Privacy-Preserving Multikeyword Similarity Search Over Outsourced Cloud Data

Chia-Mu Yu, Chi-Yuan Chen, and Han-Chieh Chao, *Senior Member, IEEE*

***Abstract*—The amount of data generated by individuals and enterprises is rapidly increasing. With the emerging cloud computing paradigm, the data and corresponding complex management tasks can be outsourced to the cloud for the management flexibility and cost savings. Unfortunately, as the data could be sensitive, the direct data outsourcing would have the problem of privacy leakage. The encryption can be used, before the data outsourcing, with the concern that the operations can still be accomplished by the cloud. We consider the multikeyword similarity search over outsourced cloud data. In particular, with the consideration of the text data only, multiple keywords are specified by the user. The cloud returns the files containing more than a threshold number of input keywords or similar keywords, where the similarity here is defined according to the edit distance metric. We propose three solutions, where blind signature provides the user access privacy, and a novel use of Bloom filter's bit pattern provides the speedup of search task at the cloud side. Our final design to achieve the search is secure against insider threats and efficient in terms of the search time at the cloud side. Performance evaluation and analysis are used to demonstrate the practicality of our proposed solutions.**

***Index Terms*—Cloud computing, counterintelligence, outsourced data, privacy, similarity search.**

I. Introduction

CLOUD computing has become a new computing paradigm. Currently, an increasing number of individuals and enterprises are generating a huge amount of data everyday. It is no longer economically feasible to maintain their own hardware and staffs for data management. Recently, a reasonable and popular choice to mitigate the burden of data management is to outsource the complex data management task to the cloud with the major benefit of cost savings. One may have concern that the cloud cannot always be trusted; it may purposely and unsolicitedly examine the outsourced data [1]. To keep the advantage of cost savings and protect the data privacy, the data, before outsourced to the cloud, need to be encrypted. Despite the success in gaining the data privacy, data encryption does not allow the cloud to answer the users' queries on the data. A straightforward solution for the user to overcome such a difficulty is to simply download the entire data set. This, however, is practically infeasible because of the huge volume of the incurred bandwidth consumption.

The problem of retrieving information from the encrypted files has already been very challenging. In the context of outsourced cloud data, the problem is even aggravated by a large number of on-demand users and a huge amount of outsourced data files. Therefore, with the given considerations, it is extremely difficult to meet the requirements of both retrieval feasibility and system performance.

In the current use, text data that can be seen everywhere would be the one delivering the majority of information. An important method of retrieving information on the text data is the keyword search, in which only the text files containing the specific keywords are returned to the user. However, possibly due to the lack of the files perfectly matching the input keywords, the keyword search may return an empty result. In this sense, the user naturally turns to seek for the similar result. Here, the similar result could be the files containing part of input keywords or containing the words similar to the input keywords. Such similar keyword search can find numerous applications, such as record linkage [21] and biological database [32]. Due to its ability in enhancing system usability and overall user experience, the research on the similar keyword search has been conducted extensively. Unfortunately, most of them are considered without security and privacy concerns [18], [23]. Still, only a few research efforts on similar keyword search are done under the constraint of encrypted texts.

A. Related Work

Searchable encryption [7], [12]–[14], [22], [25], [27] is a cryptographic primitive developed for performing the keyword search over the encrypted data. The basic idea behind the very first searchable encryption is to encrypt each word in a text file individually. Then, as can easily be known, the search cost of the given basic searchable encryption would be very high. The subsequent research efforts are put to develop an index that can support more efficient keyword search. Another line of

Manuscript received August 24, 2014; revised November 22, 2014; accepted January 24, 2015. This work was supported in part by the Ministry of Science and Technology of Taiwan under Grant MOST 102-2218-E-155-006-MY2, Grant 103-2221-E-197-018, and Grant 101-2221-E-197-008-MY3; by the National Nature Science Foundation of China under Grant 61170296 and Grant 60873241; and by the Program for New Century Excellent Talents in University under Grant 291184.

C.-M. Yu is with the Department of Computer Science and Engineering, Yuan Ze University, Taoyuan 32003, Taiwan, and also with Innovation Center for Big Data and Digital Convergence, Taoyuan 320, Taiwan (e-mail: chiamuyu@saturn.yzu.edu.tw).

C.-Y. Chen is with the Department of Computer Science and Information Engineering, National Ilan University, Yilan 260, Taiwan (e-mail: chiyuan.chen@ieee.org).

H.-C. Chao is with the Department of Electronic Engineering and the Institute of Computer Science and Information Engineering, National Ilan University, Yilan 260, Taiwan, with the Department of Electrical Engineering, National Dong Hwa University, Hualien 97401, Taiwan, with the College of Information Engineering, Yangzhou University, Yangzhou 225009, China, and also with the School of information Science and Engineering, Fujian University of Technology, Fuzhou 100044, China (e-mail: hcc@niu.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSYST.2015.2402437

research on searchable encryption is to enrich search predicates; therefore, conjunctive keyword search, subset query, and range query over encrypted data are also introduced. The advantage of searchable encryption is its provable security. Nevertheless, although different searchable encryption techniques are proposed, applying searchable encryption directly cannot tackle the similarity keyword search problem because it can only handle the exact keyword matching.

Recently, privacy-preserving keyword search [26], [30], such as secure ranked search, where only the files with better matching to the input keywords are returned, has been studied in the settings of single keyword [29] and multikeyword [15]. Nonetheless, in the setting of secure ranked search, only the number of keyword matches is concerned, and the similarity between the input keywords and the actual words in text is not taken into account. On a different front, privacy assured similarity search, where the files containing exactly the same keyword or containing similar keyword are returned, has also been studied. However, in the setting of privacy assured similarity research, only single keyword is allowed, restricting the practical use.

B. Problem Description

In this paper, we focus on privacy-preserving multikeyword similarity search (PPMKSS) over the outsourced cloud data. In the PPMKSS over the outsourced cloud data, the data are encrypted and then outsourced to the cloud. The user, after gaining the authorization, sends keywords to the cloud, which returns to the user the files containing as many keywords or their variants as possible. A more mathematical definition of multikeyword similarity search can be found in Section III-C.

Our problem is considered more difficult than the one in [15] and [29] because, in our problem, the keywords found in the returned result is allowed to be different from the input keywords. On the other hand, our problem is considered more challenging than the one in [31] because multiple input keywords are taken into account. Multikeyword similarity search, as stated in Section I, can find numerous applications.

A concrete example of multikeyword similarity search is that, when several keywords are input to Google, although some of them are misspelled, the webpages containing as many keywords closest to the input keywords as possible will be returned.

C. Contribution

Our contribution can be summarized as follows.

1) PPMKSS over the outsourced cloud data is considered for the first time in the literature.
2) We propose a user authorization scheme with the guarantee of user access privacy by using blind signature.
3) By taking advantage of keyword suppressing technique and the Bloom filter, we propose three PPMKSS solutions, namely, PPMKSS-1, PPMKSS-2, and PPMKSS-3, to achieve PPMKSS. The first one could be thought of as a natural extension of the method in [31]. In PPMKSS-2, we use Bloom filter as a tool to perform the private set intersection with the attempt to find out the satisfying files. PPMKSS-3 can be regarded as a hybrid method combining the advantages of PPMKSS-1 and PPMKSS-2. In particular, the last proposed solution is highly efficient in terms of the incurred storage and computation cost at the cloud side.
4) Simulation result and analysis are conducted to evaluate the proposed methods and guarantee the performance.

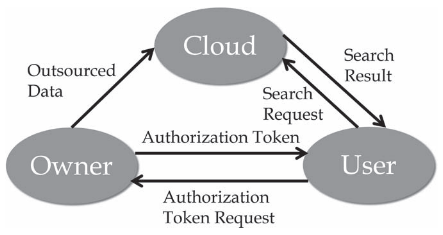


Fig. 1. Conceptual illustration of PPMKSS.

II. System Model

A. Notations

The system model considered in this paper is shown in Fig. 1, where the owner owns the data. In this paper, this amounts to the owner having a collection $C = \{f_1, \ldots, f_n\}$ of text files. A predefined set $W = \{w_1, \ldots, w_p\}$ of distinct keywords in C can be used to retrieve particular files. The owner outsources the data C to the cloud. However, to keep the data private, the data will be encrypted before outsourced to the cloud. Since the data could be valuable to the user, the user may purchase the authorization token to perform the multikeyword similarity search to access the data of interest. To achieve this goal, before returning the search result, the cloud checks whether the user has specific token to make sure the user is authorized to perform the search. An authorized user submits q keywords $X = \{x_1, \ldots, x_q\}$ and two parameters $\delta \leq q$ and d to the cloud. The parameter δ is used to specify the minimum number of input keywords or their variants needed to be contained in the returned files, whereas the parameter d is used to specify the extent to which the input keywords are allowed to vary. The cloud performs similarity search over the encrypted data C and sends back all the set C_X of encrypted files that fulfill the requirement of similarity keyword search on input keywords X. A more formal definition of the problem of PPMKSS over outsourced cloud data can be found in Section III-C.

B. Insider Threat

We consider an insider threat from the cloud. In particular, the cloud management or the staffs may have the privacy breach by checking the file contents stored in the cloud. In essence, a cloud with such an insider threat can be *honest but curious*. By honest, we mean that the cloud faithfully follows the designed protocols to interact with both the owner and user. For example, once the cloud finds C_X, it faithfully returns C_X to the user. By curious, we mean that the cloud will look into each incoming message, trying to gain additional information. The honest-but-curious assumption is contrast to the so-called malicious cloud, which might returns part of results only, or in general, might not follow the designed protocol. In this paper, we only focus on how the cloud can respond to the similarity search request over the encrypted data. The consideration of honest-but-curious cloud is consistent with most of the searchable encryption research. Therefore, we only assume honest-but-curious cloud in this paper.

C. Design Goals

The following design goals will be achieved with the consideration of insider threats from the cloud.

1) **Multikeyword similarity search**: all of the files containing at least a threshold number of keywords similar to the input keyword specified by the user will be returned to the user.
2) **Data privacy**: The cloud cannot learn additional information from the outsourced encrypted data and the corresponding index.
3) **Keyword-file association privacy**: The cloud learns nothing about the association between files and keywords.
4) **User access privacy**: Both the owner and cloud cannot know which user is performing the multikeyword similarity search.
5) **File access privacy**: Both the owner and cloud cannot know which file is accessed more frequently.
6) **Keyword access privacy**: Both the owner and cloud cannot know which keyword is queried more frequently.

The given privacy guarantees are used to defend against the insider threats (see Section II-B). In particular, since the insider is able to have access to every file content, the data privacy via data encryption guarantees no file content is leaked. The use of crytpographic hashed inverted list and blind signature (described later) also ensure the keyword–file, user access, file access, and keyword access privacy.

III. Background Knowledge

Here, some of the background knowledge used in this paper are introduced. First, since the similarity between two words are frequently used in identifying the similar words, it is essential to have a quantitative metric to measure the similarity. For the text data, edit distance is a natural metric to measure the similarity. The notion of edit distance will be presented in Section III-A. In our proposed solutions, the Bloom filter is often used as a means to reduce the storage cost and gain the access speedup at the cloud side. Therefore, the Bloom filter is also briefly described in Section III-B. Finally, the formal definition of PPMKSS over outsourced cloud data is shown in Section III-C.

A. Edit Distance

Edit distance is a quantitative metric to measure the similarity between two strings [2]. The edit distance $\mathrm{ed}(w_1, w_2)$ between two words is defined as the minimum number of operations required to transform from one word to another. The operations considered here are character insertion, deletion, and replacement. Given a keyword w, the notation $S_{w,d}$ is used to represent the set of all possible keyword variants, each of which has edit distance less than or equal to d. In other words, for each $w' \in S_{w,d}$, $\mathrm{ed}(w, w') \leq d$. We also note that the edit distance is always integer valued simply because of its definition.

B. Bloom Filter

As a probabilistic data structure, a Bloom filter consists of an array of b bits. Together with k independently and randomly selected hash functions, $h_1, \ldots, h_k$, with range $[0, b-1]$, it is used to represent a set of elements with the support of membership query. Assume that a Bloom filter B is used to represent a set $S = \{s_1, \ldots, s_m\}$ of m elements. To insert an element s_i, the bits $B[h_j(s_i)]$ for $1 \leq j \leq k$ are set to 1. Note that the bit remains unchanged when being already set to 1. To check whether an element x is in the set S, we can check whether the bits $B[h_j(x)]$ for $1 \leq j \leq k$ are all 1's. If and only if they are all equal to 1, x is deemed to be an element of S. The size b of the Bloom filter is independent of the size of elements and can be constant, which is very memory efficient. Nevertheless, the membership query on the Bloom filter has false positive but has no false negative. In other words, it is probable to falsely consider an element that actually does not belong to S as an element of S. In [9], such false-positive probability can be obtained as $(1-(1-(1/b))^{km})^k \approx (1-e^{-km/b})^k$. The optimization between the performance efficiency (e.g., array length or hash functions required) of the Bloom filter and the false-positive probability can be obtained, but it is beyond the scope of this paper. Note that one of the characteristics of the Bloom filter we use in the design of our proposed solutions is that the query result is always correct if the content to be queried is indeed stored in the Bloom filter.

C. Multikeyword Similarity Search

In the following, we define the problem of multikeyword similarity search over outsourced cloud data. Given a collection $C = \{f_1, \ldots, f_n\}$ of encrypted files, a set $W = \{w_1, \ldots, w_p\}$ of predefined distinct keywords, a set $X = \{x_1, \ldots, x_q\}$ of keywords, a threshold d for the minimum edit distance, and a threshold δ for the minimum occurrences of keywords appearing in the file, the result of the multikeyword similarity search is C_X such that, for each file $f \in C_X$, $\sum_{i=1}^{q} \beta_i \geq \delta$, where β_i is defined as $\beta_i = |S_{k(f),d} \cap S_{x_i,d}|$ with $K(f) = \bigcup_w S_{w,d}$, where w's are the keywords extracted from the file f, denoting the set of all keyword variants contained in the file f.

IV. Proposed Methods

In this section, we propose three solutions to enable PPMKSS. The protocol that the user purchases the authorization token without compromising the user access privacy is presented in Section IV-A. The first solution, i.e., PPMKSS-1, described in Section IV-B, can be regarded as a direct extension of the method in [31]. The second solution, i.e., PPMKSS-2, which is described in Section IV-C, finds the keyword existence by leveraging the bit pattern of the Bloom filter. The third solution, i.e., PPMKSS-3, which is described in Section IV-D,

TABLE I
NOTATION TABLE

	Description
n	the number of files
c_i	the i-th file
C	the set of files to be outsourced
p	the number of distinct keywords
w_i	the i-th keyword
W	the set of keywords defined in the system
q	the number of keywords input by the user
x_i	the i-th input keyword
X	the set of input keywords
d	the edit distance input by the user
δ	the minimum number of keyword appearances in the returned files
$S_{w,d}$	the set of keyword variants, each of which has edit distance $\leq d$ to the word w
B	the Bloom filter
$S_{W,d}$	the set of keyword variants, each of which has edit distance $\leq d$ to the words $w \in W$
C_X	the search result returned by the cloud
$K(f)$	the set of all keyword variants contained in the file f
F_w	the set of files containing the keywords w
$C_{w,d}$	the set of files containing the keywords in $S_{w,d}$
$H(S_{X,d})$	the set of the hashes of the elements from $S_{X,d}$

utilizes the bit pattern of the Bloom filter on the index, instead of on the files, to make sure whether a specific keyword is contained in a file. The notations frequently used in this paper are summarized in Table I.

A. User Authorization

In our considered scenario, the user may access the files in the cloud by presenting the authorization token to the cloud first. More precisely, for the access control policy, we assume that once the user presents an authorization token to the cloud, the user is allowed to perform PPMKSS once. Moreover, after a particular authorization token is used for a search request, it is no longer valid. To perform another keyword search, the user is required to present another authorization token. The straightforward solution is to use the digital signature [5] to construct the authorization tokens. In particular, when the user purchases the authorization token, the owner signs and then sends back the authorization token to the user. Hence, if the user has a need to perform the search, the user submits the signed token to the cloud. The cloud returns the search result if the signature on the signed token can be verified successfully. On the other hand, for the user access privacy, it is assumed that, although many users may gain the right to access the files in the cloud, the user does not want to leak the information about who is submitting a search request. Therefore, this use of the conventional digital signature cannot have the guarantee of the user access privacy.

To achieve the user access privacy, we propose to use blind signature protocol [11], which itself depends on the well-known Rivest–Shamir–Adleman (RSA) cryptosystem. The use of blind signature can have the guarantee that the owner signs and the cloud verifies the authorization token without knowing the user's identity. In particular, as stated earlier, the blind signature is based on RSA. Thus, the owner creates the RSA key pair, the public key $\langle \mu, e \rangle$, and the private key π. Here, the public key $\mu = p_1 p_2$ is the product of two prime numbers p_1 and p_2, and $e \in [1, \phi]$ is coprime to $\phi = (p_1 - 1)(p_2 - 1)$. In addition, π is selected such that $e\pi = 1 (\text{mod } \phi)$. The public key μ here is usually chosen as a 1024-bit number for the sufficient security guarantee. Note that, in our use, the public key e is used for the signature verification only. Because e determines the computation cost at the cloud side and larger e implies more computation cost, e can be chosen to be 3 for achieving a very efficient signature verification. Afterward, the owner sends the public key to the cloud and keeps the private key confidential. This enables the cloud to verify the signed token while guaranteeing that no one can forge the signature.

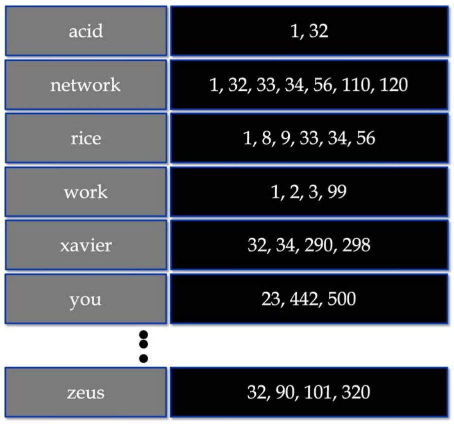
```
Algorithm: The blind signature
1. the user picks a $\lambda$-bit random number $m$
2. the user picks a secret key $v$ satisfying $gcd(\mu, v) = 1$
3. the user sends $m^* = mv^e \pmod{\mu}$
4. the owner returns $\sigma_m^* = (m^*)^d$
5. the user calculates $\sigma = v^{-1}\sigma_m^*$
6. the user keeps $\langle m, \sigma_m \rangle$
```

Fig. 2. Blind signature.

The user is required to purchase the authorization token to perform the search. In our use, the authorization token is assumed a λ-bit random bit string, where λ is a system parameter. Therefore, how the owner uses the blind signature is to sign a random bit string. More precisely, the procedure is that the user generates an authorization token and sends it to the owner. The owner signs the received authorization token and sends it back to the user. When the user wants to perform the search, it presents the signed authorization token to the cloud for the verification. As can be easily observed, the authorization token itself being a random number does not have a role in achieving the security. The importance comes from the signature associated with the token. Only if the signature can be verified successfully that the cloud returns the search result. The algorithms executed jointly by the owner and user to sign and obtain the signed authorization token is shown in Fig. 2. In Step 3, if the owner faithfully follows the protocol, the owner returns the σ_m^* to the user. Note that σ_m^* can be thought of as the signature of the authorization token anonymized by a random number k. In Step 4, one can easily see that σ_m actually is a valid RSA signature on m because

$$\begin{aligned}\sigma_m &= k^{-1}\sigma_m^* = k^{-1}m^d k^{\text{ed}} \\ &= k^{-1}m^d k = m^d \pmod{\mu}. \qquad (1)\end{aligned}$$

In Fig. 2, one can know that the owner cannot know both m and σ_m. This implies that the owner and cloud cannot distinguish the origins of two signatures, therefore achieving the user access privacy. In other words, even if the owner and cloud receive an authorization token, they cannot link it to the user who generates the token. Note that, although the authorization token can have different monetary value and therefore represents different numbers of authorized search requests, we do not consider such case in this paper. However, the proposed protocol can be extended to allow the authorization token with different monetary values in a straightforward way.



acid	1, 32
network	1, 32, 33, 34, 56, 110, 120
rice	1, 8, 9, 33, 34, 56
work	1, 2, 3, 99
xavier	32, 34, 290, 298
you	23, 442, 500
⋮	
zeus	32, 90, 101, 320

Fig. 3. Example of the inverted list.

B. PPMKSS-1

1) Basic Idea: Inverted list is a fundamental technique to support keyword search. We thus still follow the notion of inverted list to have our design for similarity search.

Inverted list can be considered a 2-D array, each row of which consists of a tuple. The length of the array is equivalent to the number keywords that can be searched in the system. The first entry of each tuple is a keyword, whereas the second entry of each tuple consists of the file IDs that the corresponding files contain the keyword.

The basic idea to support multikeyword similarity search is to construct a very large inverted list such that, even if the input keywords are not included in the keywords predefined in the system, they still can be found in the inverted list. In this sense, the "similar" keywords can also be thought of as part of predefined keywords. Afterward, since many file IDs will be found including some of input keywords, some postprocessing will be conducted to identify the files that contain at least δ input keywords. An example can be shown in Fig. 3. It can be seen that each keyword is associated with a list of file IDs. Consider the first row for example. It describes that, files 1 and 32 contain the keywords "acid," and the files 23, 442, and 500 contain the keyword "you." The inverted list is beneficial to the exact keyword search. Consider an example where the input keywords are "acid" and "network." The way the cloud responds to the search request is to perform the set intersection between {1,32} and {1,32,33,34,56,110,120}. After that, the cloud finds that there are two files, i.e., files 1 and 32, containing both input keywords.

2) Protocol Description: The proposed PPMKSS-1 method is shown in Fig. 4. There are two phases: preprocessing and searching. During the preprocessing phase, the owner constructs and sends out the index for the further searching purpose. During the searching phase, the authorized user sends the keywords to the cloud, and then the cloud returns the search result to the user.

The aforementioned method of constructing a very large inverted list is theoretically useful; however, it is practically useless. This is because the storage cost is overwhelming. For example, consider the keyword CLOUD. In the case of edit distance d, this implies $((\ell_w+1)+\ell_w)^d 26^d$ more entries

```
**Algorithm:** PPMKSS-1
**Preprocessing phase:**
1. owner: construct the encrypted inverted list $I$,
in which each row is $\langle h_{\tilde{k},w}, \mathcal{E}_{\tilde{k}}(F_w)\rangle$,
where $w \in S_{w',d}$, $w' \in W$
2. owner $\rightarrow$ cloud: $I$
**Searching phase:**
1. authorization token verification succeeds
2.   user: generate $S_{X,d}$ and the corresponding $H(S_{X,d})$,
where $H(S_{X,d})$ is the set of hashes
with (the key $\tilde{k}$) of each element in $S_{W,d}$
3.   user $\rightarrow$ cloud: $H(S_{W,d})$
4.   cloud: construct $C_{x_i,d}$, $1 \le i \le q$, by directly
comparing the elements in $H(S_{X,d})$ to the elements
in the first column of $I$
5.   cloud: examine $\binom{q}{\delta}$ combinations of $C_{x_i,d}$'s
to see whether the file under the consideration
contains an enough number of input keywords
6.   cloud $\rightarrow$ user: $C_X$
7.
8.   cloud: deny the search request
```

Fig. 4. PPMKSS-1.

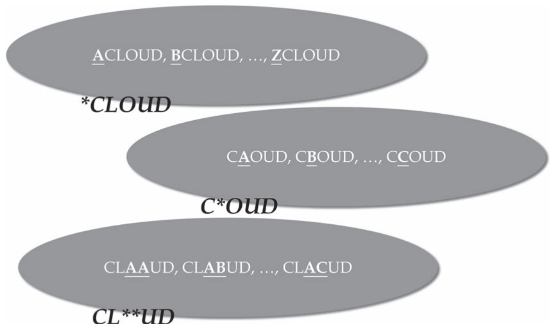


Fig. 5. Example of the use of wildcard in the suppressing technique.

needed to be inserted to the inverted list, where ℓ_w is the number of alphabets, because there are $(\ell_w+1)+\ell_w$ positions that can be manipulated and each position has 26 possibilities for alphabets if only alphabets are considered. Obviously, directly constructing and maintaining such a huge inverted list would be a heavy burden. Thus, the technique of suppressing the length of inverted list [20], [31] will be used here.

The method of suppressing the inverted list is to use the wildcard * to succinctly represent many keyword variants by using one only keyword. For example, in the case of $d=1$, the variants of keyword CLOUD could be ACLOUD, BCLOUD, . . ., ZCLOUD, . . . if none of the suppressing method is used. However, if the suppressing technique is used, the given keyword variants could be represented as *CLOUD only, in which * can present all of the possibilities that the wildcard can be filled. An example can be found in Fig. 5. Obviously, once the suppressing technique is used, the number of keyword variant representation can be significantly reduced, although in the given case, *CLOUD only represent ACLOUD, BCLOUD, . . ., ZCLOUD, and many of the other variants are to be transformed.

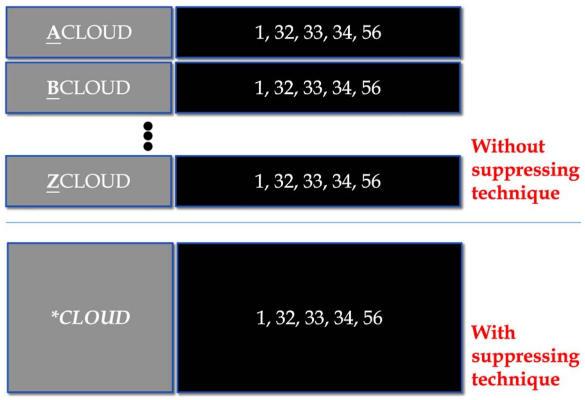


Fig. 6. Example of storage cost reduction by using the suppressing technique.

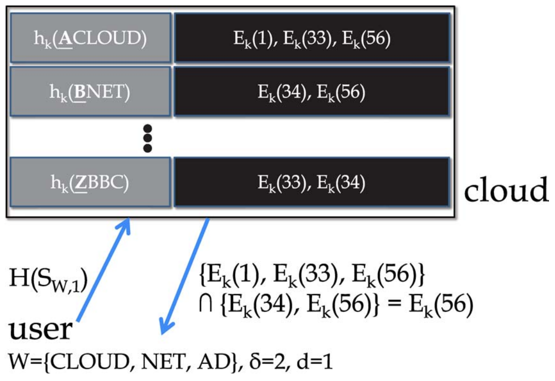


Fig. 7. PPMKSS-1.

This directly implies that the length of the inverted list can also be significantly reduced. A concrete example of using the suppressing technique to suppress the keyword variants is shown in Fig. 6.

After the suppressing technique is used, the owner applies the cryptographic hash function to the first column of the inverted list (line 1 of preprocessing phase in Fig. 4), i.e., the owner calculates the corresponding hash. Afterward, the owner applies the encryption algorithm, such as Advanced Encryption Standard, to the second column of the inverted list (line 1 of preprocessing phase in Fig. 4), i.e., the owner calculates the corresponding encrypted list of file IDs. Note that the owner actually generates two distinct secret keys first, and then use these two keys to calculate the hash and encryption, respectively. Finally, the owner sends such an encrypted inverted list to the cloud (line 2 of preprocessing phase in Fig. 4). Apparently, from the cloud point of view, the cloud, after receiving such an encrypted inverted list, cannot have much information. The cloud can only see a list of hash values and the encryptions. Without the secret key, the cloud cannot know the content of the encrypted inverted list.

Suppose that the user wants to perform the multikeyword similarity search, then it needs to purchase the authorization token from the owner. As stated in Section IV-A, the user can obtain the authorization tokens without the worry about the identity leakage. Therefore, in the following description, we assume that the user has obtained at least one authorization token and wants to perform the multikeyword similarity search. In this case, the user submits the authorization token to the cloud for the authorization verification purpose. The user's search request will be denied if the authorization verification fails and will be further processed otherwise. In the case of the successful verification, the owner first calculates all of the possible keyword variants of the input keywords. Let $X = \{x_1, \ldots, x_q\}$ be the set of input keywords. The user computes $S_{x_i,d}$, $i \in [1, q]$, where $S_{x_i,d}$ denotes the set of all words with the edit distance d to the input keyword x_i (line 1 of searching phase in Fig. 4). In other words, each element in $S_{x_i,d}$ is either the input keyword itself or a variant of the input keywords. The reason that the user calculates such a set is to fit the setting of how the owner constructs the inverted list, where each element of the first column of the inverted list is a possible keyword variant. After that, the user computes the hashes of these keyword variants (line 1 of searching phase in Fig. 4). This procedure is also used to fit the setting of the encrypted inverted list. Here, the hash of the keyword variant is used as an index to look up the corresponding file IDs whose corresponding files contain the requested keyword.

After receiving the hashes from the user, the cloud searches for the matching in the encrypted inverted list (line 4 of searching phase in Fig. 4). However, instead of searching for the matching directly, the cloud searches for the matching for different categories. In other words, for each $S_{x_i,d}$, the cloud constructs a set $C_{x_i,d}$ of file IDs whose corresponding files contain the keywords specified in $S_{x_i,d}$. For example, if the file ID 35 is associated with an entry in $S_{x_i,d}$, then the file ID 35 is included in $C_{x_i,d}$. Such file ID can be found by looking to the encrypted inverted list. After the construction of $C_{x_i,d}$, the cloud can be sure that each file in $C_{x_i,d}$ contains an input keyword variant. Thus, a natural way to discover which files contain at least δ input keywords is to examine the different combinations of the $C_{x_i,d}$. For example, in the case of $|X| = q = 3$ and $\delta = 2$, the cloud would have $C_{x_1,d}$, $C_{x_2,d}$, and $C_{x_3,d}$. Then, the cloud can perform the set intersection operation over the $C_{x_1,d}$, $C_{x_2,d}$, and $C_{x_3,d}$, i.e., the cloud calculates $C_{1,2} = C_{x_1,d} \cap C_{x_2,d}$, $C_{2,3} = C_{x_2,d} \cap C_{x_3,d}$, and $C_{1,3} = C_{x_1,d} \cap C_{x_3,d}$. Here, each time, only two $C_{x_i,d}$ are considered because of the δ specified in the user's request. Obviously, we can see that, for example, the files in $C_{1,2}$ all contain at least two input keywords or their keyword variants. Finally, the cloud sends back $C_X = C_{1,2} \cup C_{2,3} \cup C_{1,3}$ to the user as the search result.

Once the user receives the result returned by the cloud, it decrypts the encrypted file IDs to obtain the files containing at least two specified keywords or their keyword variants.

The solution (see Fig. 7) can be used to solve the problem of PPMKSS. As can be easily seen, this method only has one parameter to be set. In particular, in the preprocessing phase, the owner constructs the inverted list and the corresponding encrypted inverted list. Here, only d affects the storage cost at the cloud side. When the parameter d is larger, more rows are considered in the inverted list and encrypted inverted list. In the search phase, the parameters d and δ are user-specified parameters that can be variant because of different practical applications. They are also not the parameters required in

```
Algorithm: PPMKSS-2
Preprocessing phase:
1. owner: construct the Bloom table $I'$,
in which each row is $\langle$file ID $i, B(K(f_i))\rangle$
2. owner $\rightarrow$ cloud: $I'$
Searching phase:
1. authorization token verification succeeds
2.  user $\rightarrow$ cloud: $H(S_{X,d})$
3.  cloud: construct $C_{x_i,d}$, $1 \leq i \leq q$, by checking
whether the elements in $H(S_{X,d})$ are in $I'$
4.  cloud: examine $\binom{q}{\delta}$ combinations of $C_{x_i,d}$'s
to see whether the file under the consideration
contains an enough number of input keywords
5.  cloud $\rightarrow$ user: $C_X$
6.
7.  cloud: deny the search request
```

Fig. 8. PPMKSS-2.

the solution. Nonetheless, the cloud has the heavy burden to calculate a large number of hashes to make sure which files contain the specified keywords. In the following, we take a different approach (as shown in Section IV-C) to find out the files with the specified input keywords.

C. PPMKSS-2

1) Basic Idea: Instead of using the inverted list, in PPMKSS-2, the owner uses a more direct way to do the keyword search. In particular, once the keywords are selected and associated with the file, when the input keywords are sent to the cloud, the cloud is able to find out the files containing the input keywords. However, this straightforward method implies two problems. First, since there would be a large number of keyword variants, directly comparing the keywords would be a heavy computation cost for the cloud. Thus, we propose to use the Bloom filter to store the keyword variant and at the same time offer a more efficient membership query at the cloud side. However, the induced problem from the use of the Bloom filter is that, because the Bloom filter represents the keyword variants, the cloud may also query the Bloom filter to know the keywords associated with the particular files, breaching the keyword privacy. To solve this problem, we propose to use different sets of hash functions to disable the cloud's ability to check the keyword existence in files.

2) Protocol Description: The proposed PPMKSS-2 method is shown in Fig. 8. Before outsourcing the files to the cloud, the owner constructs a table, in which the first column consists of the file IDs, whereas the second column consists of the keywords and the keyword variants extracted from the corresponding file (line 1 of the preprocessing phase in Fig. 8). As can easily be known, such setting is completely opposite to the case in Section IV-B. Here, keyword variants are associated with the files, whereas in PPMKSS-1, the file IDs are associated with the keywords. As we can know from information retrieval literature, the setting in this section is more inefficient during

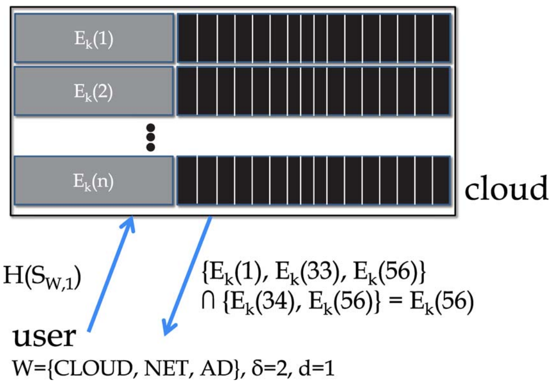

$E_k(1)$
$E_k(2)$
$E_k(n)$
cloud
$H(S_{W,1})$
$\{E_k(1), E_k(33), E_k(56)\}$
$\cap \{E_k(34), E_k(56)\} = E_k(56)$
user
W={CLOUD, NET, AD}, δ=2, d=1

Fig. 9. PPMKSS-2 (the black parts denote Bloom filters).

the file retrieval. However, the setting in this section serves to explain the use of the Bloom filter in retrieving the files. A more practical solution will be presented in Section IV-D. An example of such a table is shown in Fig. 9, where the first column (gray column) contains the encrypted file Ids, and the second part (black columns) consists of the keywords. Similar to the given setting, the owner applies the encryption function to the first column, resulting the encrypted file IDs. In the setting in Section IV-B, the owner is supposed to apply the hash function to each keyword variant in the second column. Nevertheless, instead of doing so, the owner actually adds all of the keyword variants in a row to a Bloom filter. Here, let B_i be the Bloom filter associated with the file ID i. The table with the first column being the encrypted file ID and the second column being the Bloom filter constructed as given is called the Bloom table. After constructing the Bloom table, the owner sends the Bloom table to the cloud.

Suppose that the user wants to perform the multikeyword similarity search, then it needs to purchase the authorization token from the owner. The token verification procedures are completely the same as given earlier; therefore, we omit the corresponding description here. Assume the $X = \{x_1, \ldots, x_q\}$ are the keywords to be searched. The user computes $S_{x_1,d}, \ldots, S_{x_q,d}$, in a way similar to that given. Afterward, the users calculates and submits the hashes of the $S_{x_1,d}, \ldots, S_{x_q,d}$ to the cloud.

After receiving the hashes of the $S_{x_1,d}, \ldots, S_{x_q,d}$ from the user, the cloud checks whether they are in the B_i for each i. The file i is included in C_X if $\sum_{j=1}^{q} \theta_j \geq \delta$, where $\theta_j = 1$ if some elements in $S_{x_j,d}$ appears in f_i and $\theta_j = 0$ otherwise.

D. PPMKSS-3

The PPMKSS-2 method, although theoretically feasible, is very inefficient in terms of search time because the cloud needs to access each file to make sure whether the particular files contain the input keyword variants. Thus, in this section, we combine the idea about the encrypted inverted list in Section IV-B and the idea about the Bloom table in Section IV-C to construct a new data structure to support the PPMKSS.

PPMKSS-3 is shown in Fig. 10. The owner first constructs the inverted list as in Section IV-B. However, instead of con-

```
**Algorithm:** PPMKSS-3
**Preprocessing phase:**
1. owner: construct the Bloom inverted list $I''$,
in which each row is $\langle B(S_{w,d}), \mathcal{E}_{\tilde{k}}(F_w)\rangle$,
where $B(S_{w,d})$ denotes a Bloom filter
with the elements from $S_{w,d}$
2. owner: insert $z - |I''|$ dummy elements to $B(S_{w,d})$
3. owner $\rightarrow$ cloud: $I''$
**Searching phase:**
1. authorization token verification succeeds
2.  user: generate $B(S_{X,d})$
3.  user $\rightarrow$ cloud: $B(S_{X,d})$
4.  cloud: construct $C_{x_i,d}$, $1 \leq i \leq q$, by directly
comparing $B(S_{X,d})$ to $B(S_{w,d})$'s, $w \in W$
5.  cloud: examine $\binom{q}{\delta}$ combinations of $C_{x_i,d}$'s
to see whether the file under the consideration
contains an enough number of input keywords
6.  cloud $\rightarrow$ user: $C_X$
7.
8.  cloud: deny the search request
```

Fig. 10. PPMKSS-3.

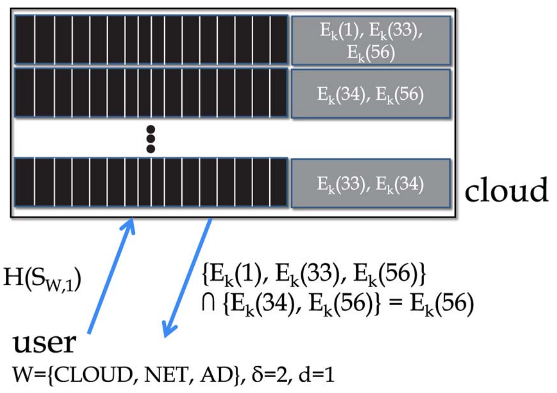

E$_k$(1), E$_k$(33), E$_k$(56)
E$_k$(34), E$_k$(56)
E$_k$(33), E$_k$(34)
cloud
H(S$_{W,1}$)
{E$_k$(1), E$_k$(33), E$_k$(56)} ∩ {E$_k$(34), E$_k$(56)} = E$_k$(56)
user
W={CLOUD, NET, AD}, δ=2, d=1

Fig. 11. PPMKSS-3 (the black parts denote Bloom filters).

structing the corresponding encrypted inverted list, the owner constructs a so-called Bloom inverted list (line 1 of the preprocessing phase in Fig. 10), where the first column of the Bloom inverted list corresponds to the hashes of the first column of the inverted list. In other words, each element in the first column of the encrypted Bloom inverted list is the hash of a keyword variant. An example is shown in Fig. 11. In this sense, the first column of the Bloom inverted list is completely the same as the first column of the encrypted inverted list in Section IV-B. Nevertheless, the file IDs in the inverted list are all added to a Bloom filter. As a consequence, for a Bloom inverted list, each keyword variant is associated with a Bloom filter. However, if the Bloom inverted list is directly outsourced to the cloud, the cloud is also able to test the keyword existence for each file by simply launching the membership query. To counteract against such abuse, we add $z - |I''|$ dummy elements to $B(S_{w,d})$ (line 2 of preprocessing phase in Fig. 10). By controlling the number of dummy elements, the Bloom inverted list is secure against the cloud attempting to know the keyword existence in files. Finally, at the end of the preprocessing phase, the owner sends the Bloom inverted list to the cloud.

Suppose that the user wants to perform the multikeyword similarity search, then it needs to purchase the authorization token from the owner. The token verification procedures are completely the same as given earlier; therefore, we omit the corresponding description here. Assume the $X = \{x_1, \ldots, x_q\}$ are the keywords to be searched. The user computes $S_{x_1,d}, \ldots, S_{x_q,d}$, in a way similar to that given. Afterward, the users calculates and submits Bloom filter $B(S_{X,d})$ to the cloud.

After receiving $B(S_{X,d})$ from the user, the cloud simply counts the number of common 0's in both $B(S_{X,d})$ and $B(S_{w,d})$, $w \in W$. By knowing the number of common 0's in both $B(S_{X,d})$ and $B(S_{w,d})$, one can infer the number of elements appearing in both sets. Note that this technique has been developed in [28] very recently. In essence, given the number of common bit 0 positions n_0 in both the Bloom filters $B(S_{X,d})$ and $B(S_{w,d})$, the number of elements appearing in both sets can be estimated as

$$\frac{2\left|\bigcup_{w\in W} S_{w,d}\right| \frac{z-|I''|}{\left|\bigcup_{w\in W} S_{w,d}\right|} - |I''|\left(\ln|I''| - \ln n_0\right)}{k} \tag{2}$$

$$= \frac{2\left(z-|I''|\right) - |I''|\left(\ln|I''| - \ln n_0\right)}{k}. \tag{3}$$

The remaining procedures are exactly the same as those presented in the first solution. For example, in the case of $|X| = q = 3$ and $\delta = 2$, the cloud would have $C_{x_1,d}$, $C_{x_2,d}$, and $C_{x_2,d}$. Then, the cloud can perform the given estimation over the $C_{x_1,d}$, $C_{x_2,d}$, and $C_{x_2,d}$, i.e., the cloud calculates $C_{1,2} = C_{x_1,d} \cap C_{x_2,d}$, $C_{2,3} = C_{x_2,d} \cap C_{x_3,d}$, and $C_{1,3} = C_{x_1,d} \cap C_{x_3,d}$. Here, each time, only two $C_{x_i,d}$'s are considered because of the δ specified in the user's request. Obviously, we can see that, for example, the files in $C_{1,2}$ all contain at least two input keywords or their keyword variants. Finally, the cloud sends back $C_X = C_{1,2} \cup C_{2,3} \cup C_{1,3}$ to the user as the search result.

V. Performance Evaluation and Security Analysis

The current keyword search on commercial systems is implemented with the inverted list. Our proposed PPMKSS-2 method is not an inverted list-based approach and therefore not practically useful. The main purpose of PPMKSS-2 is to inspire us to increase the performance by taking advantage of the Bloom filter. Hence, we eventually decide not to include the numerical results in this paper.

A. *Performance Evaluation*

We evaluate the performance according to the storage cost, computation cost, and communication cost. First of all, the storage cost of three proposed solutions is highly efficient mainly due to the use of the suppressing technique and Bloom filter.

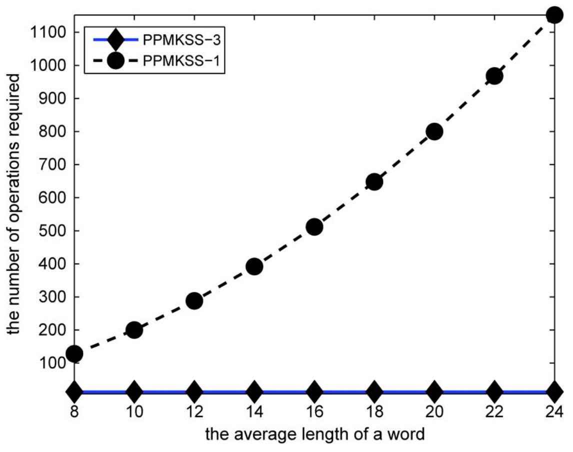


Fig. 12. Comparison of the computation cost of PPMKSS-1 and PPMKSS-3.

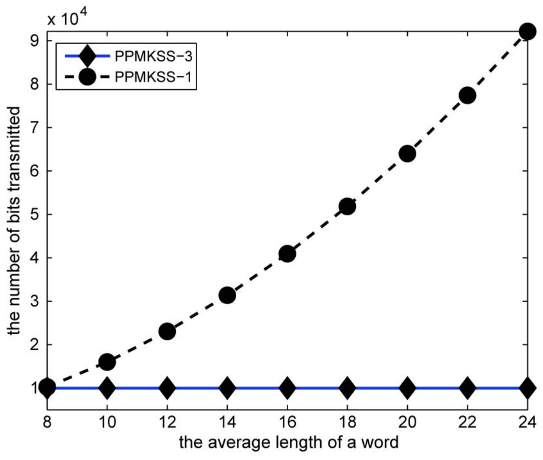


Fig. 13. Comparison of the communication cost of PPMKSS-1 and PPMKSS-3.

The use of the suppressing technique significantly shortens the length of the inverted list, directly implying the reduced storage cost. Moreover, the use of the Bloom filter further cuts the storage overhead of the inverted list because, in the original case, the hashes need to be stored in plaintext and cause a high storage burden, whereas after the use of the Bloom filter, the hashes can be all added to a single Bloom filter. Because in the proposed solutions, only membership query is concerned. The Bloom filter offers a storage-efficient way to accomplish the membership query in our proposed solutions.

On the other hand, the computation cost is also proportional to the storage cost because more storage cost implies that more elements need to be checked, resulting in more computation cost. In this sense, the use of suppressing technique and the Bloom filter also contributes to the computation overhead reduction. The computation cost reduction due to the use of the suppressing technique is obvious; furthermore, the comparison of PPMKSS-1 and PPMKSS-3, shown in Fig. 12, can also be used to show the computation overhead reduction due to the use of the Bloom filter. In Fig. 12, we can see that the computation cost of PPMKSS-1 grows steadily, whereas the computation cost of PPMKSS-3 remains stable. This can be explained in the following way. In the first solution, as d becomes larger, the number of possible keyword variants also grows, which means that the length of the inverted list will be increased. As a consequence, the computation cost is increased as well because the cloud has to examine each element individually. However, in PPMKSS-3, only the number of common 0's of two Bloom filters matters. Counting the number of 1's from the intersection of two bit strings can be accomplished in logarithmic time. Therefore, the curve of the computation cost of PPMKSS-3 goes rather stable.

Moreover, the communication cost actually also depends on the use of the suppressing technique and the Bloom filter. In the first solution, since no Bloom filter is used, the user has to submit the hashes of all possible keyword variants individually. Therefore, the communication cost depends on the average length of the input keywords. However, in PPMKSS-3, the communication cost is determined by the number of keyword specified in the system. Note that, usually, the number of the keywords specified in the system is more than the number of keywords manually input by the user. Thus, compared with PPMKSS-1, PPMKSS-3 can achieve much lower communication cost. This can also be confirmed in Fig. 13.

B. Security Analysis

Our proposed solutions are supposed to fulfill the requirements stated in Section II-C. The first solution can be thought of as a natural extension of the method in [31]. In essence, only some postprocessing are further considered compared with the method in [31]. Thus, the security of our first solution directly inherits the security guarantee of the method in [31]. In addition, PPMKSS-3 strikes the balance between search performance and incurred overhead; here for simplicity, we only discuss the security analysis of PPMKSS-3.

PPMKSS-3 achieves the data privacy simply because the files are always encrypted from the cloud point of view. Thus, the cloud, without the access of the secret key, cannot know the file contents. In addition, because of the use of blind signature, the user access privacy can also be guaranteed. More specifically, both the owner and cloud cannot know which user ever submits or is submitting the search request to the cloud. Moreover, the keyword–file association privacy, keyword privacy, and file privacy can also be achieved simply because the keywords are not revealed in plaintext, and the files are always encrypted. Hence, the cloud is not able to infer the association between them or infer the frequently used keyword/file.

VI. Conclusion

We consider the problem of PPMKSS over outsourced cloud data, for the first time in the literature. With the keyword suppressing technique and the Bloom filter, three solutions, namely, PPMKSS-1, PPMKSS-2, and PPMKSS-3, are proposed as candidates for dealing with such search problem. In particular, PPMKSS-3 is highly efficient in terms of storage, computation, and communication overhead. Moreover, we also design a user authorization mechanism based on blind signature, to ensure the user access privacy. As a whole, based on our evaluation, the proposed schemes can be practically useful in offering PPMKSS.

References

[1] F.-H. Liu, H.-F. Lo, L.-C. Chen, and W.-T. Lee, "Comprehensive security integrated model and ontology within cloud computing," *J. Internet Technol.*, vol. 14, no. 6, pp. 935–946, 2013.

[2] M. J. Atallah, *Algorithms and Theory of Computation Handbook*. Boca Raton, FL, USA: CRC Press, 1998.

[3] M. Armbrust *et al.*, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

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

[5] B. Schneier, *Applied Cryptography: Protocols, Algorithms, Source Code in C*. Hoboken, NJ, USA: Wiley, 1996.

[6] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in *Proc. Annu. CRYPTO*, 2007, pp. 535–552.

[7] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. Int. Conf. Theory Appl. EUROCRYPT*, 2004, pp. 506–522.

[8] L. Ballard, S. Kamara, and F. Monrose, "Achieving efficient conjunctive keyword searches over encrypted data," in *Proc. ICICS*, 2005, pp. 414–426.

[9] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet Math.*, vol. 1, pp. 485–509, 2005.

[10] D. Boneh and B. Waters, "Conjunctive, subset, range queries on encrypted data," in *Proc. Theory of Cryptography Conf. TCC*, 2007, pp. 535–554.

[11] D. Chaum, "Security without identification: Transaction systems to make big brother obsolete," *Commun. ACM*, vol. 28, no. 10, pp. 1030–1044, Oct. 1985.

[12] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. ACM Conf. CCS*, 2006, pp. 79–88.

[13] D. Cash *et al.*, "Dynamic searchable encryption in very-Large databases: Data structures and implementation," in *Proc. NDSS*, 2014, pp. 1–16.

[14] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data," in *Proc. ACNS*, 2005, pp. 442–455.

[15] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou, "Privacy-preserving multi-keyword ranked search over encrypted cloud data," in *Proc. IEEE INFOCOM*, 2011, pp. 829–837.

[16] E.-J. Goh, "Secure indexes," Cryptology ePrint Archive, Santa Barbara, CA, USA, Rep. No. 2003/216, 2004.

[17] P. Golle, J. Staddon, and B. R. Waters, "Secure conjunctive keyword search over encrypted data," in *Proc. ACNS*, 2004, pp. 31–45.

[18] M. Kargar, A. An, N. Cercone, P. Godfrey, and J. Szlichta, "MeanKS: Meaningful keyword search in relational databases with complex schema," in *Proc. ACM Int. Conf. SIGMOD*, 2014, pp. 905–908.

[19] S. Kamara and K. Lauter, "Cryptographic cloud storage," in *Proc. Financial Cryptography Workshops*, Jan. 2010, pp. 136–149.

[20] J. Li *et al.*, "Fuzzy keyword search over encrypted data in cloud computing," in *Proc. IEEE INFOCOM*, 2010, pp. 1–5.

[21] C. Li, B. Wang, and X. Yang, "Vgram: Improving performance of approximate queries on string collections using variable-length grams," in *Proc. Int. Conf. VLDB*, 2007, pp. 303–314.

[22] M. Naveed, M. Prabhakaran, and C. A. Gunter, "Dynamic searchable encryption via blind storage," in *Proc. IEEE Symp. SP*, 2014, pp. 639–654.

[23] M. Qiao, L. Qin, H. Cheng, J. X. Yu, and W. Tian, "Top-K nearest keyword search on large graphs," *Very Large Data Base Endowment*, vol. 6, no. 10, pp. 901–912, Aug. 2013.

[24] E. Shi, J. Bethencourt, T.-H. H. Chan, D. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *Proc. IEEE Symp. SP*, 2007, pp. 350–364.

[25] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," in *Proc. NDSS*, 2014, pp. 1–15.

[26] W. Sun *et al.*, "Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 11, pp. 3025–3035, Nov. 2014.

[27] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. IEEE Symp. SP*, 2000, pp. 44–55.

[28] J. Sun, R. Zhang, and Y. Zhang, "Privacy-preserving spatialtemporal matching," in *Proc. IEEE INFOCOM*, 2013, pp. 800–808.

[29] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," in *Proc. ICDCS*, 2010, pp. 253–262.

[30] C. Wang, N. Cao, K. Ren, and W. Lou, "Enabling secure and efficient ranked keyword search over outsourced cloud data," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 8, pp. 1467–1479, Aug. 2012.

[31] C. Wang, K. Ren, S. Yu, and K. M. R. Urs, "Achieving usable and privacy-assured similarity search over outsourced cloud data," in *Proc. IEEE INFOCOM*, 2012, pp. 451–459.

[32] Z. Zhang, M. Hadjieleftheriou, B. C. Ooi, and D. Srivastava, "Bed-tree: An all-purpose index structure for string similarity search based on edit distance," in *Proc. ACM Int. Conf. SIGMOD*, 2010, pp. 915–926.

Chia-Mu Yu received the Ph.D. degree from National Taiwan University, Taipei, Taiwan, in 2012.

From 2005 to 2010, he was a Research Assistant with the Institute of Information Science, Academia Sinica, Taipei. From September 2010 to September 2011, he was a Visiting Scholar with Harvard University, Cambridge, MA, USA. From January 2012 to September 2012, he was a Visiting Scholar with Imperial College London, London, U.K. From October 2012 to July 2013, he was a Postdoctoral Researcher with IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA, where he will serve as Visiting Professor from February 2015 to March 2015. He is currently an Assistant Professor with the Department of Computer Science and Engineering, Yuan Ze University, Taoyuan, Taiwan. He is also with Innovation Center for Big Data and Digital Convergence, Taoyuan.

Dr. Yu is currently an Associate Editor for IEEE ACCESS and an Associate Editor for *Security and Communication Networks* and for *Gate to Multimedia Processing*. His research interests include cloud security, sensor network security, and cryptography.

Chi-Yuan Chen received the Ph.D. degree in electrical engineering from National Dong Hwa University, Hualien, Taiwan, in 2014.

Since 2014, he has been an Assistant Professor with the Department of Computer Science and Information Engineering, National Ilan University, Yilan, Taiwan. His research interests include mobile communication, network security, and quantum computing.

Dr. Chen has served as the Associate Editor-in-Chief for *Journal of Internet Technology* since February 2014. He is a member of the Association for Computing Machinery.

Han-Chieh Chao (SM'04) received the M.S. and Ph.D. degrees in electrical engineering from Purdue University, West Lafayette, IN, USA, in 1989 and 1993, respectively.

From September 2008 to July 2010, he served as the Director of the Computer Center for the Ministry of Education of Taiwan. He is currently a Joint Appointed Full Professor with the Department of Electronic Engineering and the Institute of Computer Science and Information Engineering, National Ilan University, Yilan, China, where he also serves as President. He is also with the Department of Electrical Engineering, National Dong Hwa University, Hualien, Taiwan; the College of Information Engineering, Yangzhou University, Yangzhou, China; and the School of information Science and Engineering, Fujian University of Technology, Fuzhou, China.

Dr. Chao serves as the Editor-in-Chief for *Institution of Engineering and Technology (IET) Networks*, the *Journal of Internet Technology*, the *International Journal of Internet Protocol Technology*, and the *International Journal of Ad Hoc and Ubiquitous Computing* He has served as the Guest Editor for *Mobile Networking and Applications*, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, IEEE COMMUNICATIONS MAGAZINE, *Computer Communications*, *IEE Proceedings Communications*, the *Computer Journal*, *Telecommunication Systems* , *Wireless Personal Communications*, and *Wireless Communications & Mobile Computing*. He is a Fellow of IET (IEE) and a Chartered Fellow of the British Computer Society.