2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering

Achieving Secure and Effective Search Services in Cloud Computing

Qin Liu[†*], Shuyu Pei[†], Kang Xie[‡*], Jie Wu[§], Tao Peng[¶], and Guojun Wang[¶]
[†]College of Computer Science and Electronic Engineering, Hunan University, P. R. China, 410082
[‡] Key Lab of Information Network Security, Ministry of Public Security, P. R. China, 201204
[§]Center for Networked Computing, Temple University, Philadelphia, PA 19122, USA
[¶] School of Computer Science and Educational Software, Guangzhou University, P. R. China, 510006
[*]Correspondence to: gracelq628@hnu.edu.cn; xiekang@stars.org.cn

***Abstract*—One critical challenge of today's cloud services is how to provide an effective search service while preserving user privacy. In this paper, we propose a wildcard-based multi-keyword fuzzy search (WMFS) scheme over the encrypted data, which tolerates keyword misspellings by exploiting the indecomposable property of primes. Compared with existing secure fuzzy search schemes, our WMFS scheme has the following merits: 1) *Efficiency*. It eliminates the requirement of a predefined dictionary and thus supports updates efficiently. 2) *High accuracy*. It eliminates the false positive and false negative introduced by specific data structures and thus allows the user to retrieve files as accurate as possible. 3) *Flexibility*. It gives the user great flexibility to specify different search patterns including keyword and substring matching. Extensive experiments on a real data set demonstrate the effectiveness and efficiency of our scheme.**

***Index Terms*—cloud computing, searchable encryption, fuzzy search, wildcard.**

I. Introduction

As cloud computing provides overwhelming benefits to consumers, outsourcing data services is booming. To protect data security from the cloud service provider (CSP), many research and techniques have been put forward [1], [2]. Searchable encryption (SE) [3]–[5] that enables secure searches over the encrypted data allows the user to retrieve data of interest in a privacy-preserving way. Compared with *exact* search, *fuzzy* search allows the user to enter keywords with uncertainties or inconsistencies in their forms, and thus it can greatly improve the user experience of query services.

Recently, Li et al. [6] proposed a wildcard-based fuzzy search scheme, which exploited the edit distance to quantify keyword similarity. The main drawback of their scheme is the requirement of a predefined dictionary that covers possible keyword misspellings, making update inefficient. Moreover, it permits only single keyword in a query, requiring many rounds to support multi-keyword search. Since then, a few works [7]–[11] have been conducted to improve the search efficiency or to support multi-keyword fuzzy search. However, the research in this field is still in its infancy, and enabling effective fuzzy search in cloud computing remains a challenging problem.

In this paper, we propose a wildcard-based multi-keyword fuzzy search (WMFS) scheme, which allows the users to search on multiple *unsure* keywords in an effective way. In our scheme, the user replaces several unsure letters in a keyword with symbol $'*'$. For example, the user can issue query $Q = (s * cur * ty)$ to retrieve appropriate files if she is unsure of the second and sixth letters of the keyword "security". Different from previous wildcard-based fuzzy search schemes [6]–[8] requiring the predefined dictionary, the main idea of our WMFS scheme is to represent both the query and the index as vectors, the elements of which are set to primes or the reciprocals of primes, ensuring that all reciprocals will be eliminated only when the query matches the index. For data privacy, both vectors will be encrypted with the secure kNN method [12]. Therefore, the level of the match can be quantified by judging whether the inner product of two encrypted vectors is an integer or not.

Specifically, we first construct a basic WMFS scheme to solve single-keyword fuzzy queries. Then, we extend it to the multi-keyword setting, where a query supports either keyword or substring matching. Our proposed WMFS scheme has the following merits: 1) *Efficiency*. It eliminates the requirement of a predefined dictionary and thus supports updates efficiently. 2) *High accuracy*. It eliminates the false positive and false negative introduced by specific data structures (e.g., Bloom filters [13] and locality-sensitive hashing (LSH) [14] utilized in [9], [10]) and thus allows the user to retrieve files as accurate as possible. 3) *Flexibility*. Unlike existing work that is difficult to achieve keyword and substring matching simultaneously, our scheme gives the users great flexibility to specify different search patterns in their queries. The main contributions of this paper are as follows:

- To the best of our knowledge, it is the first attempt to devise a secure and effective fuzzy search scheme, which offers efficiency, high accuracy, and flexibility.
- The proposed scheme utilizes the indecomposable property of primes to support an efficient multi-keyword fuzzy search over the encrypted data.
- We evaluate our scheme on a real data set. The results demonstrate that our scheme is efficient and accurate.

II. Preliminaries

A. System Model

Fig. 1 shows our system model consisting of three types of entities: the cloud service provider (CSP), the data owner, and

2324-9013/18/31.00 ©2018 IEEE
DOI 10.1109/TrustCom/BigDataSE.2018.00192

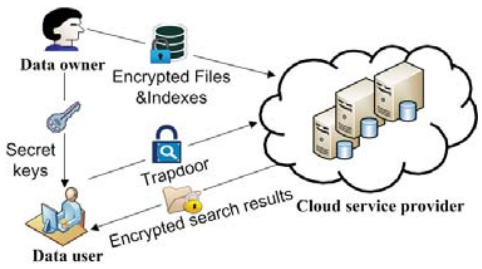

Fig. 1. System model.

the data user (the user for short). The CSP have rich computing resources to provide data storage and query services.

Given a collection of files $\mathcal{D}$ and the universal keyword set $\mathcal{W}$ extracted from $\mathcal{D}$, the data owner first builds a set of indexes $\mathbb{I}$ and uploads files and indexes in the encrypted forms, $\{\mathcal{C}, \mathbb{I}'\}$, to the CSP. To retrieve files matching query $\mathcal{Q}$, the user first requests secret keys SK from the data owner, and then sends a trapdoor, Q', to the CSP. On receiving the search request, the CSP will evaluate the trapdoor on the encrypted indexes and return the search results, $\mathcal{C}_Q$, to the user, who will perform decryption locally to recover file contents.

B. Adversary Model

The data owner and the user are assumed to be fully trusted. The CSP is the potential adversary, which is assumed to be *honest but curious*. That is, the CSP will always correctly execute a given protocol, but may try to learn some additional information about the stored data and the received message. In terms of the information available to the CSP, we mainly consider *the known background model* [10], [12], in which the CSP is able to know additional background information besides the encrypted database, the submitted trapdoors, and the returned search results. The background refers to the keyword frequency and distribution among the file collection, which can be used to infer certain keywords and their trapdoors.

While data privacy can be preserved through standard symmetric key encryption (SKE), e.g., AES, our privacy objective is to protect the search privacy in the following aspects:

- **Keyword secrecy.** The CSP cannot deduce the contents of keywords from the encrypted indexes and trapdoors.
- **Trapdoor unlinkability.** The CSP cannot decide whether two trapdoors are issued for the same query or not.

C. Secure kNN

Given the index vector I and query vector Q, its i-th element, respectively, is denoted as $I[i]$ and $Q[i]$. The secure kNN scheme [12] tailored for our WMFS scheme mainly consists of the following algorithms:

• $Key(1^\kappa) \rightarrow sk$: It takes a security parameter $\kappa \in \mathbb{N}$ as input and generates the secret key $sk = (M_1, M_2, S)$, where M_1, M_2 are $d \times d$ invertible matrices and S is a bit string of d bits.

• $EncI(I, sk) \rightarrow I'$: It splits the index I into two vectors $\{I_a, I_b\}$ as follows: for $i = 1$ to d, if the i-th bit of S equals to 1, $I[i]$ is randomly split into $I_a[i], I_b[i]$ such that $I_a[i]+I_b[i] = I[i]$; if the i-th bit of S equals to 0, both $I_a[i]$ and $I_b[i]$ are set to $I[i]$. The encrypted index is a pair $I' = (I'_a, I'_b)$ where $(I'_a = M_1^T I_a, I'_b = M_2^T I_b)$.

• $EncQ(Q, sk) \rightarrow Q'$: It chooses a random prime r and sets $\widetilde{Q} = rQ$ for the unlinkability of trapdoors. Then, it splits the scaled query $\widetilde{Q}$ into two vectors: $\widetilde{Q}_a$ and $\widetilde{Q}_b$ as follows: for $i = 1$ to d, if the i-th bit of S equals to 0, $\widetilde{Q}[i]$ is split into $\widetilde{Q}_a[i]$ and $\widetilde{Q}_b[i]$ such that $\widetilde{Q}[i] = \widetilde{Q}_a[i] + \widetilde{Q}_b[i]$; if the i-th bit of S equals to 1, both $\widetilde{Q}_a[i]$ and $\widetilde{Q}_b[i]$ are set to $\widetilde{Q}[i]$. The encrypted query is a pair $Q' = (Q'_a, Q'_b)$ where $(Q'_a = M_1^{-1}\widetilde{Q}_a, Q'_b = M_2^{-1}\widetilde{Q}_b)$.

• $Search(I', Q') \rightarrow v$: It calculates $v = I'_a \cdot Q'_a + I'_b \cdot Q'_b = rI^T Q$ as the search result.

III. Scheme Overview

A. Notations and Definitions

The set of all binary strings of length η is denoted as $\{0,1\}^\eta$ and the set of finite binary strings as $\{0,1\}^*$. For quick reference, the most relevant notations are listed below:

- $\mathcal{D} = \{D_1, \ldots, D_n\}$: A collection of n files.
- $\mathcal{C} = \{C_1, \ldots, C_n\}$: A collection of n ciphertexts, where C_i is the ciphertext of file D_i for $i \in [1, n]$.
- $\mathcal{W} = \{w_1, \ldots, w_m\}$: A dictionary of m keywords extracted from $\mathcal{D}$, where $|w_j|$ denotes the number of letters in keyword w_j and $w_j(l)$ denotes the l-th letter in w_j for $1 \le j \le m$ and $1 \le l \le |w_j|$.
- $\mathbb{A} = \{'a', \ldots, 'z'\}$: An English alphabet consisting of 26 English letters, where $\mathbb{A}[i]$ denotes its $i-$th letter.
- I, I': The index vector and its encrypted version, where $I[i]$ and $I'[i]$ denote the $i-$th elements.
- Q, Q': The query vector and the encrypted trapdoor, where $Q[i]$ and $Q'[i]$ denote the $i-$th elements.

Let symbol $'*'$ stand for *unsure*. There are two types of keywords: *exact* keywords and *fuzzy* keywords, where each letter in an exact keyword is chosen from $\mathbb{A}$, but a fuzzy keyword contains symbol $'*'$. The searchable index contains only exact keywords, but the query may contain fuzzy keywords. The distance between keywords w_1 and w_2, denoted as $dist(w_1, w_2)$, is determined by their edit distance, which excludes the number of symbol $'*'$s. Keywords w_1 and w_2 are considered *similar* if their distance is 0.

Matching. *Let K denote the number of keywords in query $\mathcal{Q}$. $\mathcal{Q} \bowtie \mathcal{I}$ if $dist(\mathcal{I}[i], \mathcal{Q}[i]) = 0$ for $i \in [1, K]$, where $\mathcal{I}[i]$ and $\mathcal{Q}[i]$ denote the $i-$th keyword in $\mathcal{I}$ and $\mathcal{Q}$, respectively.*

That is, for AND queries, $\mathcal{Q} \bowtie \mathcal{I}$ if all keywords in $\mathcal{Q}$ are similar to corresponding keywords in $\mathcal{I}$.

B. The Definition of WMFS

A WMFS scheme is a protocol among the data owner, the user, and the CSP as follows:

• $GenKey(1^\kappa) \rightarrow SK$: The data owner takes the security parameter κ as the input and outputs the secret keys SK, which will be sent to the authorized user.

• $BuildIndex(\mathcal{D}, \mathcal{W}, SK) \rightarrow \mathbb{I}$: Given a collection of files $\mathcal{D}$ and the universal keyword set $\mathcal{W}$, the data owner builds a set of indexes $\mathbb{I}$ with her secret keys SK.

• $EncIndex(\mathbb{I}, SK) \rightarrow \mathbb{I}'$: Given a set of indexes $\mathbb{I}$, the data owner takes the secret keys SK as inputs and outputs the encrypted indexes $\mathbb{I}'$.

• $BuildQuery(\mathcal{Q}, SK) \rightarrow Q$: Given a query $\mathcal{Q}$ consisting of a set of keywords, the user builds a query vector Q based on the secret keys SK.

• $EncQuery(Q, SK) \rightarrow Q'$: To retrieve files matching query vector Q, the user generates a trapdoor Q' with the secret keys SK and sends Q' to the CSP.

• $Search(\mathbb{I}', Q') \rightarrow \mathcal{C}_Q$: The CSP evaluates the trapdoor Q' on the encrypted indexes $\mathbb{I}'$ to output the search results $\mathcal{C}_Q$.

IV. Basic WMFS Scheme

A. Construction

Let $L = \max(|w_1|, \ldots, |w_m|)$ denote the max length of the keywords. To hide the length of each keyword, we pad w_i with *dummy letters* such that $|w_i| = L$ for $1 \leq i \leq m$.

Let $KNN = (Key, EncI, EncQ, Search)$ be a secure kNN scheme as described in Section II-C, and let $SKE = (Gen, Enc, Dec)$ be a symmetric-key encryption scheme. Let $F : \{0,1\}^\kappa \times \{0,1\}^* \rightarrow \{0,1\}^\kappa$ be a PRF. Our basic WMFS scheme is constructed as follows:

• $GenKey(1^\kappa) \rightarrow SK$: Given a security parameter κ, the data owner runs $KNN.Key(1^\kappa)$ algorithm to generate sk Then, she randomly chooses L primes $\mathcal{P} = \{p_1, \ldots, p_L\}$, L random strings $\mathcal{S} = \{s_1, \ldots, s_L\}$, and a κ-bit string k_f. The secret keys are set as $SK = (sk, k_f, L, \mathcal{P}, \mathcal{S})$.

• $BuildIndex(\mathcal{D}, \mathcal{W}, SK) \rightarrow \mathbb{I}$: To build a searchable index $\mathcal{I}_j$ for keyword $w_j \in \mathcal{W}$, the data owner first constructs a $d-$dimensional vector, I_j, where each element is initialized with 1. For $1 \leq l \leq L$, she calculates:

$$pos_l = \begin{cases} F_{k_f}(w_j(l)), & if \quad l \in [1, |w_j|] \\ F_{k_f}(\mathcal{S}[l - |w_j|]), & if \quad l \in (|w_j|, L] \end{cases} \tag{1}$$

and sets $I_j[pos_l] = I_j[pos_l]/p_l$. The other elements will be filled with random integers. Next, she constructs an $m \times n$ binary matrix E such that:

$$E[j][i] = \begin{cases} 1 & if\, w_j\, is\, contained\, in\, D_i \\ 0 & otherwise \end{cases}$$

where $E[j][i]$ is the element in the j-th row and i-th column of E for $j \in [1, m]$ and $i \in [1, n]$. The set of indexes are set as $\mathbb{I} = (\{I_j | w_j \in \mathcal{W}\}, E)$.

• $EncIndex(\mathbb{I}, SK) \rightarrow \mathbb{I}'$: To encrypt the searchable index $\mathcal{I}_j$, the data owner runs the $KNN.EncI(I_j, sk)$ algorithm and outputs I'_j for each keyword $w_j \in \mathcal{W}$. To encrypt the binary matrix E, she runs $SKE.Enc(k_e, E[j])$ and outputs $E'[j]$ for $1 \leq j \leq m$, where $E[j]$ denotes the bit string at the j-th row of E. The encrypted indexes are set as $\mathbb{I}' = (\{I'_j | w_j \in \mathcal{W}\}, E')$.

• $BuildQuery(\mathcal{Q}, SK) \rightarrow Q$: Given a query $\mathcal{Q}$ consisting of keyword $w_j \in \mathcal{W}$, the user first constructs a $d-$dimensional vector Q, where each element is initialized to 1. For $1 \leq l \leq L$, he calculates pos_l with Eq. 1 and sets $Q[pos_l] = Q[pos_l] \times p_l$ if $w_j(l) \neq' *'$; otherwise, he calculates $pos_{l1} = F_{k_f}('a'), \ldots, pos_{l26} = F_{k_f}('z')$ and sets $Q[pos_{li}] = Q[pos_{li}] \times p_l$ for $1 \leq i \leq 26$.

• $EncQuery(Q, SK) \rightarrow Q'$: To generate a trapdoor, the user runs the $KNN.EncQ(sk, Q)$ algorithm and outputs Q'.

• $Search(\mathbb{I}', Q') \rightarrow \mathcal{C}_Q$: For $1 \leq j \leq m$, the CSP runs the $KNN.Search(I'_j, Q')$ algorithm to calculate the inner product of I'_j and Q'. If the result of $I'_j \cdot Q'$ is an integer, the CSP puts $E'[j]$ in $\mathcal{C}_Q$.

B. Correctness Analysis

Let $\mathbb{U} = \mathbb{A} \cup \mathcal{S}$ denote the union of the English alphabet and the dummy letters, where $\mathbb{U}[i]$ denotes the i-th letter in $\mathbb{U}$. For ease of illustration, we assume that $\mathbb{U}[i]$ corresponds to $I[i]$ and $Q[i]$. Our basic WMFS scheme is considered incorrect if the following cases happen:

Case 1. The result of $I \cdot Q$ is not an integer if query $\mathcal{Q}$ matches index $\mathcal{I}$.

Case 2. The result of $I \cdot Q$ is an integer if query $\mathcal{Q}$ mismatches index $\mathcal{I}$.

For case 1, the result of $I \cdot Q$ not being an integer means that at least one reciprocal in the index vector cannot be eliminated. Due to the construction of the $BuildIndex$ algorithm, $I[i] = 1/p_j$ means that $\mathbb{U}[i]$ is the j-th letter of keyword $w_1 \in \mathcal{I}$. If $1/p_j$ cannot be eliminated, $Q[i]$ will be set to an integer V that is indivisible by p_j. Here, V may be 1 or the product of the primes in $\mathcal{P}/p_j$. Due to the construction of the $BuildQuery$ algorithm, either $Q[i] = 1$ or $Q[i] = \prod_{l \neq j} p_l$ means that neither $\mathbb{U}[i]$ nor symbol $'*'$ appears in the j-th position of keyword $w_2 \in \mathcal{Q}$. That is, $\mathcal{I} \not\bowtie \mathcal{Q}$, which contradicts the assumption. Therefore, case 1 is not true.

For case 2, the result of $I \cdot Q$ being an integer means that all reciprocals in the index vector are eliminated. Given $I[i] = 1/p_j$ denoting that $\mathbb{U}[i]$ is the j-th letter of keyword $w_1 \in \mathcal{I}$, $Q[i]$ needs to be set to p_j or an integer V that is divisible to p_j. Due to the construction of the $BuildQuery$ algorithm, $Q[i] = p_j$ means that $\mathbb{U}[i]$ is in the j-th position of keyword $w_2 \in \mathcal{Q}$ and $Q[i] = V$ means that symbol $'*'$ is at the j-th position of keyword $w_2 \in \mathcal{Q}$. In any case, the index matches the query, which contradicts the assumption. Therefore, case 2 is not true and our basic WMFS scheme is correct. ■

C. Security Analysis

Theorem 1. *Our basic WMFS construction is secure under the known background model.*

Before stating our security theorem, we provide a more formal and concise description of our scheme's leakage:

• *History* $\mathcal{H} = (\mathcal{D}, \mathcal{W}, \mathbb{Q})$ where $\mathcal{D}$ is a collection of files $\{D_1, \ldots, D_n\}$, $\mathcal{W}$ is a set of keywords $\{w_1, \ldots, w_m\}$, and $\mathbb{Q}$ is a sequence of submitted queries $(\mathcal{Q}_1, \ldots, \mathcal{Q}_k)$.

• *View* $\mathcal{V} = (\mathcal{C}, \mathbb{I}', \mathbb{T})$ is the encrypted form of history under the secret keys SK. That is, $\mathcal{C} = \{C_1, \ldots, C_n\}$ is a set of ciphertexts (where C_i is the ciphertext of file $D_i \in \mathcal{D}$), $\mathbb{I}' = \{I'_1, \ldots, I'_m\}$ is a set of encrypted searchable indexes (where I'_i is built for keyword $w_i \in \mathcal{W}$), and $\mathbb{T} = (Q'_1 \ldots, Q'_k)$ is a

sequence of trapdoors (where Q'_i is the trapdoor created for query $\mathcal{Q}_i \in \mathbb{Q}$). The CSP can only see views.

• *Trace of history* $Tr(\mathcal{H}) = \{Tr(\mathcal{Q}_1), \ldots, Tr(\mathcal{Q}_k)\}$ where $Tr(\mathcal{Q}_i) = \{(I'_j, v_{ji}) | \mathcal{I}_j \bowtie \mathcal{Q}_i, 1 \leq j \leq m\}$ and v_{ji}, as the inner product of index vector I_j and query vector Q_i, is a random integer. The trace of a history captures the information which can be learned by the CSP, i.e., the access pattern and the search results induced by $\mathcal{H}$.

Proof sketch. Due to the CPA-security of SKE, no PPT adversary can distinguish between $\bar{\mathcal{C}}$ and $\mathcal{C}$. The indistinguishability of indexes and trapdoors is based on the indistinguishability of KNN and the introduced randomness. The encrypted indexes $\bar{\mathbb{I}}'$ and the trapdoors $\bar{\mathbb{T}}$ generate the same trace as the one that the CSP has. Therefore, we claim that no PPT adversary can distinguish $\bar{\mathcal{V}}$ from $\mathcal{V}$. As described in [19], KNN can resist ciphertext-only attacks (in which the CSP only knows encrypted databases and trapdoors), but is vulnerable to linear analyses (in which the CSP can solve linear equations to recover index vectors with sufficient query vector and trapdoor pairs). However, a set of random primes and a PRF are introduced to build vectors in our construction. Therefore, it is hard for the adversary to obtain plaintext vectors to initiate chosen-plaintext attacks. For example, if all primes are randomly chosen from $[1, 10000]$, then the total number of primes that can be used is $l = 1231$. Without knowing $\mathcal{P}$, there are $C(l, L)$ possible ways to construct these primes. For $l = 1231$ and $L = 30$, the adversary needs to guess about 2^{199} for brute force attacks. In terms of strength, this is more powerful than 128-bit symmetric keys. Intuitively, our scheme encodes keywords to vectors, which can be regarded as an encapsulation of keywords before KNN encryption. Hence, the adversary with (keyword, trapdoor) pairs cannot distinguish the output of the linear analyses from a random string without knowing the secret keys. ■

V. Advanced WMFS Scheme

A. Construction

Let $H : \{0,1\}^* \to \{0,1\}^\kappa$ be a collision-free hash function and let L denote the max length of keywords. Let $\mathcal{I}[j]$ and $\mathcal{Q}[j]$ denote the $j-$th keyword in index $\mathcal{I}$ and query $\mathcal{Q}$, respectively, where $|\mathcal{I}[j]|$ (resp. $|\mathcal{Q}[j]|$) denotes the length of a keyword and $\mathcal{I}[j](l)$ (resp. $\mathcal{Q}[j](l)$) denotes the $l-$th letter in a keyword.

• $GenKey(1^\kappa) \to SK$: The data owner sets $SK = \{sk, k_e, L, \mathcal{P}, \mathcal{S}\}$ as the basic scheme.

• $BuildIndex(\mathcal{D}, \mathcal{W}, SK) \to \mathbb{I}$: For each file $D_i \in \mathcal{D}$ the data owner builds a d-dimensional vector I_i, where each element is initialized to 1. For $\mathcal{I}_i[j]$, she performs as follows. For $l \in [1, L]$, she calculates:

$$pos_l = \begin{cases} H(j, \mathcal{I}_i[j](l)), & if \quad l \in [1, |\mathcal{I}_i[j]|] \\ H(j, \mathcal{S}[l - |\mathcal{I}_i[j]|]), & if \quad l \in (|\mathcal{I}_i[j]|, L] \end{cases} \tag{2}$$

and sets $I_j[pos_l] = I_j[pos_l] \times p_l$. Thus, $\mathbb{I} = \{I_1, \ldots, I_n\}$.

• $EncIndex(\mathbb{I}, SK) \to \mathbb{I}'$: The data owner runs the $KNN.EncI(I_i, sk)$ algorithm and outputs I'_i for each document $D_i \in \mathcal{D}$. She then sets $\mathbb{I}' = \{I'_i | D_i \in \mathcal{D}\}$.

• $BuildQuery(\mathcal{Q}, SK) \to Q$: The user first constructs a $d-$dimensional vector Q, where each element is initialized with 1. For $\mathcal{Q}[j]$, the user performs as follows. For $1 \leq l \leq L$, if $\mathcal{Q}[j](l) \neq' *'$, he calculates:

$$pos_l = \begin{cases} H(j, \mathcal{Q}[j](l)), & if \quad l \in [1, |\mathcal{Q}[j]|] \\ H(j, \mathcal{S}[l - |\mathcal{Q}[j]|]), & if \quad l \in (|\mathcal{Q}[j]|, L] \end{cases} \tag{3}$$

and sets $Q[pos_l] = Q[pos_l] \times 1/p_l$. The other elements will be filled with random integers.

• $EncQuery(Q, SK) \to Q'$: The user runs the $KNN.EncQ(Q, sk)$ algorithm to output a trapdoor Q'.

• $Search(\mathbb{I}', Q') \to \mathcal{C}_Q$: For $1 \leq i \leq n$, the CSP runs the $KNN.Search(I'_i, Q')$ algorithm to calculate the inner product of I'_i and Q'. If the output is an integer, it puts C_i in $\mathcal{C}_Q$.

Remark 1. The advanced WMFS scheme can support substring matching if the users never pad their queries to length L. Therefore, our WMFS scheme gives the user great flexibility to specify different search patterns in their queries.

Remark 2. In the advanced scheme, if reciprocal $1/p_l$ located at pos_l of Q cannot be eliminated, corresponding element $I[pos_l]$ is set to 1. To add more randomness to the search results, $I[pos_l]$ can be set to a random integer that cannot be decomposed to any prime in $\mathcal{P}$.

B. Correctness Analysis

The advanced WMFS scheme is considered incorrect if either case 1 or case 2 defined in Section IV-B happens. For case 1, the result of $I \cdot Q$ not being an integer means that at least one reciprocal in the query vector Q cannot be eliminated. Suppose that the reciprocal $1/p_l$ located at pos_l of Q cannot be eliminated, i.e., corresponding element $I[pos_l]$ is set to 1. Due to the construction of the $BuildIndex$ and $BuildQuery$ algorithms, the $l-$th letter of the $j-$th keyword in the index is different from that in the query, we have $Q \not\bowtie I$ and it contradicts the assumption. Therefore, case 1 will not happen.

For case 2, the result of $I \cdot Q$ being an integer means that all reciprocals in the query vector Q are eliminated. Suppose that $\mathcal{Q}[j](l)$ is dissimilar to $\mathcal{I}[j](l)$. According to the construction of the $BuildIndex$ and $BuildQuery$ algorithms, $pos_l = H(j, \mathcal{Q}[j](l))$ in Q is set $1/p_l$ and $pos_l = H(j, \mathcal{I}[j](l))$ in I is set to p_l. However, due to the collision-free property of the hash function H, the probability of mapping two different inputs to the same location is very low. Therefore, case 2 will not happen and our advanced scheme is correct. ■

C. Security Analysis

Theorem 2. *Our advanced WMFS scheme is secure under the known background model.*

The definitions of history, views, and traces are the same as those in the basic scheme. The security of our advanced scheme can also be proven in a simulation-based approach. Since the search result on $\bar{\mathbb{I}}'$ with trapdoor $\bar{\mathbb{Q}}'$ generates the same trace as the one that the CSP has, we claim that no PPT adversary can distinguish $\bar{\mathcal{V}}$ from $\mathcal{V}$. Similarly, the PPT adversary cannot distinguish the output of the linear analysis from a random string without knowing the primes. ■

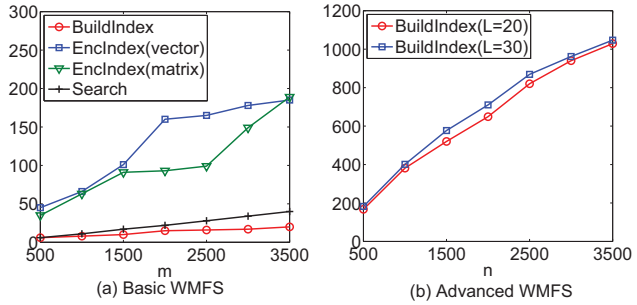

Fig. 2. The execution time (ms). (a) The performance of our basic scheme while the number of keywords m ranges from 500 to 3500; (b) The time of our advanced scheme for building a set of indexes for $n = [500, 3500]$ files.

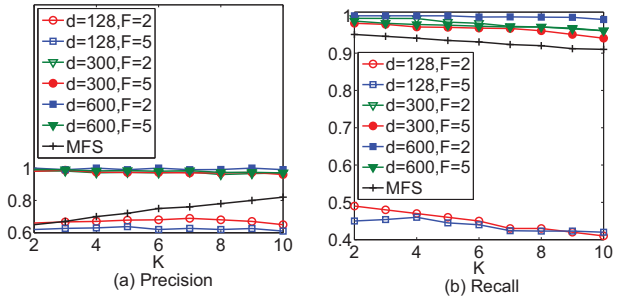

Fig. 4. The accuracy of our advanced WMFS scheme. The number of keywords in a query K ranges from 2 to 10.

VI. Evaluation

We will compare the performance of our WMFS scheme with the multi-keyword fuzzy search (MFS) scheme proposed in [10] in terms of the execution time and the result accuracy. The MFS scheme utilizes Bloom filters and a 2-stable $(\sqrt{3}, 2, p_1, p_2)$-LSH family to build indexes, where the size of vectors is set to $d = 8000$ and $l = 30$ hash functions are employed to support 1 edit distance difference.

A. Parameter Setting

Experiments are conducted on a local machine running the Microsoft Windows 7 Ultimate operating system with an Intel Core i5 CPU running at 2.6GHz and 8GB memory. The programs are implemented in Java, compiled using Eclipse 4.3.2. We apply HMAC-SHA1 as the collision-free hash function and employ the block cipher (AES) for file encryption.

We conduct a performance evaluation on the recent 10 years' IEEE INFOCOM publication, which includes more than 3600 files. The number of files is set to $n = [500, 3500]$. For each file, we extract $N = [5, 20]$ keywords to build its index, where the maximal length of keyword L is set to $[20, 30]$. The universal keyword set contains $m = [500, 3500]$ keywords. In advanced scheme, each user will query with $K = [2, 25]$ keywords, where each query contains $F = [2, 5]$ symbol $'*'$s. To generate a fuzzy keyword in a query, we randomly choose one letter from a keyword and replace it with symbol $'*'$. The size of vectors is set to $d = 128$ and $d = [128, 300, 600]$ in the basic and advanced WMFS scheme, respectively.

B. Experiment Results

1) Efficiency. Fig. 2-(a) shows the execution time of the basic WMFS scheme, while m ranges from 500 to 3500 under the setting of $L = 30$ and $L = 20$ results in the similar trend. Fig. 2-(b) shows the execution time of the advanced $BuildIndex$ algorithm, while n ranges from 500 to 3500 under the setting of $N = 20$. Fig. 3 shows the comparison results between our advanced scheme and the MFS scheme. For Figs. 3-(b)$\sim$(d), we fix with $N = 10$ and $L = 30$. The MFS scheme requires the computation of multiple LSHs to map elements to a Bloom filter of $d = 8000$ elements. Our advanced scheme requires only the multiplication and hash operations and the size of vector is much smaller, making it more efficient in general.

2) Accuracy. As the work in [10], the accuracy of our scheme is measured by the definitions of the widely used performance metrics: *precision* and *recall*. Let t_p denote true positive, f_p as false positive, and f_n as false negative. Here, t_p is $\mathcal{I} \bowtie \mathcal{Q}$ and the result of $I \cdot Q$ is an integer, f_p is $\mathcal{I} \not\bowtie \mathcal{Q}$ but the result of $I \cdot Q$ is an integer, and f_n is $\mathcal{I} \bowtie \mathcal{Q}$, but the result of $I \cdot Q$ is not an integer. The precision can be calculated with $t_p/(t_p + f_p)$ and the recall can be calculated with $t_p/(t_p + f_n)$.

The division operation may result in a loss of precision. To determine if a value is an integer or not, we round up after the $x-$th decimal point. When $x = 7$,the basic scheme to have the highest accuracy. The reason is that a large part fractional numbers, e.g., 3.000459, will be determined as integers if x is a small value, making a higher false positive. When x is a large value, a lot of values, e.g., 3.000000007, will be determined as fractional numbers, making the false negative relatively higher. Furthermore, F has a great impact on f_p when x is a small value, and on f_n as x increases.

With the fixed $x = 7$, we then evaluate the accuracy of our advanced scheme under the different settings of F and d. From the experiment results, we know that d plays an important role on the accuracy. The collision rate of the hash function is relatively higher when the size of vectors $d = 128$. Therefore, in Fig. 4, the accuracy of our scheme is worse than the MFS scheme under $d = 128$. However, while d increases to 300, either our precision or recall is better than their scheme.

VII. Related Work

The first SE scheme where both queries and data were encrypted under a symmetric key was proposed by Song et al. [15]. The main drawback of their scheme was that the search cost grew linearly with the database. To improve the query efficiency, Goh [3] developed a secure searchable index scheme based on Bloom filters. As a seminal work in SE, Curtmola et al. [5] provided a rigorous security definition and constructed schemes based on an inverted index. Recently, Kurosawa et al. [21] constructed verifiable SE schemes, in which a user could detect any cheating behavior from malicious servers. As an attempt to enrich search predicates,

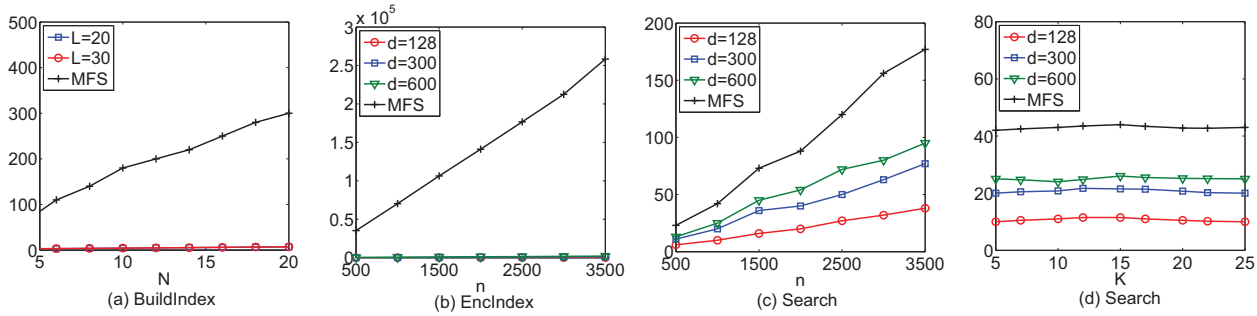

Fig. 3. The comparison of execution time (ms). (a) The time for building a single index, each containing N keywords; (b) The time for encryption n indexes; (c) The time for searching n files with fixed query keywords $K = 20$; (d) The time for searching K keywords with the fixed file size $n = 1000$.

searchable encryption schemes that support conjunctive keyword search [4], subset query [16], and range query [17], have also been proposed. In an attempt to optimize the search results, ranked SE schemes [18] have also been proposed to allow users to retrieve the best-matched files. However, all of these works only supported exact keyword search.

To improve search experiences, Li et al. [6] proposed the first wildcard-based fuzzy search scheme, which tolerated keyword misspellings in the query. However, their scheme required a predefined dictionary and supported only single-keyword searches. Since then, a few works have been conducted in this field. For example, Chuah et al [7] proposed a bedtree-based fuzzy search scheme to enable efficient updates. Liu et al. [8] improved the scheme by reducing the index size. In [9], LSH functions were used to generate file index. Inspired by their work, Wang et al. [10] proposed a multi-keyword fuzzy search scheme which supported the constant size indexes. However, the combined effect of false positives (introduced by Bloom filters) and false negatives (introduced by LSH) seriously impacted the accuracy. Boldyreva et al. [20] improved the security of existing fuzzy search schemes based on closeness graphs. To enrich the search patterns, Wang et al. [11] proposed a scheme for a generalized pattern-matching string-search. All of the aforementioned fuzzy search schemes have only partially addressed our design goals.

VIII. Conclusion

In this paper, we propose a WMFS scheme to achieve secure and effective search services in cloud computing. The proposed scheme supports an efficient multi-keyword fuzzy search over the encrypted data by exploiting the indecomposable property of primes. Experiment results demonstrate that our scheme is efficient and accurate. However, our scheme requires an order among the keywords in the multi-keyword setting. Therefore, as part of our future work, we will try to design an improved scheme supporting unordered matching.

Acknowledgment

This work was supported in part by NSFC grants 61632009; Key Lab of Information Network Security, Ministry of Public Security; and NSF grants CNS 1757533, CNS1629746, CNS 1564128, CNS 1449860, CNS 1461932, CNS 1460971, and IIP 1439672.

References

[1] K. Xie, X. Li, X. Wang, G. Xie, J. Wen, J. Cao, and D. Zhang, "Fast tensor factorization for accurate internet anomaly detection," *IEEE/ACM Transactions on Networking*, 2017.
[2] K. Xie, X. Li, X. Wang, J. Cao, G. Xie, J. Wen, D. Zhang, and Z. Qin, "On-line Anomaly Detection with High Accuracy," *IEEE/ACM Transactions on Networking*, 2018.
[3] E.-J. Goh, "Secure indexes," *IACR Cryptology ePrint Archive*, 2003.
[4] P. Golle, J. Staddon, and B. Waters, "Secure conjunctive keyword search over encrypted data," in *Proc. of ACNS*, 2004.
[5] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proc. of CCS*, 2006.
[6] 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 INFOCOM*, 2010.
[7] M. Chuah and W. Hu, "Privacy-aware bedtree based solution for fuzzy multi-keyword search over encrypted data," in *Proc. of ICDCS Workshops*, 2011.
[8] C. Liu, L. Zhu, L. Li, and Y. Tan, "Fuzzy keyword search on encrypted cloud storage data with small index," in *Proc. of CCIS*, 2011.
[9] M. Kuzu, M. S. Islam, and M. Kantarcioglu, "Efficient similarity search over encrypted data," in *Proc. of ICDE*, 2012.
[10] B. Wang, S. Yu, W. Lou, and Y. T. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *Proc. of INFOCOM*, 2014.
[11] 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 *Proc. of INFOCOM*, 2015.
[12] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis, "Secure knn computation on encrypted databases," in *Proc. of ACM SIGMOD*, 2009.
[13] D. Guo, J. Wu, H. Chen, Y. Yuan, and X. Luo, "The dynamic bloom filters," *IEEE Transactions on Knowledge and Data Engineering*, 2010.
[14] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc. of SCG*, 2004.
[15] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. of S&P*, 2000.
[16] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proc. of TCC*, 2007.
[17] E. Shi, J. Bethencourt, T.-H. Chan, D. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *Proc. of S&P*, 2007.
[18] Q. Liu, X. Nie, X. Liu, T. Peng, and J. Wu, "Verifiable ranked search over dynamic encrypted data in cloud computing," *Proc. of IWQoS*, 2017.
[19] B. Yao, F. Li, and X. Xiao, "Secure nearest neighbor revisited," in *Proc. of ICDE*, 2013.
[20] A. Boldyreva and N. Chenette, "Efficient fuzzy search on encrypted data," in *International Workshop on Fast Software Encryption*. Springer, 2014, pp. 613–633.
[21] K. Kurosawa and Y. Ohtaki, "How to update documents verifiably in searchable symmetric encryption," in *Proc. of CNS*, 2013.