of Information Technology, Deakin University, Australia. E-mail: yang.xiang@deakin.edu.au

J. Wu is with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA, USA. E-mail: jiewu@temple.edu

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2765323, IEEE Transactions on Services Computing

efficiency, but also causes more leakages. We aim to design a framework to achieve the desired efficiency without hurting the system privacy. Therefore, we analyze the possible leakages of our framework in different threat models and design two cryptographic schemes to balance the trade-offs among search time, storage cost, and privacy for different security levels. The multi-keyword fuzzy search function is implemented by the flexible combination of some widely-used tools, including bigram vector representation, locality-sensitive hashing (LSH) [14], and the Bloom filter [15]. Our contributions can be summarized as follows:

1) To the best of our knowledge, this is the first multi-keyword fuzzy search scheme for encrypted cloud data whose search complexity is independent of the file set size;
2) Compared to existing multi-keyword fuzzy search solutions [11], [16], we design a novel two-stage index structure and corresponding search algorithms to improve the search efficiency on encrypted cloud data;
3) To protect data privacy, we analyze the possible leakages of our framework in different threat models and we propose two specific schemes to achieve leakage-resiliency;
4) A comprehensive analysis of the privacy and efficiency is given, and experiments on a real-world dataset further show that our design has a low overhead in the search process.

The rest of the paper is organized as follows. Section II presents the formulation of our problem and the preliminaries. Sections III and IV provide detailed descriptions of our EliMFS framework and leakage-resilient schemes, respectively. Sections V and VI present theoretical and experimental analyses, respectively. Section VII summarizes the related work, and the conclusion appears in Section VIII.

II. Problem Statement

Before describing our schemes, we state the efficient multi-keyword fuzzy search problem in the outsourcing storage setting and introduce some preliminaries.

A. System Model

Our system consists of three entities: a cloud server, a data owner, and a group of authorized users, as shown in Fig. 1. Initially, the data owner, who holds a file set, creates a secret key, constructs a customized secure index for this file set, and uploads the secure index along with the encrypted file set to the cloud server. When the data owner needs to search his files with keywords, $w_1, w_2, \ldots, w_n$, he generates a token based on the queried keywords and his secret key. After the token is sent to the cloud server, the cloud server executes a search with the token and the secure index, and then returns the matched file identifiers to finish the search process. With the search results, the data owner can download the corresponding encrypted files from the cloud server and decrypt them with his secret key. As the gray arrows showed, other authorized users attempting to search the data owner's file set can obtain the tokens and file decryption keys shared with them, according to access control strategies.

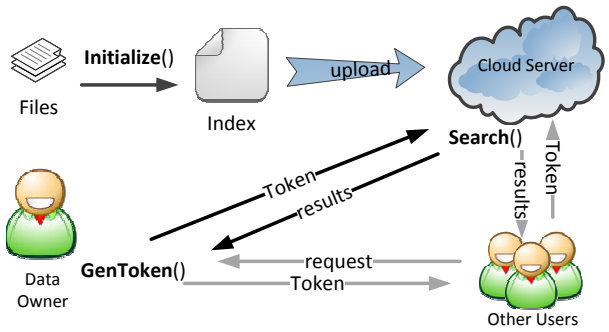


Fig. 1. Structure of encrypted cloud storage system

B. Threat Model & Design Goals

We assume that both the data owner and the authorized users are trusted, but that the cloud server is honest-but-curious. That means the cloud server follows the designed protocols honestly, but it may collect, analyze, and leak the information of its customers [17]. Regarding privacy requirements, two threat models with different adversary capabilities are considered in this paper, as in other related works [3], [11].

- *Known Ciphertext Model*: The cloud server can only observe the encrypted file set, the encrypted index, the submitted tokens, and the search results. It has no prior knowledge of the file set or the search keywords.
- *Known Background Model*: In addition to the knowledge in the Known Ciphertext Model, the cloud server is also able to obtain some additional background information like the statistical information about the files, the queried keywords and their corresponding tokens, and the frequency of searched keywords.

Considering the above threat models, our scheme is designed to achieve the following privacy and performance goals.

- *Multi-keyword Fuzzy Search*: One scheme should support multi-keyword fuzzy searches over outsourced encrypted files. For instance, a search based on a spelling mistake "clod security" can still return the "cloud security" relevant files.
- *Efficiency*: Considering that large file sets may exist in the cloud, our scheme aims to be efficient and practical in terms of *file-set-size independence*. That is, the search time should be sublinear with respect to the file set size.
- *Leakage-resilience*: The cloud server should neither infer private information nor leverage the intermediate results of searches in the targeted threat models.

C. Security Definitions

We recall the semantic security definitions from [11] and [13], and define the notions below:

- *History*: A *History* is the information which reflects the interaction between the data owner and the cloud server. It consists of the file set $\mathcal{D}$, the keyword set $\mathcal{W}$, and the query set $\mathcal{Q} = \{q_1, q_2, \ldots\}$ that is being searched.
- *View*: It is what the cloud server can actually see. The *View* of a *History* consists of the encrypted file set $\mathcal{ED}$, the encrypted index $\mathcal{I}$, and the tokens $\mathrm{Token}_1, \ldots, \mathrm{Token}_{|\mathcal{Q}|}$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2765323, IEEE Transactions on Services Computing

- *Trace*: The *Trace* of a *History* is defined as the information that can be revealed to the cloud server, such as the file set size and the search results. The *Trace* of our schemes will be listed in detail in Section IV.

Definition 1. *For two* Views *with the same* Trace *where one view is generated from a* History *selected by the cloud server and the other is generated by a simulator, an EliMFS scheme is semantically secure under the known ciphertext model, if no probabilistic polynomial time (PPT) adversary is able to distinguish them with a probability non-negligibly larger than* $1/2$.

Definition 2. *Given a collection of keyword-token pairs and two* Views *with the same* Trace *where one view is generated from a* History *selected by the cloud server and the other is generated by a simulator, an EliMFS scheme is semantically secure under the known background model, if no PPT adversary is able to distinguish them with a probability non-negligibly larger than* $1/2$.

D. Preliminaries

1) Gram Counting Order: An n-gram is a contiguous sequence of n items from a given sequence of characters [18]. For example, given a keyword "cloud", the 1-gram sequence is $\{c, l, o, u, d\}$, and the 2-gram sequence is $\{cl, lo, ou, ud\}$. In our structure, we choose a 2-gram sequence to parse a keyword, and record it in an array of size 26×26. Each element in the array represents one possible bigram. If a bigram exists in the 2-gram sequence, the corresponding element is set to be 1; otherwise it remains 0. In this way, all the keywords are transformed into bit arrays that can be input parameters of Locality-Sensitive Hashing functions.

2) Locality-Sensitive Hashing: Locality-Sensitive Hashing (LSH) has the capacity to map similar items to the same bucket with a high probability. In this paper, all keywords are processed by LSH before being attached to indexes or tokens, so that the fuzzy search can be achieved. A hash function family $\mathcal{F}$ is (R_1, R_2, P_1, P_2)-sensitive, if for any inputs of p, q, all hash functions, $h \in \mathcal{F}$, satisfy the following conditions:

- if $d(p, q) \leq R_1$, then $h(p) = h(q)$ with a probability of at least P_1;
- if $d(p, q) \geq R_2$, then $h(p) = h(q)$ with a probability of at most P_2;

where $d(p, q)$ denotes some kind of distance (e.g. Euclidean distance or Hamming distance) between p and q, R_1 and R_2 are the distance thresholds, and P_1 and P_2 represent the probability thresholds. There are multiple options for implementing LSH, such as LSHs based on bit sampling and P-stable distribution. To recognize similar keywords, we employ a P-stable distribution LSH [19] that maps a d-dimensional vector to an integer. The hash functions of a P-stable hash family follow $h_{\vec{a},b}(\vec{v}) = \lfloor \frac{\vec{a}\cdot\vec{v}+b}{c} \rfloor$, where $\vec{a}$ is a d-dimensional vector and each dimension is chosen independently from a P-stable distribution. b is a random real number chosen from $[0, c]$. The P-stable distribution LSH first projects $\vec{v}$ onto $\vec{a}$, then it utilizes c to separate them. A different $\vec{a}$ and b result in different hash functions in the family.

3) Bloom Filter: A Bloom filter [20], using a bit array of m bits to characterize a set $\mathcal{S}$, verifies whether an element is a member of a set. Each element in $\mathcal{S}$ is denoted by k bits randomly chosen from the m array positions. The Bloom filter Bf_s of the set $\mathcal{S}$ is structured as follows:

a. Set Bf_s to be an m bit length array of all zeros and choose a hash family $\mathcal{H} = \{h_i \mid h_i : a \to [1, m], a \in \mathcal{S}, i \in [1, k]\}$.
b. For each $q \in \mathcal{S}$, set the positions $h_i(q)$ ($i \in [1, k]$) in Bf_s to 1.

To check several elements, we generate a bit array in the same way. We then compute the inner product of this bit array and Bf_s. If the result equals nk, it means that these elements belong to $\mathcal{S}$, where n is the number of elements to be checked. In our scheme, every file has its own Bloom filter that corresponds to the multiple contained keywords.

4) Secure kNN Computation: Secure kNN (k-nearest neighbour) computation, proposed by Wong [21], is designed to compute the distance between two encrypted database records. In a secure kNN computation, all database records are extended to m-dimension vectors and encrypted by a vector S of m bits and two $m \times m$ invertible matrices $\{M_1, M_2\}$. The algorithms are introduced in our schemes in Section IV. More details of secure kNN computation are referred to in [21].

III. EliMFS Framework

In this section, we present the definition and main phases of our EliMFS framework and build a two-stage index structure to improve the search efficiency.

A. Definition

We first give the definitions of the two-stage index and the EliMFS scheme. The main notations used in this paper are summarized in TABLE I. An example of the two-stage index structure is shown in Fig. 2.

Definition 3 (EliMFS)**.** *An Efficient Leakage-resilient Multi-keyword Fuzzy SSE search scheme is based on the two-stage index structure and consists of three polynomial-time algorithms as follows:*

- $(SK, \text{Index}) \leftarrow \mathbf{Initialize}(\lambda, \mathcal{D}, \mathcal{W})$*: This algorithm takes as inputs the secure parameters* λ*, the file set* $\mathcal{D}$*, and the corresponding keyword set* $\mathcal{W}$*. It generates the secret key* SK *and builds up the secure index* $\text{Index} = (\mathcal{I}_1, \mathcal{I}_2)$*. Here, the first-stage index,* $\mathcal{I}_1$*, is an inverted index in which each tag corresponds to a list of identifiers. The second-stage index,* $\mathcal{I}_2$*, is a forward index in which each file identifier corresponds to the keywords it contains;*
- $\text{Token} \leftarrow \mathbf{GenToken}(SK, \vec{w})$*: Given secret key* SK *and the keyword list* $\vec{w} = \{w_1, w_2, \ldots, w_n\}$*, the data owner computes a* Token *as a certificate to search* $\vec{w}$ *in* $\mathcal{D}$*;*
- $\vec{id} \leftarrow \mathbf{Search}(\text{Index}, \text{Token})$*: The cloud server inputs the secure* Index *and the* Token *received from the requested user and outputs the file identifier list,* $\vec{id}$*, that matches the search keywords in the* Token.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2765323, IEEE Transactions on Services Computing

TABLE I
NOTATIONS

Notation	Meaning
$\mathcal{D}$	the file set of a data owner
$\mathcal{W}$	the keyword set of $\mathcal{D}$
$\mathcal{D}(w)$	file identifiers matching the keyword w
$q \xleftarrow{\$} \mathcal{S}$	sampling an element q randomly from a set $\mathcal{S}$
$x \leftarrow \mathbf{F}$	an algorithm $\mathbf{F}$ with output x
$[1, n], n \in \mathbb{N}$	an integer set $\{1, 2, \ldots, n\}$
$\mathcal{I}_1, \mathcal{I}_2$	the first- and second- stage index
λ	a secure parameter
SK	the secret key of the data owner
$w, \vec{w}$	a keyword, and a keyword set
$id, \vec{id}$	a file identifier, and a file identifier set
ϑ	the intermediate value of a keyword
$(\mathbf{Enc}, \mathbf{Dec})$	a symmetric encryption algorithm
$\mathbf{F}()$	a pseudo-random function
Bf	a Bloom filter
$\mathcal{Q}$	a set of search queries
m	the length of Bloom filter
$\mathcal{F} = \{f_1, \ldots, f_l\}$	a LSH family with l hash functions
$\mathcal{H} = \{h_1, \ldots, h_k\}$	the k hash functions of Bloom filter
tag	the entrance of a keyword in $\mathcal{I}_1$
e	an encrypted file identifier
thr	the threshold attached in a token
ip	the inner product of two Bloom filters

B. Main Phases

In this section, we introduce the three phases of the EliMFS framework: *Initialization*, *Token Generation*, and *Search*, with a search example of the keywords "cloud" and "storage," as shown in Fig. 2. Two instances of the framework with different efficiency privacy trade-offs are proposed in Section IV.

1) Initialization: In this phase, the data owner has two major tasks: generating the secret key and constructing a secure index. On one hand, the secret key can be initialized by the existing cryptographic methods, e.g., DES or AES. On the other hand, to implement an efficient multi-keyword fuzzy search, we build a two-stage secure index containing various file identifiers and keywords. In the first-stage index, each keyword is transformed into a tag, which points to all the file identifiers that contain this keyword. The second index is just the opposite — it is generated by all the file identifiers with their matched keywords.

In consideration of privacy, each file identifier is encrypted by the algorithm $\mathbf{Enc}()$ when we use this framework. For the fuzzy search function, a keyword w_i is transformed into an intermediate value $\vartheta_i (i \in [1, |\mathcal{W}|])$ by sequentially computing a 26×26-length bit array of a 2-gram sequence and an LSH value (introduced in Section II-D). As illustrated in Fig. 2, in the first-stage index, the tag_i is created by ϑ_i and a pseudo-random function $\mathbf{F}(x)$. In the second-stage index, each file identifier maps to a Bloom filter [20] deduced by all the intermediate values of multiple target keywords. In this two-stage index structure, the first stage is able to efficiently execute single keyword searches and shrink the search range in multi-keyword searches. The second stage is used to estimate whether the results of the first stage indeed contain all target keywords.

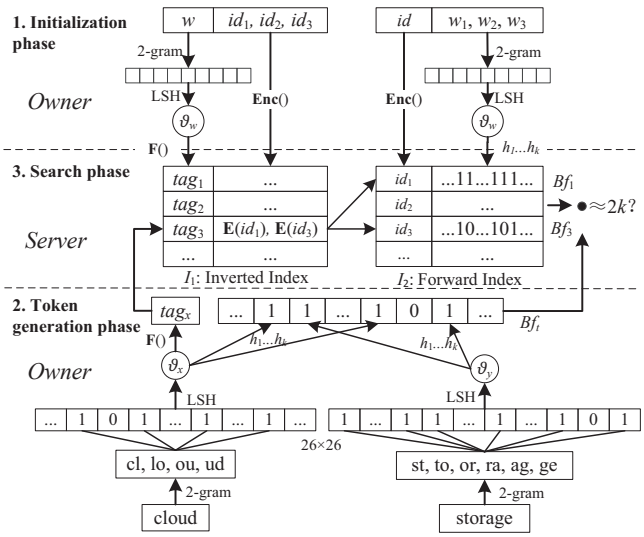


Fig. 2. The framework of EliMFS schemes

2) Token Generation: This phase is also executed on the data owner. In this phase, a token is generated for keyword searches. The token consists of two parts: a tag for searching in the first-stage index and an item for searching in the second-stage index.

The bottom of Fig. 2 shows an example of token generation for the keywords "cloud" and "storage." A token consists of a tag and a Bloom filter. To calculate these two metadata, the keywords "cloud" and "storage" should be first transferred to two intermediate values ϑ_x and ϑ_y; this is similar to the process in the initialization phase. Then, the data owner randomly chooses a keyword with a small $\mathcal{D}(w)$ and calculates the tag_x by the intermediate value and a pseudo-random function $\mathbf{F}()$. Assume the selected keyword is "cloud" in this example. Meanwhile, the Bloom filter is generated from both ϑ_x and ϑ_y by the hash functions $h_1, \ldots h_k$.

3) Search: Unlike the two aforementioned phases, the Search phase is executed on cloud servers. The keyword search process is first executed in the first-stage index. Having obtained a set of file identifiers from the first-stage search, the cloud servers can now check this relatively small file set (instead of the whole file set) using the second part of the token.

As shown in Fig. 2, after receiving the tag_x, the cloud server searches in the first-stage index to find the encrypted file identifier list (i.e. $\mathbf{E}(id_1), \mathbf{E}(id_3)$) that matches the keyword "cloud." Then, the cloud server decrypts each encrypted file identifier in the list with the secret key received from the data owner. In the second-stage index, the Bloom filter verifies whether the file also matches the other keyword "storage." As mentioned in Section II-D, the inner products $Bf_1 \cdot Bf_t$ and $Bf_3 \cdot Bf_t$ are calculated as decision fundaments, where Bf_i is the Bloom filter of id_i and Bf_t is the Bloom filter of the token. If the inner product is approximately equal to $2k$,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2765323, IEEE Transactions on Services Computing

TABLE II
INITIALIZATION STEPS IN ELIMFS-B

	Step 1:
1	$K_1, K_2 \xleftarrow{\$} \{0,1\}^{\lambda}$, $M_1, M_2 \in \mathbb{R}^{m\times m}$, $S \in \{0,1\}^m$
2	$SK = \{K_1, K_2, M_1, M_2, S\}$
	Step 2: For each $w \in \mathcal{W}$
3	$\vartheta_w \leftarrow \mathcal{F}(w)$, $tag \leftarrow \mathbf{F}(K_1, \vartheta_w)$, $K_e \leftarrow \mathbf{F}(K_2, \vartheta_w)$, $h_1(\vartheta_w), \ldots, h_k(\vartheta_w)$
4	For all $id \in \mathcal{D}(w)$: append $e \leftarrow \mathbf{Enc}(K_e, id)$ to $\mathcal{I}_1[tag]$; set $\mathcal{I}_2[id][i] = 1$, $i \in \{h_1(\vartheta_w), h_2(\vartheta_w), \ldots, h_k(\vartheta_w)\}$
	Step 3: For each $\mathcal{I}_2[id] = I$
5	Initialize $I^{'}, I^{''}$ to m-bit length vectors
6	$I^{'}[j] = I^{''}[j] = I[j]$ if $S[j] = 1$; $I^{'}[j] = \frac{1}{2}I[j] + r$, $I^{''}[j] = \frac{1}{2}I[j] - r$ if $S[j] = 0$, $j \in [1, m]$, r is a random number
7	$\mathcal{I}_2[id] = \{M_1^T \cdot I^{'}, M_2^T \cdot I^{''}\}$

TABLE III
GENTOKEN STEPS IN ELIMFS-B

	Step 1:
1	$\vartheta_{w_1} \leftarrow \mathcal{F}(w_1)$, $tag \leftarrow \mathbf{F}(K_1, \vartheta_{w_1})$, $K_e \leftarrow \mathbf{F}(K_2, \vartheta_{w_1})$
	Step 2:
2	Initialize Bf_q to be a m-bit length array of all zeros
3	For all $w_i \in \vec{w}$: get $\vartheta_{w_i} \leftarrow \mathcal{F}(w_i)$, $h_1(\vartheta_{w_i}), \ldots, h_k(\vartheta_{w_i})$, and set $Bf_q[j] = 1$, $j \in \{h_1(\vartheta_{w_i}), \ldots, h_k(\vartheta_{w_i})\}$
	Step 3:
4	Initialize $Bf_q^{'}, Bf_q^{''}$ to m-bit length vectors
5	$Bf_q^{'}[j] = Bf_q^{''}[j] = Bf_q[j]$ if $S[j] = 0$; $Bf_q^{'}[j] = \frac{1}{2}Bf_q[j] + r^{'}$, $Bf_q^{''}[j] = \frac{1}{2}Bf_q[j] - r^{'}$ if $S[j] = 1$, $j \in [1, m]$, $r^{'}$ is a random number
6	Chose $thr \leqslant k\|\vec{w}\|$, $\text{Token} = \{tag, M_1^{-1} \cdot Bf_q^{'}, M_2^{-1} \cdot Bf_q^{''}, K_e, thr\}$

the corresponding file contains both of the keywords and the identifier of this file will be attached to the search result. At last, these identifiers are sent to the data owner so that he can retrieve files from the cloud server.

IV. EFFICIENT LEAKAGE-RESILIENT SCHEMES

To resist leakages in different threat models, we instantiate two specific schemes for the EliMFS framework.

A. EliMFS-B: Basic Scheme

1) Basic Leakages: Based on the analysis of various privacy threats in cloud search, we list the basic leakages that need to be resisted:

- *Plaintext Leakage*: The content of the outsourced files and the searched keywords may be obtained by the cloud server if the schemes are not secure.
- *Search Pattern Leakage*: The search Pattern indicates whether two tokens are performing for the same query. Since most existing schemes use deterministic algorithms, *search pattern leakage* is usually not well protected. In this paper, we have a semi-protection function against *search pattern leakage*; the cloud server can only estimate whether two queries have the same keyword that is used in the first-stage search. For instance, given the tokens of queries $Q = (w_1, w_2, \ldots), Q^{'} = (w_1^{'}, w_2^{'}, \ldots)$, the cloud server is able to judge whether $w_1 = w_1^{'}$, but cannot know whether $Q = Q^{'}$.
- *Subset Leakage*: This is a special leakage introduced by the multi-keyword search since the cloud server may obtain the intermediate search results of the queried multiple keywords. For example, if a multi-keyword search is implemented by searching each keyword in query $\vec{w}$, then the cloud server can obtain the search result of any subset of $\vec{w}$. Given the token of a query $Q = (w_1, w_2, \ldots, w_n)$, the *Subset Leakage* is the result of a tampered query $Q^{'} = (w_1, \vec{w})$ in our two-stage index. The notation $\vec{w}$ refers to any nonvoid subset of $w_2 \ldots w_n$.

2) Scheme Construction: We then detail our solution to the basic leakages: EliMFS-B. For simplicity, we use w_i to represent the 2-gram sequence of keyword w_i.

The steps of algorithm $\mathbf{Initialize}(\lambda, \mathcal{D}, \mathcal{W})$ are shown in TABLE II. Given a secure parameter λ and a file set $\mathcal{D}$ with a keyword set $\mathcal{W}$, the data owner generates his secret key, SK, and the secure index, $\text{Index} = \{\mathcal{I}_1, \mathcal{I}_2\}$. Obviously, this algorithm consists of three steps. In step 1, the data owner generates his secret key, which contains two random sequences (K_1, K_2), two invertible matrices (M_1, M_2), and a vector (S). Step 2 builds the index for the file set $\mathcal{D}$. To generate an inverted index $\mathcal{I}_1$ and a forward index $\mathcal{I}_2$ simultaneously, the data owner initializes $\mathcal{I}_1$ and $\mathcal{I}_2$ to empty arrays. Then, for each keyword in $\mathcal{W}$, the data owner computes the intermediate value, the entrance in $\mathcal{I}_1$, the secret key for encrypting file identifiers, and the k hash values used in the Bloom filter. Here, $\mathcal{F} = \{f_i \mid f_i : \{0,1\}^{26\times 26} \rightarrow n, n \in \mathbb{Z}, i \in [1, l]\}$, $\mathbf{F} : \{0,1\}^{\lambda} \times \{0,1\}^{\lambda} \rightarrow \{0,1\}^{\lambda}$, and $\mathcal{H} = \{h \mid h : \{0,1\}^* \rightarrow [1, m]\}$. The data owner encrypts all the file identifiers that match w with K_e, attaches them to set $\mathcal{I}_1[tag]$, and then sets all the positions $h_1(\vartheta_w), h_2(\vartheta_w), \ldots, h_k(\vartheta_w)$ in $\mathcal{I}_2[id]$ to 1. In the last step, the data owner encrypts the Bloom filters in the second index ($\mathcal{I}_2$) using the secure kNN encryption [21] with (M_1, M_2, S). The Bloom filter I is split into two vectors $\{I^{'}, I^{''}\}$ according to S, and then $\mathcal{I}_2[id]$ is set to be $\{M_1^T \cdot I^{'}, M_2^T \cdot I^{''}\}$. Finally, the data owner gets $\text{Index} = \{\mathcal{I}_1, \mathcal{I}_2\}$ and sends the index to the cloud server with the corresponding encrypted files.

To search with keywords $\vec{w} = \{w_1, w_2, \ldots, w_n\}$, the data owner needs to generate a token as a certificate by following the steps in TABLE III. Assuming w_1 is the chosen keyword for the first-stage search, the data owner first computes the intermediate value, the entrance in $\mathcal{I}_1$, and the secret key for decrypting file identifiers. Note that if $n = 1$, the next two steps cannot be executed and tag can be sent to the cloud server directly. In step 2, a Bloom filter (Bf_q) is built from all the keywords in $\vec{w}$. Each keyword is processed with $\mathcal{F}()$ and inserted into Bf_q with k hash functions. In the third step, the Bloom filter Bf_q is encrypted with M_1, M_2, S in a manner similar to that in $\mathbf{Initialize}()$. At last, the data owner chooses a threshold (thr) whose value can be selected arbitrarily based on the preferred search accuracy. thr is used to compare the inner products during the second search stage to judge whether a file ID should be returned. $\text{Token} = \{tag, M_1^{-1} \cdot Bf_q^{'}, M_2^{-1} \cdot Bf_q^{''}, K_e, thr\}$ is the final output and is sent to the cloud server.

Upon receiving the token, the cloud server executes $\mathbf{Search}()$, as shown in TABLE IV. First, the cloud server reads all the encrypted identifiers in $\mathcal{I}_1[tag]$. If $\text{Token} = \{tag\}$, the cloud server returns these identifiers and finishes the algorithm. If $\text{Token} = \{tag, M_1^{-1} \cdot Bf_q^{'}, M_2^{-1} \cdot Bf_q^{''}, K_e, thr\}$,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2765323, IEEE Transactions on Services Computing

TABLE IV
SEARCH STEPS IN ELIMFS-B

	Step 1:
1	Retrieve $\mathcal{I}_1[tag]$
	Step 2: For each $e \in \mathcal{I}_1[tag]$
2	$id \leftarrow \mathbf{Dec}(K_e, e)$, and get $\mathcal{I}_2[id] = \{M_1^T \cdot I^{'}, M_2^T \cdot I^{''}\}$
3	$M_1^T I^{'} \cdot M_1^{-1} B f_q^{'} + M_2^T I^{''} \cdot M_2^{-1} B f_q^{''} = I^{'T} \cdot B f_q^{'} + I^{''T} \cdot B f_q^{''} = I^T \cdot B f_q$
4	If $I^T \cdot B f_q \geq thr$, attach id to $\vec{id}$

the cloud server checks the Bloom filters of $\mathcal{I}_1[tag]$ one by one in Step 2. The encrypted identifiers are decrypted by K_e, and the corresponding Bloom filters $\mathcal{I}_2[id] = \{M_1^T \cdot I^{'}, M_2^T \cdot I^{''}\}$ are retrieved. Here, the secret key K_e is known by the cloud server, but it reveals nothing except the search results of w_1. Since the search results of queried keywords are not included in the protected data, we consider this leakage to be acceptable. With $\mathcal{I}_2[id]$ and $\{M_1^{-1} \cdot B f_q^{'}, M_2^{-1} \cdot B f_q^{''}\}$, the inner product $I^T \cdot B f_q$ can be easily obtained as shown in line 3 of TABLE IV. Then, the server determines the feedback to the data owner according to the comparison results between the inner products and thr. If $I^T \cdot B f_q \geq thr$, the cloud server attaches id to the result. The results ($\vec{id}$) are returned to the data owner, and with these identifiers, the data owner can retrieve files from the cloud server as he wishes.

3) Analysis: From the perspective of practicality, both *efficiency* and *security* are important considerations.

Regarding efficiency, the search complexity of EliMFS-B is $O(|\mathcal{D}(w_1)|)$, the communication cost is constantly $O(1)$, and the storage cost on the cloud server is $O(|\mathcal{W}| + |\mathcal{D}|)$. A detailed analysis can be found in Section V. All in all, EliMFS-B performs very efficiently, although it can only protect basic leakages in the known ciphertext model. It is suitable for applications that have a high efficiency requirement and a general privacy requirement, meaning that EliMFS-B can achieve semantic security with *Trace*1 as shown below.

For *History* $Hi = (\mathcal{D}, \mathcal{W}, \mathcal{Q} = \{q_1, q_2, \ldots, q_n\})$, *Trace*1 is

- *Size Leakage*: The file set size $|\mathcal{D}|$ and the keyword set size $|\mathcal{W}|$.
- *Result Leakage*: This leakage includes all the search results $(R_1 \ldots R_n)$, $R_i = \{(id, ip), id \in \mathcal{D}(q_i)\}$, and the results of the first search stage $\mathcal{D}(w_{11}) \ldots \mathcal{D}(w_{n1})$, where id is the identifier that matches query q_i, ip is the inner product $\mathcal{I}_2[id] \cdot B f_{q_i}$, and w_{i1} is the selected keyword in query q_i for the first-stage search.
- *Equality Leakage*: This leakage indicates whether two queries have the same w_{i1}. We number the keywords from 1 to $|\mathcal{W}|$, and set *Equality Leakage* to be a vector (EL) of length $|\mathcal{Q}|$, where $EL[i]$ is the sequence number of $w_{i1} \in q_i$.

Theorem 1. *EliMFS-B is semantically secure with Trace1 under the known ciphertext model if* $\mathbf{F}()$ *is a secure psuedo-random function and* $(\mathbf{Enc}, \mathbf{Dec})$ *is a secure symmetric encryption algorithm.*

Proof: To prove this theorem, we have to construct a simulator that is indistinguishable from a real instance of our scheme. Recall that the security game between an attacker $\mathcal{A}$ and a challenger $\mathcal{C}$ under the known ciphertext model is defined as follows.

1) $\mathcal{A}$ chooses a *History* $Hi = (\mathcal{D}, \mathcal{W}, \mathcal{Q} = \{q_1, \ldots, q_n\})$ whose *Trace* is Tr. Then, $\mathcal{A}$ sends Hi and Tr to $\mathcal{C}$.
2) $\mathcal{C}$ randomly chooses a bit $b \xleftarrow{\$} \{0, 1\}$. Then, it calculates $Vi_b = (\mathcal{ED}_b, \mathrm{Index}_b, \mathrm{Token}_{b1}, \ldots, \mathrm{Token}_{bn})$ to get the *View* of Hi and invokes a simulator $\mathcal{S}$ with Tr to obtain a *View* $Vi_{1-b} = (\mathcal{ED}_{1-b}, \mathrm{Index}_{1-b}, \mathrm{Token}_{(1-b)1}, \ldots, \mathrm{Token}_{(1-b)n})$. Both Vi_b and Vi_{1-b} are sent back to $\mathcal{A}$.
3) $\mathcal{A}$ selects $b^{'} \in \{0, 1\}$ as the output of the game.

As described in Definition 1, if there exists a simulator that makes $Pr[b^{'} = b] = \frac{1}{2} + \varepsilon$ where ε is negligible, we can prove that Theorem 1 stands.

The simulator is designed as follows. First, $\mathcal{S}$ randomly selects $f_i^{'} \xleftarrow{\$} \{0, 1\}^{|f_i|}$, $i \in [1, |\mathcal{D}|]$ and sets $\mathcal{ED}^{'} = \{f_i^{'}, i \in [1, |\mathcal{D}|]\}$. Since the file set is encrypted by a secure symmetric encryption algorithm, $\mathcal{ED}_b$ and $\mathcal{ED}^{'}$ are indistinguishable.

After simulating the encrypted file set, the simulator generates the index and the tokens. Since the Bloom filter is encrypted by the kNN encryption, which has been proven to be secure under the known ciphertext model [21], $\mathcal{I}_2$ and $\mathcal{I}_2^{'}$ are indistinguishable. We then describe how $\mathcal{S}$ simulates $\mathcal{I}_1^{'}$:

1) Initialize $\mathcal{I}_1^{'}$ to an empty array.
2) For each $w_i \in \mathcal{W}$, select $tag_i^{'}, K_i^{'} \xleftarrow{\$} \{0, 1\}^{\lambda}$, and $i \in [1, |\mathcal{W}|]$ and initialize $\mathcal{I}_1^{'}[tag_i^{'}]$ to be an empty array.
3) For each $id_j \in \mathcal{D}(w_{i1}), i \in [1, n], j \in [1, |\mathcal{D}|]$, generate $e_{ij}^{'} \leftarrow \mathbf{Enc}(K_{EL[i]}^{'}, id_j)$ and append $e_{ij}^{'}$ to $\mathcal{I}_1^{'}[tag_{EL[i]}^{'}]$.
4) Pad $\mathcal{I}_1^{'}$ to the required size with random data.

The tokens are constructed as follows. For each $q_i \in \mathcal{Q}$, $i \in [1, n]$, get $tag_{EL[i]}^{'}$ and set $\mathrm{Token}_i^{'} = (tag_{EL[i]}^{'}, Enc(Bf_i^{'}), K_{EL[i]}^{'}, thr_i^{'})$. Here, $Enc(Bf_i^{'})$ is generated by the secure kNN encryption $thr_i^{'} = \mathrm{Min}(ip)_i, ip \in R_i$.

Since the keywords are encrypted by a secure pseudo-random function $\mathbf{F}(\cdot)$, file identifiers are encrypted by a secure symmetric encryption algorithm $(\mathbf{Enc}, \mathbf{Dec})$, the Bloom filter is encrypted by the kNN encryption (proven to be secure under the known ciphertext model), and the index and tokens of Vi_b and Vi_{1-b} are indistinguishable.

This is done so that the *Plaintext Leakage* is preserved. Due to the random number $r^{'}$, different tokens will be generated from the same query. The cloud server can only know whether two tokens have the same first-stage searched keyword, not whether they are generated from the same query, so the *Search Pattern Leakage* is semi-protected. Further, all the queried keywords are searched together in the second search stage using an m-length array. As a result, it is impossible for the cloud server to know the search result of any subset of $\vec{w}$ except w_1, and *Subset Leakage* is prevented. ■

B. EliMFS-E: Extended Scheme

1) Advanced threats: Aside from basic leakages, the two-stage structure incurs an additional threat: *Cross Leakage*.

Given two tokens of different queries $Q_1 = (w_1, w_2, \ldots, w_n)$ and $Q_2 = (w_1^{'}, w_2^{'}, \ldots, w_n^{'})$, where

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2765323, IEEE Transactions on Services Computing

w_1 and $w_1^{'}$ are different, the server is able to search the queries $Q_3 = (w_1, w_2^{'}, \ldots, w_n^{'})$, and $Q_4 = (w_1^{'}, w_2, \ldots, w_n)$.

The following example illustrates how cross leakage occurs. Suppose the server receives two tokens, $\text{Token}_1 = (tag_1, Bf_1, K_{e1}, thr_1)$ for query $\vec{w}_1 = (w_1, w_2)$ and $\text{Token}_2 = (tag_2, Bf_2, K_{e2}, thr_2)$ for query $\vec{w}_2 = (w_3, w_4)$. To get the search result of $\vec{w}_3 = (w_3, w_2)$, we focus on the files in $\mathcal{D}(w_3)$, namely $\mathcal{I}_1[tag_2]$. We divide the files that match w_3 into four categories: a) files that match w_3, w_1, and w_2; b) files that match w_3 and w_1 but not w_2; c) files that match w_3 and w_2 but not w_1; d) files that match w_3 only. The judgement conditions are as follows:

$$f \in \mathcal{D}(w_3) \begin{cases} f \in \mathcal{I}_1[tag_1] \begin{cases} \mathcal{I}_2[id_f] \cdot Bf_1 \approx k|\vec{w}_1| & a \\ \mathcal{I}_2[id_f] \cdot Bf_1 \neq k|\vec{w}_1| & b \end{cases} \\ f \notin \mathcal{I}_1[tag_1] \begin{cases} \mathcal{I}_2[id_f] \cdot Bf_1 \approx k(|\vec{w}_1| - 1) & c \\ \mathcal{I}_2[id_f] \cdot Bf_1 \neq k(|\vec{w}_1| - 1) & d \end{cases} \end{cases}$$

The search result of query (w_3, w_2) consists of all the ids in categories a) and c). The result of query $\vec{w}_4 = (w_1, w_4)$ can be obtained similarly.

Although the cloud server is only given a threshold thr_1 instead of $|\vec{w}_1|$, it can still obtain the approximate values of $|\vec{w}_1|$ according to the maximum value of inner products when searching in the second-stage index. Therefore, an extended scheme that can prevent *Cross Leakage* is necessary.

In the known background model, the kNN encryption may no longer be secure, since the cloud server can obtain more historical information. For instance, if the cloud server collects a large number of keywords and their corresponding tokens, it may utilize linear analyses to break the kNN encryption and link plaintext Bloom filters to encrypted ones [22]. Table IV shows the linear equation: $I^T \cdot Bf_q = M_1^T I^{'} \cdot M_1^{-1} Bf_q^{'} + M_2^T I^{''} \cdot M_2^{-1} Bf_q^{''}$, in which only I, an m-bit length vector, is unknown to the cloud server under the known background model. With enough query-token pairs, the cloud server can construct m similar linear equations to derive the coordinates of I.

2) Scheme Construction: The main reason for *Cross Leakage* is that the encrypted Bloom filter of a file is revealed as soon as any of its keywords are searched in the first-stage index. Therefore, we propose EliMFS-E, which solves the problem by binding the Bloom filters in $\mathcal{I}_2$ to both files and keywords. The main process of EliMFS-E is similar to EliMFS-B with several differences.

To prevent linear analyses of the cloud server under the *known background model*, we replace the hash functions $h_1 \ldots h_k$ with keyed hash functions when generating Bloom filters. When generating his secret key SK, the data owner chooses k hash keys ($key_i, \ldots key_k$) and embeds them into SK. To calculate the locations of the k hash values for a keyword w in the Bloom filter, functions $h_i(\vartheta_w, key_i)$, $i \in [1, k]$ are invoked. The detailed **Initialize()** algorithm is similar to that of EliMFS-B and is shown in TABLE V.

First, the data owner picks his secret key, which includes k additional keys ($key_1, key_2, \ldots, key_k$). In Step 2, the Bloom filters of all the files are built and stored in a temporary set ($\mathcal{I}_{temp}$). Each keyword is transformed into an intermediate value and is inserted into $\mathcal{I}_{temp}$ by $h_1(\vartheta_w, key_1), h_2(\vartheta_w, key_2), \ldots, h_k(\vartheta_w, key_k)$, where functions h_1 to h_k are chosen from a hash family $\mathcal{H} = \{h \mid h : \{0,1\}^* \times \{0,1\}^\lambda \rightarrow [1, m]\}$. The indexes $\mathcal{I}_1, \mathcal{I}_2$ are constructed in Step 3. In line 5, the entrance in $\mathcal{I}_1$, the secret key for encrypting file identifiers, and a specific vector (S_w) of each keyword are calculated. Then, for each file that matches w, the data owner inserts the encrypted identifier e into $\mathcal{I}_1$ and sets $\mathcal{I}_2[e]$ as the Bloom filter encrypted by $\{M_1, M_2, S_w\}$. Thus, the data owner can get the whole secret index, $\text{Index} = \{\mathcal{I}_1, \mathcal{I}_2\}$, and sends it to the cloud server with the corresponding encrypted files.

The improvement of the $\mathbf{GenToken}()$ algorithm is similar, as shown in TABLE VI. The hash functions $h_i(\vartheta_w)$ are replaced with keyed hash functions $h_i(\vartheta_w, key_i)$ and the Bloom filter of the query $(w_1, w_2, \ldots)$ is encrypted by (M_1, M_2, S_{w_1}), where S_{w_1} is calculated from the intermediate value of w_1 (namely, ϑ_{w_1}). Because the secret key K_e would no longer be sent to the cloud server, the decryption operation in the $\mathbf{Search}()$ algorithm is removed (TABLE VII).

TABLE V
INITIALIZATION STEPS IN ELIMFS-E

	Step 1:
1	$K_1, K_2, key_1, \ldots, key_k \xleftarrow{\$} \{0,1\}^\lambda$, $M_1, M_2 \in \mathbb{R}^{m \times m}$, $S \in \{0,1\}^m$
2	$SK = \{K_1, K_2, key_1, \ldots, key_k, M_1, M_2, S\}$
	Step 2: Initialize $\mathcal{I}_{temp}$ and for each $w \in \mathcal{W}$:
3	$\vartheta_w \leftarrow \mathcal{F}(w)$, $h_1(\vartheta_w, key_1), h_2(\vartheta_w, key_2), \ldots, h_k(\vartheta_w, key_k)$
4	For all $id \in \mathcal{D}(w)$, set $\mathcal{I}_{temp}[id][i] = 1$, $i \in \{h_1(\vartheta_w, key_1), h_2(\vartheta_w, key_2), \ldots, h_k(\vartheta_w, key_k)\}$
	Step 3: For each $w \in \mathcal{W}$
5	$tag \leftarrow \mathbf{F}(K_1, \vartheta_w)$, $K_e \leftarrow \mathbf{F}(K_2, \vartheta_w)$, $S_w = S$, and set $S_w[h_i(\vartheta_w, key_i)] = 1 - S[h_i(\vartheta_w, key_i)]$, $i \in [1, k]$
6	For each $id \in \mathcal{D}(w)$: • Append $e \leftarrow \mathbf{Enc}(K_e, id)$ to $\mathcal{I}_1[tag]$ • Get $I = \mathcal{I}_{temp}[id]$ and initialize $I^{'}, I^{''}$ to m-bit length vectors • $I^{'}[j] = I^{''}[j] = I[j]$, if $S_w[j] = 1$; $I^{'}[j] = \frac{1}{2} I[j] + r$, $I^{''}[j] = \frac{1}{2} I[j] - r$, if $S_w[j] = 0$, $j \in [1, m]$, r is a random number • $\mathcal{I}_2[e] = \{M_1^T \cdot I^{'}, M_2^T \cdot I^{''}\}$

TABLE VI
GENTOKEN STEPS IN ELIMFS-E

	Step 1:
1	$\vartheta_{w_1} \leftarrow \mathcal{F}(w_1)$, $tag \leftarrow \mathbf{F}(K_1, \vartheta_{w_1})$, $K_e \leftarrow \mathbf{F}(K_2, \vartheta_{w_1})$, $S_{w_1} = S$, and set $S_{w_1}[h_i(\vartheta_{w_1}, key_i)] = 1 - S_{w_1}[h_i(\vartheta_{w_1}, key_i)]$, $i \in [1, k]$
	Step 2:
2	Initialize Bf_q to be a m-bit length array of all zeros
3	For all $w_i \in \vec{w}$: get $\vartheta_{w_i} \leftarrow \mathcal{F}(w_i)$, $h_1(\vartheta_{w_i}, key_1), \ldots, h_k(\vartheta_{w_i}, key_k)$, and set $Bf_q[j] = 1$, $j \in \{h_1(\vartheta_{w_i}, key_1), \ldots, h_k(\vartheta_{w_i}, key_k)\}$
	Step 3:
4	Initialize $Bf_q^{'}, Bf_q^{''}$ to m-bit length vectors
5	$Bf_q^{'}[j] = Bf_q^{''}[j] = Bf_q[j]$ if $S[j] = 0$; $Bf_q^{'}[j] = \frac{1}{2} Bf_q[j] + r^{'}$, $Bf_q^{''}[j] = \frac{1}{2} Bf_q[j] - r^{'}$ if $S[j] = 1$, $j \in [1, m]$, $r^{'}$ is a random number
6	Chose $thr \leqslant k|\vec{w}|$, $\text{Token} = \{tag, M_1^{-1} \cdot Bf_q^{'}, M_2^{-1} \cdot Bf_q^{''}, thr\}$

TABLE VII
SEARCH STEPS IN ELIMFS-E

	Step 1:
1	Retrieve $\mathcal{I}_1[tag]$
	Step 2: For each $e \in \mathcal{I}_1[tag]$
2	Get $\mathcal{I}_2[e] = \{M_1^T \cdot I^{'}, M_2^T \cdot I^{''}\}$
3	$M_1^T I^{'} \cdot M_1^{-1} Bf_q^{'} + M_2^T I^{''} \cdot M_2^{-1} Bf_q^{''} = I^{'T} \cdot Bf_q^{'} + I^{''T} \cdot Bf_q^{''} = I^T \cdot Bf_q$
4	If $I^T \cdot Bf_q \geq thr$, attach id to $\vec{id}$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2765323, IEEE Transactions on Services Computing

3) Analysis: Regarding *efficiency*, EliMFS-E has a higher storage cost ($O(\Sigma_{w\in\mathcal{W}}|\mathcal{D}(w)|)$) than EliMFS-B. The search complexity and communication cost are the same in EliMFS-E as in EliMFS-B: $O(|w_1|)$ and $O(1)$, respectively. EliMFS-E also provides a stronger leakage-resilient ability, which means a higher semantic security guarantee with *Trace*2.

For *History* $Hi = (\mathcal{D}, \mathcal{W}, \mathcal{Q} = \{q_1, q_2, \ldots, q_n\})$, *Trace*2 is

- *Size Leakage*: The file set size $|\mathcal{D}|$, the keyword set size $|\mathcal{W}|$, the quantity of file-keyword pairs $\Sigma_{w\in\mathcal{W}}|\mathcal{D}(w)|$, and the result size of the first-search stage $|\mathcal{D}(w_{i1})|$, where w_{i1} is the selected keyword in query q_i for the first stage search.
- *Result Leakage*: $(R_1 \ldots R_n)$, $R_i = \{(id, ip), e \in \mathcal{D}(q_i)\}$, where id is the identifier that matches query q_i and ip is the inner product $\mathcal{I}_2[id] \cdot Bf_{q_i}$.
- *Equality Leakage*: The sequence number of the selected keyword of each query is $(EL[1], \ldots, EL[n])$.

Theorem 2. *EliMFS-E is semantically secure with Trace2 under the known background model if* $\mathbf{F}()$ *is a secure pseudo-random function and* $(\mathbf{Enc}, \mathbf{Dec})$ *is a secure symmetric encryption algorithm.*

Proof: As shown in the proof of Theorem 1, we have to construct a simulator that produces indistinguishable traces. The difference between the two theorems is that EliMFS-E is designed for the known background model. In the known background model, we allow the cloud server to collect a certain number of keyword-token pairs that are different from the queries used in the game. With these pairs and two *Views* following the same *Trace*, the cloud server should not be able to distinguish which one is generated by the simulator. Formally, the security game between an attacker $\mathcal{A}$ and a challenger $\mathcal{C}$ under the known background model is defined as follows.

1) $\mathcal{A}$ chooses q and sends it to $\mathcal{C}$. Then, $\mathcal{C}$ responds with the corresponding tokens. This step can be carried out by $\mathcal{A}$ adaptively for polynomial times, and the set of all queries is denoted by $\mathcal{Q}'$.
2) $\mathcal{A}$ chooses a *History*, $Hi = (\mathcal{D}, \mathcal{W}, \mathcal{Q} = \{q_1, \ldots, q_n\})$ whose *Trace* is Tr, and $q_j \notin \mathcal{Q}'$ for $1 \le j \le n$. Then $\mathcal{A}$ sends Hi and Tr to $\mathcal{C}$.
3) $\mathcal{C}$ randomly chooses a bit $b \xleftarrow{\$} \{0, 1\}$. Then, it calculates $Vi_b = (\mathcal{ED}_b, \mathrm{Index}_b, \mathrm{Token}_{b1}, \ldots, \mathrm{Token}_{bn})$ to get the *View* of Hi and invokes a simulator $\mathcal{S}$ with Tr to obtain a *View* $Vi_{1-b} = (\mathcal{ED}_{1-b}, \mathrm{Index}_{1-b}, \mathrm{Token}_{(1-b)1}, \ldots, \mathrm{Token}_{(1-b)n})$. Both Vi_b and Vi_{1-b} are sent back to $\mathcal{A}$.
4) $\mathcal{A}$ selects $b' \in \{0, 1\}$ as the output of the game.

$\mathcal{S}$ simulates $\mathcal{ED}'$ in the same way here as in the proof of Theorem 1. To simulate the index and tokens, $\mathcal{S}$ first randomly picks $M_1', M_2' \in \mathbb{R}^{m\times m}$, $S' \in \{0, 1\}^m$. The index is generated as follows.

1) Initializes $\mathcal{I}_1', \mathcal{I}_2'$, and $\mathcal{I}_{temp}'$ to empty arrays, and initializes each item of $\mathcal{I}_{temp}'$ to be an m bit length array of all zeros.
2) For each $q_i \in \mathcal{Q}$ and $i \in [1, n]$,
   a) Generate a Bloom filter (Bf_i') in which the number of bit 1 equals $\mathrm{Max}(ip), ip \in R_i$.
   b) For each $(id_j, ip_j) \in R_i$ and $j \in [1, |\mathcal{D}|]$, randomly choose ip_j 1s from Bf_i', construct a bit array Bf_{ij}', and set $\mathcal{I}_{temp}'[j]$ as $\mathcal{I}_{temp}'[j] + Bf_{ij}'$.
   c) For the elements of $\mathcal{I}_{temp}'$ that are bigger than 1, replace them with 1.
3) For each $q_i \in \mathcal{Q}$, if there does not exist $EL[j] = EL[i] (j < i)$, select $tag_{EL[i]}', e_1', \ldots, e_{|\mathcal{D}(w_{i1})|}' \xleftarrow{\$} \{0, 1\}^\lambda$ and set $\mathcal{I}_1'[tag_{EL[i]}'] = (e_1', \ldots, e_{|\mathcal{D}(w_{i1})|}')$.
4) Choose a free e' for each $id_x \in R_i$, set $\mathcal{I}_2'[e']$ to be encrypted $\mathcal{I}_{temp}'[x]$, and mark e' as used.
5) For the other free e', generate elements in $\mathcal{I}_2'$ for them randomly.
6) Pad $\mathcal{I}_1'$ and $\mathcal{I}_2'$ to the required size with random data.

With all this be done, $\mathcal{I}_1'$ and $\mathcal{I}_2'$ are built.

Then, $\mathcal{S}$ constructs the tokens. For each $q_i \in \mathcal{Q}$ and $i \in [1, n]$, $\mathcal{S}$ gets $tag_{EL[i]}'$ and encrypts Bf_i' with $EL[i]$ and SK'. $\mathcal{S}$ sets $\mathrm{Token}_i' = (tag_{EL[i]'}, Enc(Bf_i'), thr_i')$, $thr_i' = Min(ip)_i, ip \in R_i$.

Basic leakages are not incurred since the files are encrypted by semantically secure symmetric encryption, keywords are encrypted by a secure pseudo-random function $\mathbf{F}()$, and file identifiers are encrypted by a secure symmetric encryption algorithm $(\mathbf{Enc}, \mathbf{Dec})$. Due to the the indistinguishability of keyed hash functions, no PPT adversary is able to carry out linear analysis on kNN encryption, so EliMFS-E is secure under the known background model.

Given two tokens of queries $\vec{w}_1 = (w_1, w_2)$ and $\vec{w}_2 = (w_3, w_4)$, $\mathrm{Token}_1 = (tag_1, Bf_1, thr_1), \mathrm{Token}_2 = (tag_2, Bf_2, thr_2)$, and the server cannot get the correct inner product of the Bloom filters encrypted by w_3', or of Bf_1, which is encrypted by w_1'. Therefore, the approach for obtaining the search result of (w_3, w_1) mentioned above no longer works, and *Cross Leakage* is prevented. Note that the search result of the first-stage search can be easily covered up by obfuscating operations like padding each item in $\mathcal{I}_1$ to a fixed size with random encrypted file identifiers. ■

V. Theoretical Analysis

Before dealing with experiments, we first analyze the performance of our schemes theoretically. As shown in TABLE VIII, we compare the asymptotic performance of our schemes with the schemes of [5], [11] and [23].

[23] provided a fuzzy keyword searchable encryption scheme with an inverted index of size $O(\mathcal{W})$. In this scheme, the entire index must be scanned to find the fuzzy result of a keyword. Thus, its search time complexity is $O(|\mathcal{W}|)$. Wang *et al.* [5] proposed a method of searching multiple keywords over an inverted index. To get the result, the cloud server has to compute the product of the token, whose length is $|\mathcal{W}|$, with the entire inverted index, which includes a $|\mathcal{W}| \times |\mathcal{W}|$ matrix. As a result, the efficiency of this scheme is not quite satisfactory, and it does not support a fuzzy keyword search. The scheme of [11] provided the functions of both multi-keyword and fuzzy keyword searches. However, its search time

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2765323, IEEE Transactions on Services Computing

TABLE VIII
COMPARISON OF ASYMPTOTIC PERFORMANCE

	Multi-keyword Search	Fuzzy Search	Search Time	Communication Cost	Storage Cost
paper [23]	×	√	$O(\|\mathcal{W}\|)$	$O(1)$	$O(\|W\|)$
paper [5]	√	×	$O(\|\mathcal{W}\|)$	$O(\|\mathcal{W}\|)$	$O(\|\mathcal{W}\|^2)$
paper [11]	√	√	$O(\|\mathcal{D}\|)$	$O(1)$	$O(\|\mathcal{D}\|)$
EliMFS-B	√	√	$O(\|\mathcal{D}(w_1)\|)$	$O(1)$	$O(\|\mathcal{W}\|+\|\mathcal{D}\|)$
EliMFS-E	√	√	$O(\|\mathcal{D}(w_1)\|)$	$O(1)$	$O(\Sigma_{w\in\mathcal{W}}\|\mathcal{D}(w)\|)$

is linear with the size of the file set because it utilizes a forward index.

The search complexity of EliMFS-B is $O(|\mathcal{D}(w_1)|)$ since the search time of the first-stage index is $O(1)$ and the second search stage only takes place in the results of the first search stage. This scheme needs only one round of communication, and its token size is constant. Therefore, its communication cost is constantly $O(1)$, no matter how many keywords are searched at one time. The storage cost of the cloud server is the space for $\mathcal{I}_1$ and $\mathcal{I}_2$; that is, $O(|\mathcal{W}|+|\mathcal{D}|)$. In conclusion, EliMFS-B has the best performance in all efficiency aspects with a basic semantic security guarantee.

EliMFS-E has a performance similar to EliMFS-B. It has a larger storage cost, but its leakage-resilient ability has improved. Since $\mathcal{I}_1$ is an inverted index and each file-keyword pair corresponds to an item in $\mathcal{I}_2$, the storage space for the index is $O(\mathcal{W}+\Sigma_{w\in\mathcal{W}}|\mathcal{D}(w)|)$, which is approximate to $O(\Sigma_{w\in\mathcal{W}}|\mathcal{D}(w)|)$. Suppose the client has a file set of 10000 files and each file contains 150 keywords; the size of the index of EliMFS-E is about 30GB when $m = 5000$. For the same file set, the size of the index of scheme EliMFS-B is only about 200MB. The querying and searching processes of EliMFS-E are similar to those of EliMFS-B. Thus, the communication cost and the search complexity remain the same. But the search time of EliMFS-E may be even shorter than that of EliMFS-B because all the Bloom filters that match the same keyword can be stored in consecutive positions. This means that only one disk read operation is needed. The decryption operation $id \leftarrow \mathbf{Dec}(K_e, e)$ is also omitted in EliMFS-E. The trade-off is worthwhile because the storage space of the cloud server is enormous. In our future works, we will improve this scheme by optimizing the storage cost with some strategies.

VI. EXPERIMENTAL ANALYSIS

To present practical utility, we tested our schemes on a real-world dataset: 10000 files with 16027 keywords selected from the Enron Email Dataset [24]. We implemented prototypes of EliMFS-B, EliMFS-E, and the basic scheme of [11], which is represented as MFS.

The programs were written in Java and were executed on a desktop equipped with an Intel Core i5-4200H CPU at 2.8 GHz, 8 GB RAM, and a 64-bit Windows 10. For simplicity, the AES algorithm was used as the psuedo-random function $\mathbf{F}()$ and the CPA secure symmetric encryption algorithm $(\mathbf{Enc}, \mathbf{Dec})$. SHA-256 and HmacSHA1 were utilized in $h_i(\vartheta_w)$ and $h_i(\vartheta_w, key_i)$, respectively.

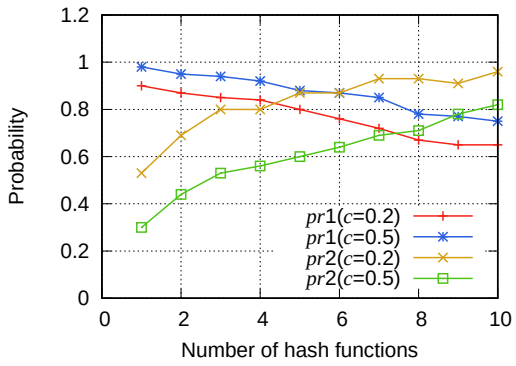


Fig. 3. The performance of keyword fuzzifying with various ls and cs

A. Search Accuracy

To achieve the best accuracy for the keyword search, the parameters in LSH and the Bloom filter need to be selected prudently. The fuzzy search is realized with Locality-Sensitive Hashing, as described. We employ a 2-stable distribution LSH family ($\mathcal{F}$) which contains l hash functions, and each hash function has the expression

$$h_{(\vec{a},b)}(\vec{v}) = \lfloor \frac{\vec{a}\cdot\vec{v}+b}{c} \rfloor$$

Here, $\vec{a}$ is a $26\times 26 = 676$ dimensional vector, and each of its dimensions are randomly chosen from the distribution defined by $f(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2}$. $b \in [0, c]$ is a random real number. While c is a fixed real number for one hash family, different choices of $\vec{a}$ and b result in different hash functions.

To achieve the optimal performance of the fuzzy search, a large number of keywords are involved in our experiment. The intermediate values of these keywords and the keywords that are similar to them are calculated to evaluate the fuzzy level. In our experiment, we use two probabilities to evaluate the accuracy of the fuzzy keyword search:

- True positive: $pr1 = \{\mathcal{F}(w_1) = \mathcal{F}(w_2)|w_1$ and w_2 only one character distinction exists.$\}$
- True negative: $pr2 = \{\mathcal{F}(w_1) \neq \mathcal{F}(w_2)|w_1$ and w_2 multiple character distinctions exist.$\}$

These probabilities should be as high as possible.

Fig. 3 shows that $pr1$ decreases when the quantity of the hash functions in $\mathcal{F}$ gets larger, while increases when c becomes bigger. However, $pr2$ acts the opposite way. That is, $pr2$ increases as the number of the hash functions in $\mathcal{F}$ increases, and decreases with the increment of c. Therefore, it is important to make a trade-off between $pr1$ and $pr2$. In the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2765323, IEEE Transactions on Services Computing

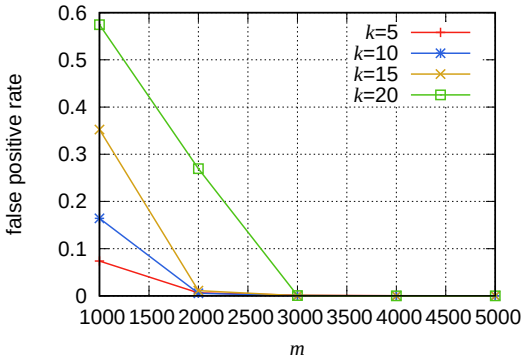


Fig. 4. The false positive rates of Bloom filter with various ms and ks

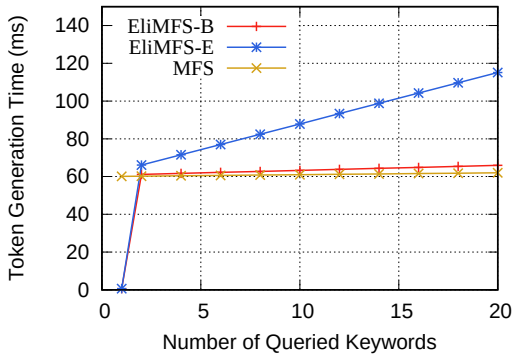


Fig. 6. The token generation times with different $|\vec{w}|$s

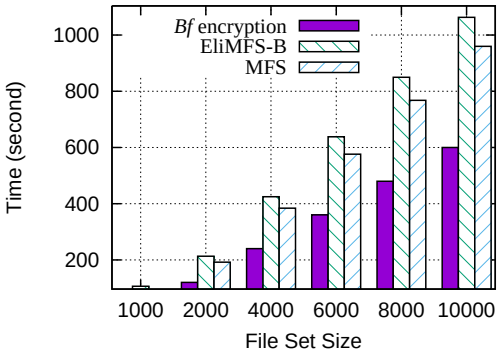


Fig. 5. The index generation times when $|\mathcal{D}|$ ranges

following experiments, we set $l = 7$ and $c = 0.3$ so that $pr1$ and $pr2$ are both larger than 0.8.

Bloom filter has a distinct advantage on spacial and temporal cost on searching compared with other data structures. However, it also has some shortcomings. There exist false positives when searching with Bloom filter, and the false positive is $P_{fp} = (1 - e^{-kn/m})^k$ [15], where n is the number of the elements in the set, m is the length of the bit array, and k is the number of hash functions. In our experiments, 100 to 182 keywords are extracted from one file, so we set n to be 180. As shown in Fig 4, the false positive rates are tested under different ms and ks. The result indicates that, fp grows with k when m is small, and fp declines with the increase of k when m is large. And fp is always smaller than 0.0004 when m is larger than 3000. Therefore in the following experiments, we set m to 5000 and k to 20 for the best search accuracy.

Due to the existence of false positives and false negatives, some matched files may have a lower inner product than other files. A trade-off between false positives and false negatives can be made by choosing a proper threshold (thr).

B. Efficiency

In this subsection, we mainly analyze the time consumption of our schemes from different aspects.

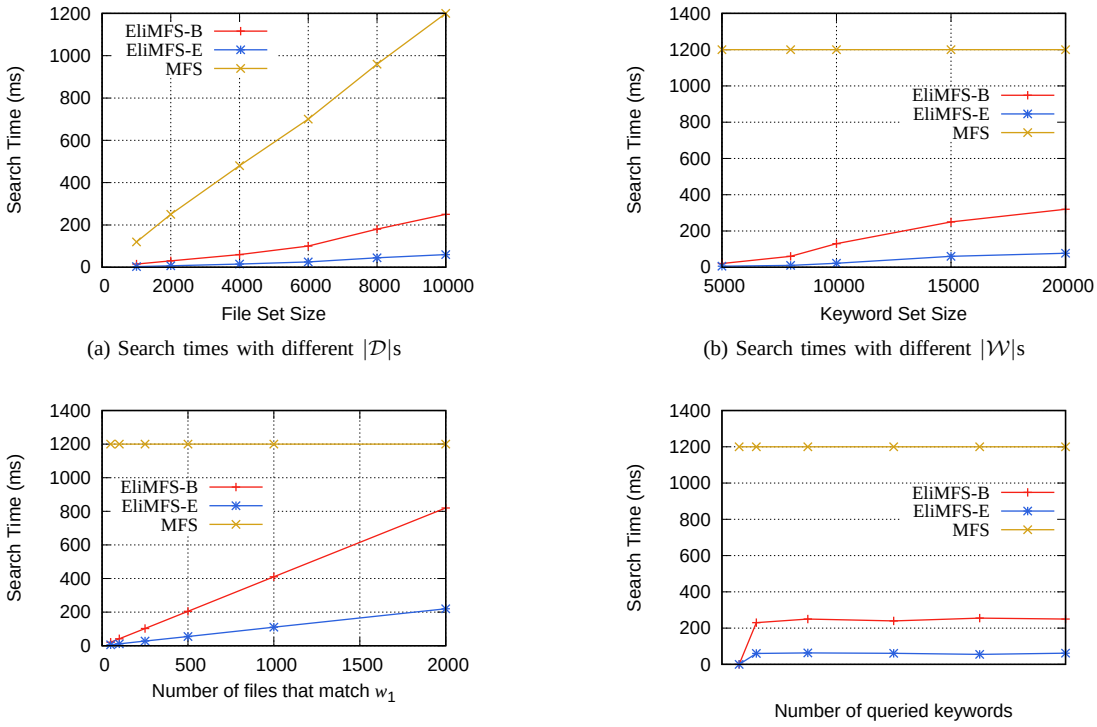
1) Index building time: In Fig. 5, the columns with stripes show the index generation time of our EliMFS-B and MFS. We can see that the generation time increases linearly with $|\mathcal{D}|$. EliMFS-B takes a little more time than MFS because it needs to build two indexes, $\mathcal{I}_1$ and $\mathcal{I}_2$, while MFS only needs to build one. However, the difference is quite small since the most time-consuming operation is the encryption of the Bloom filter, shown as the purple columns. Thus, faster matrix calculation can shorten the initialization time effectively. Index generation is executed only once when the system starts up, which has little influence on the efficiency of the whole system. Therefore, the extra time consumption of EliMFS-B is acceptable.

2) Token generation time: Fig. 6 shows the token generation time with different keyword numbers for querying. When only one keyword is searched, our schemes show a huge advantage. The token generation time is quite small because only a tag is computed. When multiple keywords are searched, the lines jump up to more than 60ms because the most time-intensive step in this phase is the encryption of the Bloom filter, which costs nearly 60ms. It is obvious that the token generation time rises linearly when the number of keywords grows. In addition, the growth rate of EliMFS-E is larger than that of both EliMFS-B and MFS. This is because $|\vec{w}|$ times of operations are needed to insert all the queried keywords into the Bloom filter, and the keyed hash function $h_i(\vartheta_w, key_i)$ takes more time than $h_i(\vartheta_w)$. Even though EliMFS-B and EliMFS-E take a little more time to generate tokens, they can obtain more benefits from the search time, as shown below.

3) Search time: As the most important performance metric, search times with various impact factors are presented in Fig. 7. Fig. 7a reflects the relationship between the search time and the file set size. With the growth of the file set size, the curve of MFS ascends linearly, while the curves of EliMFS-B and EliMFS-E grow slowly at a low level. This is because in MFS, the search process always has to go through the entire file set. The search times of our schemes, on the other hand, only rely on the results of the first search stage. When the file set gets larger, the size of the search result in the first stage grows, which leads the second search stage to expend more time. EliMFS-B is a little slower than EliMFS-E because it executes one more operation: $id \leftarrow \mathbf{Dec}(K_e, e)$.

The relationship between the search time and three other parameters are displayed in Figs. 7b, 7c, and 7d, when the file set size is fixed at 10000. The search time of MFS

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2765323, IEEE Transactions on Services Computing



(a) Search times with different $|\mathcal{D}|$s

(b) Search times with different $|\mathcal{W}|$s

(c) Search times with different $|\mathcal{D}(w_1)|$s

(d) Search times with different $|\vec{w}|s$

Fig. 7. The search time of EliMFS-B, EliMFS-E, and MFS

remains unchanged in these three images since the file set size is constant. In Fig. 7b, different quantities of keywords are extracted from these files so that various sizes of $\mathcal{W}$ are made up. We can see that the line trends of our schemes are similar to those in Fig. 7a: the search times of our schemes are much shorter than that of MFS, and they grow slowly with $|\mathcal{W}|$. The more keywords extracted from a file, the more likely a file will match. Therefore, more results can be obtained in the first search stage. This is why the search times of our schemes increase with the growth of the keyword set size.

Fig. 7c demonstrates the relationship between the search time and $|\mathcal{D}(w_1)|$ when searching in 10000 files, where w_1 is the selected keyword in a query for the first stage search. The search time of MFS stays constant as mentioned above, and in EliMFS-B and EliMFS-E, the search time linearly increases with $|\mathcal{D}(w_1)|$ because the server needs to search over all the files that match w_1 to get an accurate result for the queried keyword set. Since w_1 is the keyword that matches the fewest files, the size of $\mathcal{D}(w_1)$ would not be very large. Thus, EliMFS-B and EliMFS-E search much faster than MFS, except in the extreme case where w_1 matches almost all the files in $\mathcal{D}$.

In Fig. 7d, when the file set size is fixed (10000 files and 16027 keywords), the search time is almost independent of the query size in all three schemes; in such a scenario, our schemes spend far less time than MFS. When only one keyword is queried, the second search stage will be omitted, and as a result, the search time is extremely short. When $|\vec{w}| > 1$, the search time remains stable no matter how many keywords are searched.

In conclusion, our EliMFS schemes based on LSH, the Bloom filter, and a two-stage index, have a similar token generation time as MFS, but reduce the search time remarkably.

VII. Related Work

Searchable Encryption (SE) enables data owners to outsource their private files to a semi-trusted server without revealing the plaintexts while simultaneously guaranteeing keyword search functions. Currently, researchers propose many state-of-the-art SSE schemes: [2] proposed the first Searchable Symmetric Encryption (SSE) scheme, and full-domain search scheme rather than an index-based scheme; [25] and [26] presented SE schemes based on asymmetric encryption; [27] and [28] dealt with a malicious cloud server and proposed schemes to support verifiable and searchable symmetric encryption; [29] redefined the security of SE schemes based on the adversarial server's prior knowledge; [6] and [8] proposed schemes that support dynamic updating; [30] improved SE schemes in terms of the index I/O efficiency; [31] utilized attribute-based encryption to achieve verifiable multi-user SSE; [32] proposed a new verifiable database (VDB) framework so that clients cannot only retrieve database records, but also detect any attempt by the server to tamper with the data; [33] first applied encrypted keyword searching on de-duplicated data; and [34] and [35] applied searchable encryptions to specific applications, such as public key encryption and E-Health Clouds. However, none of these works support multi-keyword or fuzzy keyword searches.

A. Multi-keyword Search

The *multi-keyword search* schemes can search multiple keywords simultaneously over encrypted data. [3] constructed a binary data vector for each file where each bit in the vector represents a keyword of the file. This structure is similar to schemes based on the Bloom filter when $k = 1$, where k is the number of hash functions. In [36], a tree structure index based on term frequency was used to support ranked searches, and the cosine similarity measure was exploited to check whether the file index contained indicated keywords. [4] utilized an inverted index and a hash table to improve efficiency, but increased communication costs. Furthermore, [9] expanded [4] to a multi-user scenario. [37] supported a pattern-matching string search, a more flexible method than a general boolean search SSE. [5] achieved a multi-keyword search with an inverted index for the first time, but its efficiency was not quite satisfactory. [38] dealt with ranked multi-keyword searches in a multi-owner mode based on a bilinear map and the Decisional Bilinear Diffie-Hellman (DBDH) assumption, but an administration server was needed. All of, these schemes only consider *exact keyword searches*.

B. Fuzzy Keyword Search

To deal with spelling mistakes and searches for similar keywords, multiple *fuzzy keyword search* schemes were proposed [39], [40]. [7] proposed a wildcard-based fuzzy keyword search over encrypted data. [41] improved it with a smaller index. The work of [42], which proposed a higher security guarantee but with a larger space requirement, was also based on [7]. Finger-prints were extracted from keywords and encrypted with a secure kNN encryption to achieve a top-k fuzzy search in [23]. Xu et al. [43] exploited edit distances to quantify keyword similarities in their work, and they also employed a TF-IDF (Term Frequency and Inverse Document Frequency) rule to rank the results. However, most existing works (except [10] and [11]) do not support multi-keyword and fuzzy searches simultaneously.

C. Multi-keyword Fuzzy Search

To implement a fuzzy search, [10] exploited a Bedtree inverted index and realized multi-keyword search functions through pre-defined phrases, e.g. “Cloud Storage” as a single keyword. However, in this scheme, one Bloom filter must be built for each edit distance value of each keyword, which leads to a tremendous index size. In [11], Wang et al. used the forward index, the Bloom filter, and Locality-Sensitive Hashing techniques to implement multi-keyword fuzzy search. However, its search time is linear with the file set size in the cloud. Therefore, its performance may be severely influenced when there are huge file sets to search. In [44], Wang et al. proposed a ranked multi-keyword fuzzy search on encrypted data based on a two-layered index structure. Unfortunately, both of the two layers use forward index, which is still linear with the file set size in the cloud. Fu et al. [16] improved search accuracy via a uni-gram based keyword transformation method while remaining the search time linearly with the file set size. Our paper focuses on further improving the search performance without hurting the system’s privacy in terms of leakages.

VIII. Conclusion

In this paper, we focus on a multi-keyword fuzzy search over a large encrypted file set in cloud storage. An Efficient Leakage-resilient Multi-keyword Fuzzy Search (EliMFS) framework is proposed for the encrypted cloud data; it consists of a novel two-stage index structure, Locality-Sensitive Hashing, and a Bloom filter. The two-stage index structure ensures that the search time is linearly independent of the file set size. Meanwhile, the multi-keyword fuzzy search function is implemented based on the Gram Counting Order, the Bloom filter, and Locality-Sensitive Hashing. Regarding the leakages caused by the two-stage index, we present two schemes to handle threats in different threat models. Theoretical analysis and experimental evaluations demonstrate our design’s practicality.

Acknowledgment

This research was supported in part by the Key Laboratory of Aerospace Information Security and the Trusted Computing Ministry of Education; by NSF grants CNS 1460971, CNS 1439672, CNS 1301774, ECCS 1231461, ECCS 1128209, and CNS 1138963; by the National Natural Science Foundation of China under grants 61272451, 61572380, U1536204; by the Major State Basic Research Development Program of China under grant 2014CB340600; and by the National High Technology Research and Development Program (863 Program) of China under grant 2014BAH41B00. The corresponding author is Kun He.

References

[1] K. He, J. Chen, R. Du, Q. Wu, G. Xue, and X. Zhang, “DeyPoS: Deduplicatable dynamic proof of storage for multi-user environments,” *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3631–3645, 2016.

[2] D. X. Song, D. Wagner, and A. Perrig, “Practical techniques for searches on encrypted data,” in *2013 IEEE Symposium on Security and Privacy*, (Los Alamitos, CA, USA), IEEE Computer Society, 2000.

[3] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, “Privacy-preserving multi-keyword ranked search over encrypted cloud data,” in *Proceedings of 2011 IEEE INFOCOM International Conference on Computer Communications*, 2011.

[4] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rou, and M. Steiner, “Highly-scalable searchable symmetric encryption with support for boolean queries,” in *Advances in Cryptology CRYPTO 2013*, Lecture Notes in Computer Science, pp. 353–373, Springer Berlin Heidelberg, Jan. 2013.

[5] B. Wang, W. Song, W. Lou, and Y. T. Hou, “Inverted index based multi-keyword public-key searchable encryption with strong privacy guarantee,” in *Proceedings of the 2015 IEEE INFOCOM International Conference on Computer Communications*, 2015.

[6] M. Naveed, M. Prabhakaran, and C. A. Gunter, “Dynamic searchable encryption via blind storage,” Tech. Rep. 219, 2014.

[7] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, “Fuzzy keyword search over encrypted data in cloud computing,” in *Proceedings of the 2010 IEEE INFOCOM International Conference on Computer Communications*, 2010.

[8] E. Stefanov, C. Papamanthou, and E. Shi, “Practical dynamic searchable encryption with small leakage,” *IACR Cryptology ePrint Archive*, 2013.

[9] S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, “Outsourced symmetric private information retrieval,” in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, CCS ’13, (New York, NY, USA), pp. 875–888, ACM, 2013.

[10] M. Chuah and W. Hu, "Privacy-aware BedTree based solution for fuzzy multi-keyword search over encrypted data," in *Proceedings of the 2011 ICDCSW International Conference on Distributed Computing Systems*, pp. 273–281, June 2011.

[11] B. Wang, S. Yu, W. Lou, and Y. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *Proceedings of the 2014 IEEE INFOCOM International Conference on Computer Communications*, 2014.

[12] E.-J. Goh and others, "Secure indexes," *IACR Cryptology ePrint Archive*, vol. 2003, p. 216, 2003.

[13] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, (New York, NY, USA), pp. 79–88, ACM, 2006.

[14] "Locality-sensitive hashing," Jan. 2016. Page Version ID: 701793057.

[15] "Bloom filter," Feb. 2016. Page Version ID: 704138885.

[16] Z. Fu, X. Wu, C. Guan, X. Sun, and K. Ren, "Toward Efficient Multi-Keyword Fuzzy Search Over Encrypted Outsourced Data With Accuracy Improvement," *IEEE Transactions on Information Forensics and Security*, vol. 11, pp. 2706–2716, Dec. 2016.

[17] J. Chen, K. He, Q. Yuan, G. Xue, R. Du, and L. Wang, "Batch identification game model for invalid signatures in wireless mobile networks," *IEEE Transactions on Mobile Computing*, 2016.

[18] A. Behm, S. Ji, C. Li, and J. Lu, "Space-Constrained Gram-Based Indexing for Efficient Approximate String Search," in *Proceedings of the 25th IEEE ICDE International Conference on Data Engineering*, 2009.

[19] M. Datar and P. Indyk, "Locality-sensitive hashing scheme based on p-stable distributions," in *In SCG 04: Proceedings of the twentieth annual symposium on Computational geometry*, pp. 253–262, ACM Press, 2004.

[20] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[21] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis, "Secure kNN Computation on Encrypted Databases," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, (New York, NY, USA), pp. 139–152, ACM, 2009.

[22] B. Yao, F. Li, and X. Xiao, "Secure nearest neighbor revisited," in *Proceedings of the 2013 ICDE International Conference on Data Engineering*, 2013.

[23] D. Wang, S. Fu, and M. Xu, "A privacy-preserving fuzzy keyword search scheme over encrypted cloud data," in *Proceedings of the 2013 IEEE CloudCom International Conference on Cloud Computing Technology and Science*, 2013.

[24] EDRM, "Enron email dataset."

[25] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology-Eurocrypt 2004*, pp. 506–522, Springer, 2004.

[26] M. Bellare, A. Boldyreva, and A. ONeill, "Deterministic and efficiently searchable encryption," in *Advances in Cryptology - CRYPTO 2007*, Lecture Notes in Computer Science, pp. 535–552, Springer Berlin Heidelberg, 2007.

[27] R. Cheng, J. Yan, C. Guan, F. Zhang, and K. Ren, "Verifiable Searchable Symmetric Encryption from Indistinguishability Obfuscation," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, (New York, NY, USA), pp. 621–626, ACM, 2015.

[28] W. Sun, X. Liu, W. Lou, Y. T. Hou, and H. Li, "Catch you if you lie to me: Efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data," in *Proceedings of the 2015 IEEE INFOCOM International Conference on Computer Communications*, pp. 2110–2118, Apr. 2015.

[29] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-Abuse Attacks Against Searchable Encryption," in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, (New York, NY, USA), pp. 668–679, ACM, 2015.

[30] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Dynamic searchable encryption in very large databases: Data structures and implementation," in *Proceedings of the 2014 NDSS Network and Distributed System Security Symposium*, vol. 14, 2014.

[31] Q. Zheng, S. Xu, and G. Ateniese, "VABKS: Verifiable attribute-based keyword search over outsourced encrypted data," in *Proceedings of the 2014 IEEE INFOCOM International Conference on Computer Communications*, 2014.

[32] X. Chen, J. Li, X. Huang, and M. Jianfeng, "New Publicly Verifiable Databases With Efficient Updates," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 5, pp. 1–1, 2014.

[33] J. Li, X. Chen, F. Xhafa, and L. Barolli, "Secure Deduplication Storage Systems Supporting Keyword Search," *Journal of Computer and System Sciences*, vol. 81, no. 8, pp. 1532–1541, 2015.

[34] R. Chen, Y. Mu, G. Yang, F. Guo, and X. Wang, "Dual-Server Public-Key Encryption With Keyword Search for Secure Cloud Storage," *IEEE Transactions on Information Forensics and Security*, vol. 11, pp. 789–798, Apr. 2016.

[35] Y. Yang and M. Ma, "Conjunctive Keyword Search With Designated Tester and Timing Enabled Proxy Re-Encryption Function for E-Health Clouds," *IEEE Transactions on Information Forensics and Security*, vol. 11, pp. 746–759, Apr. 2016.

[36] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li, "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," in *Proceedings of the 2013 ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, (New York, NY, USA), pp. 71–82, ACM, 2013.

[37] D. Wang, X. Jia, C. Wang, K. Yang, S. Fu, and M. Xu, "Generalized pattern matching string search on encrypted data in cloud systems," in *Proceedings of the 2015 IEEE INFOCOM International Conference on Computer Communications*, pp. 2101–2109, Apr. 2015.

[38] W. Zhang, Y. Lin, S. Xiao, J. Wu, and S. Zhou, "Privacy Preserving Ranked Multi-Keyword Search for Multiple Data Owners in Cloud Computing," *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1566–1577, 2016.

[39] J. Wang, H. Ma, Q. Tang, J. Li, H. Zhu, S. Ma, and X. Chen, "Efficient verifiable fuzzy keyword search over encrypted data in cloud computing," *Computer science and information systems*, vol. 10, no. 2, pp. 667–684, 2013.

[40] P. Xu, H. Jin, Q. Wu, and W. Wang, "Public-key encryption with fuzzy keyword search: A provably secure scheme under keyword guessing attack," *IEEE Transactions on computers*, vol. 62, no. 11, pp. 2266–2277, 2013.

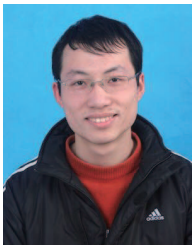
[41] C. Liu, L. Zhu, L. Li, and Y. Tan, "Fuzzy keyword search on encrypted cloud storage data with small index," in *Proceedings of the 2011 IEEE CCIS International Conference on Cloud Computing and Intelligence Systems*, 2011.

[42] A. Boldyreva and N. Chenette, "Efficient Fuzzy Search on Encrypted Data," in *Fast Software Encryption*, Lecture Notes in Computer Science, pp. 613–633, Springer Berlin Heidelberg, 2014.

[43] Q. Xu, H. Shen, Y. Sang, and H. Tian, "Privacy-preserving ranked fuzzy keyword search over encrypted cloud data," in *International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pp. 239–245, 2013.

[44] J. Wang, X. Yu, and M. Zhao, "Privacy-Preserving Ranked Multi-keyword Fuzzy Search on Cloud Encrypted Data Supporting Range Query," *Arabian Journal for Science & Engineering (Springer Science & Business Media B.V.)*, vol. 40, pp. 2375–2388, Aug. 2015.

Jing Chen received his Ph.D. degree in computer science from Huazhong University of Science and Technology, Wuhan. He is currently a full professor at the Computer School at Wuhan University. His research interests are in cloud security and network security. He has published more than 80 research papers in many international journals and conferences, such as IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, IEEE Transactions on Mobile Computing, INFOCOM, SECON, and TrustCom.



Kun He is a PH.D student at Wuhan University. His research interests include cryptography, network security, mobile computing, and cloud computing. He has published research papers in IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, IEEE Transactions on Mobile Computing, Security and Communication Networks, and IEEE TRUSTCOM.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2017.2765323, IEEE Transactions on Services Computing

Lan Deng received her B.S. degree in information security at Wuhan University, Wuhan, China in 2013. Currently she is a Master degree candidate in the Computer School at Wuhan University. Her research interests include cryptography, network security, secure de-duplication, and searchable encryption. She has published research papers in IEEE TRUSTCOM, 2014. She also served as a reviewer for IEEE Transactions on Information Forensics & Security.

Quan Yuan is an assistant professor in the Department of Math and Computer Science at the University of Texas-Permian Basin, TX, USA. His research interests include mobile computing, routing protocols, peer-to-peer computing, parallel and distributed systems, and computer networks. He has published more than 30 research papers in many international journals and conferences, such as IEEE Transactions on Parallel and Distributed Systems, INFOCOM, MobiHoc, SECON, and TrustCom.

Ruiying Du received her BS, MS, and PH.D degrees in computer science in 1987, 1994, and 2008, from Wuhan University, Wuhan, China. She is a full professor at the Computer School at Wuhan University. Her research interests include network security, wireless networks, cloud computing, and mobile computing. She has published more than 80 research papers in many international journals and conferences, such as IEEE Transactions on Parallel and Distributed Systems, the International Journal of Parallel and Distributed Systems, INFOCOM, SECON, TrustCom, and NSS.

Yang Xiang received his PhD in computer science from Deakin University, Australia. He is currently a full professor at the School of Information Technology, Deakin University. His research interests include network and system security, distributed systems, and networking. He has published more than 130 research papers in many international journals and conferences, such as IEEE Transactions on Computers, IEEE Transactions on Information Security and Forensics. He has been the PC member for more than 60 international conferences in networking and security. He serves as the Associate Editor of IEEE Transactions on Computers, Security and Communication Networks (Wiley).

Jie Wu is the Associate Vice Provost for International Affairs at Temple University. He also serves as Director of the Center for Networked Computing and as Laura H. Carnell professor. He served as Chair of Computer and Information Sciences from 2009 to 2016. Prior to joining Temple University, he was a program director at the National Science Foundation and was a distinguished professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Dr. Wu regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Service Computing and the Journal of Parallel and Distributed Computing. Dr. Wu was general co-chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, ACM MobiHoc 2014, ICPP 2016, and IEEE CNS 2016, as well as program co-chair for IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a CCF Distinguished Speaker and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.