Achieve Efficient and Verifiable Conjunctive and Fuzzy Queries over Encrypted Data in Cloud

Jun Shao, Rongxing Lu, *Senior Member, IEEE,* Yunguo Guan, and Guiyi Wei

***Abstract*—Due to the high demands of searchability over encrypted data, searchable encryption (SE) has recently received considerable attention and been widely suggested in encrypted cloud storage. Typically, the cloud server is assumed to be honest-but-curious in most SE-based cloud storage systems, i.e., the cloud server should follow the protocol to return valid and complete search results to users. However, this trust assumption is not always true due to some unanticipated situations, such as misconfigurations and malfunctions. Therefore, the function of verifiability of search results becomes crucial for the success of SE-based cloud storage systems. For this reason, many verifiable SE schemes have been proposed; however, they either fail to support query operators "OR", "AND", "$*$" and "?" simultaneously, or require many time-consuming operations. Aiming at addressing this problem, in this paper, we propose a new verifiable SE scheme for encrypted cloud storage. The proposed scheme is characterized by integrating various techniques, i.e., bitmap index, radix tree, format preserving encryption, keyed-hash message authentication code and symmetric key encryption, for achieving efficient and verifiable conjunctive and fuzzy queries over encrypted data in the cloud. Detailed security analysis shows that our proposed scheme holds the confidentiality of data and verifiability of search results at the same time. In addition, extensive experiments are conducted, and the results demonstrate our proposed scheme is efficient and suitable for users to retrieve their data from the cloud to their mobile devices.**

***Index Terms*—Cloud Storage, Verifiable Searchability, Searchable Encryption, Conjunctive Query, Fuzzy Query**

I. Introduction

Cloud storage has nowadays become one of the most popular data storage solutions, where users can outsource their data to the cloud server for the low cost and convenient access. According to the report from Research and Markets [1], cloud storage market in 2017 is US$25.171 billion, and it will reach US$92.488 billion by 2022 at a compound annual growth rate of 29.73% from 2017 to 2022. However, the privacy and security issues are still the main challenges concerned in cloud storage. The cloud storage data breaches happened from time to time due to attacks, malfunctions or misconfigurations, such as Apple iCloud celebrity leak [2], Dropbox password leak [3] and medical data leak on Amazon [4]. In this case, it would be wise to encrypt the data before uploading them to the cloud server. Meanwhile, in order to efficiently retrieve the data on demand, an encrypted index should be associated with the underlying encryption scheme, which results in the so-called searchable encryption (SE). In most SE-based cloud storage systems [5], [6], the cloud server is assumed to be honest-but-curious. That is, the cloud server would exactly follow the specific protocol. Under this trust assumption, everything goes well. For example, the user issues (encrypted) search query to the cloud server, and the latter will return the valid and complete search results to the user. However, this trust assumption is not always true in reality. In other words, the cloud server would return an invalid or incomplete search result to the user, due to the missing of the data [7], [8] or monetary reasons. In this regard, the verifiability of the search result in the cloud storage becomes a requirement.

Chai and Gong [9] firstly investigated the verifiability problem in searchable encryption. Since then, many research efforts have been dedicated to design verifiable SE in symmetric key setting [10]–[15] or public key setting [16]–[22]. However, almost all existing verifiable SE schemes suffer from at least one of the following disadvantages: i) some schemes lack the functionality simultaneously supporting query operators "`OR`", "`AND`", "$*$" and "?", which are considered as the basic searchability of the traditional database system; ii) other schemes supporting the above four kinds of query operators usually require some time-consuming operations, such as bilinear maps, which is against the trend that the resource-constrained devices, such as smart phones, are becoming the main devices for people to access internet [23].

In this paper, aiming at solving the above challenges, we would like to propose an efficient verifiable SE scheme supporting query operators "`OR`", "`AND`", "$*$" and "?" simultaneously. The proposed scheme is characterized by employing techniques bitmap index [24] and radix tree [25] to support query operators "`AND`", and "$*$" and "?". In addition, to provide the function of verifiability, the proposed scheme also applies keyed-hash message authentication code (HMAC) [26] to guarantee the integrity and authentication, and the format preserving encryption (FPE) [27], [28] for constructing the encrypted radix tree. Specifically, the main contributions of this paper are three-fold.

- By uniquely integrating the bitmap index, radix tree, FPE, and HMAC, we propose a new verifiable SE scheme that simultaneously supports query operators "`OR`", "`AND`", "$*$" and "?".
- Our proposed scheme is also efficient in terms of computational cost. In particular, the most time-consuming operations in our proposal are just the computations of

J. Shao, Y. Guan, and G. Wei are with the Department of Information Security, Zhejiang Gongshang University, Hangzhou 310018, China. E-mail: chn.junshao@gmail.com

R. Lu and J. Shao are with the Faculty of Computer Science, University of New Brunswick, Fredericton, Canada E3B 5A3. E-mail: rlu1@unb.ca

This work was supported by the Natural Science Foundation of Zhejiang Province [grant number LZ18F020003], the National Natural Science Foundation of China [grant number U1709217], and CSC Scholarship.

hash function and symmetric key encryption, which are usually considered as lightweight operations.

- We also implement a prototype to evaluate our proposed scheme, and the results demonstrate our proposed scheme is efficient in search and verification in terms of the computational cost.

The remainder of this paper is organized as follows. In Section II, we formalize the system model and security model, and identify our design goals. Then, we give some preliminaries including format preserving encryption and radix tree in Section III. In Section IV, we present the details of our proposed scheme, followed by the security analysis and performance evaluation in Section V and Section VI, respectively. Section VII reviews the related works. In the end, Section VIII gives the conclusions of our paper.

II. Models and Design Goals

In this section, we formalize our system model and security model, and identify our design goals.

A. System Model

In our system model, we mainly consider a typical single-user-and-cloud scenario, which includes two entities, namely a cloud user CU and a cloud server CS, as shown in Fig. 1.

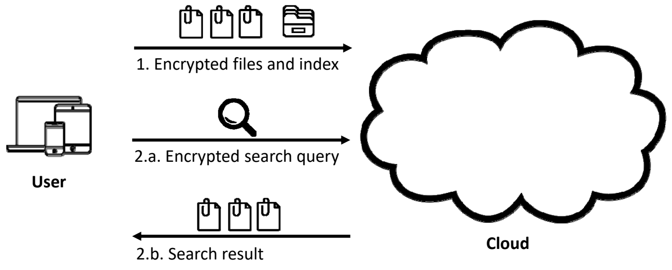
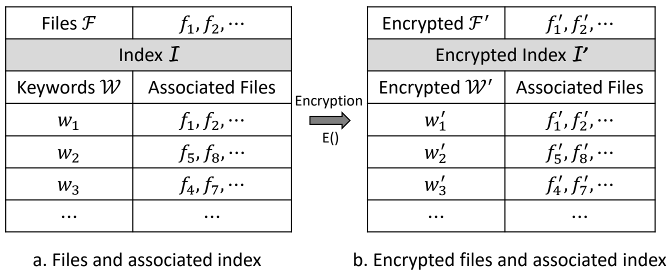


Fig. 1. The system model under consideration

CS is powerful in both storage and computing, while CU, which owns multiple terminals, e.g., laptop at home and smart phone outside, is less powerful in storage. Therefore, CU is willing to outsource his/her files to the cloud, so that he/she can later access those outsourced files anywhere and anytime via either laptop or smart phone. Outsourcing massive data to cloud is a good strategy for CU. However, since CU's data may be very sensitive, and CS cannot be fully trusted, CU has to encrypt data before outsourcing them to the cloud. For example, before outsourcing a set of files $\mathcal{F} = \{f_1, f_2, \cdots\}$ and the corresponding index $\mathcal{I}$, each entry of $\mathcal{I}$ stores a keyword and those files associated with the keyword, as shown in Fig. 2(a), CU needs to encrypt them into encrypted files and index, as shown in Fig.2(b). Later, when CU launches a query including one or many of operators "OR", "AND", "$*$" and "?", CS will return valid and complete search results, i.e., all encrypted files which match the issued query will be returned.

B. Security Model

In our security model, we no longer assume CS is honest or honest-but-curious as in many other cloud storage systems



Files $\mathcal{F}$	$f_1, f_2, \cdots$
Index $\mathcal{I}$	
Keywords $\mathcal{W}$	Associated Files
w_1	$f_1, f_2, \cdots$
w_2	$f_5, f_8, \cdots$
w_3	$f_4, f_7, \cdots$
$\cdots$	$\cdots$

a. Files and associated index

Encrypted $\mathcal{F}'$	$f_1', f_2', \cdots$
Encrypted Index $\mathcal{I}'$	
Encrypted $\mathcal{W}'$	Associated Files
w_1'	$f_1', f_2', \cdots$
w_2'	$f_5', f_8', \cdots$
w_3'	$f_4', f_7', \cdots$
$\cdots$	$\cdots$

b. Encrypted files and associated index

Fig. 2. Transform files and index into encrypted ones before outsourcing

[29], [30]. Instead, we consider CS not only has the interests in identifying the contents of files and keys uploaded by CU; but also sometimes maliciously returns incorrect or incomplete results when CU launches a query as mentioned before. Under this assumption, when CU receives the search result from CS, CU should have the ability to verify whether the result is valid and complete.

Note that, we assume CU faithfully follows the protocol, i.e., he/she will upload the valid encrypted files and index to CS, and will not frame CS for returning invalid or incomplete search results. In addition, we consider CU is *memoryless* with respect to the uploaded files and keywords, i.e., he/she will not maintain a local copy of them, but just keep some secret keys used for encrypting/verifying these files and keywords.

C. Design Goals

Based on the above-mentioned in system model and security model, our design goal is to develop a new verifiable SE scheme for cloud storage to support verifiability and complex search queries simultaneously. Specifically, the following desirable properties should be achieved.

- *Confidentiality of data*: The prominent security requirement of the proposed verifiable SE is the data confidentiality. In particular, except CU who outsourced the data to the cloud, anyone else cannot gain the data content from the encrypted files, encrypted keywords or the interactions with CU.
- *Verifiability of search results*: When CS is considered as malicious, the verifiability of the query data becomes quite critical in the cloud storage. Hence, the proposed verifiable SE scheme should enable CU to check the validity and completeness of returned data from CS. In particular, CU should be able to check whether all returned files satisfying the query conditions, and whether all encrypted files in cloud satisfying the query conditions are returned.
- *Supporting complex queries*: The proposed verifiable SE scheme should allow CU to launch various queries containing one or many of operators "OR", "AND", "$*$" and "?".
- *Efficiency for cloud users*: The proposed verifiable SE scheme should be efficient for CU in terms of computational cost. In particular, the time-consuming operations, such as exponentiation in finite cyclic groups should be avoided in our proposal.

III. Preliminaries

Before delving into the design of our proposed verifiable SE scheme, we first review some preliminaries, including format preserving encryption (FPE) [27] and radix tree [25].

A. Format Preserving Encryption

Format Preserving Encryption (FPE) has recently become a handy tool in cryptography [27], which can encrypt a plaintext of some specified format into a ciphertext of the same format. Typical examples of FPE include i) encrypting a 16-digit credit card number into a ciphertext with another 16-digit number; and ii) encrypting an English word into a ciphertext with another English word. Since FPE is one of building blocks of our design, we here recall a well-known FPE scheme in [31]. Given a secret key k_0 and a secure symmetric encryption algorithm $E()$, e.g., AES, the FPE scheme is described as follows:

- Given the capital letter set $\mathcal{M} = \{\text{'A'}, \text{'B'}, \cdots, \text{'Z'}\}$, order all letters in $\mathcal{M}$, i.e., $\texttt{ord}(\text{'A'}) = 0, \texttt{ord}(\text{'B'}) = 1, \cdots, \texttt{ord}(\text{'Z'}) = 25$. Note that, we can consider a more general set including not only capital letters, but also small letters, numbers, and special characters. For simplicity of description, we just take the capital letter set as an example here.
- For each letter $i \in \mathcal{M}$, use the secret key k_0 and $E()$ to compute its weight $\texttt{weight}(i) = E(k_0, i)$. Then, sort $\{\text{'A'}, \text{'B'}, \cdots, \text{'Z'}\}$ based on their calculated weights. For example, Fig. 3 shows one possible sorting result, which creates a random and private one-to-one mapping in $\mathcal{M}$.
- With the private mapping in Fig. 3, we can construct a FPE scheme. For example, given a word "CLOUD", we can encrypt it into a ciphertext "EDCBQ"; given a ciphertext "JIEBPWHT", we can also recover it as "SECURITY".

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕
M	S	E	Q	I	K	N	Y	W	F	A	D	O	R	C	G	L	P	J	H	B	X	Z	U	T	V

Fig. 3. Private one-to-one mapping based on weights calculated from $\texttt{weight}(i) = E(k_0, i)$, where $i \in \mathcal{M}$.

B. Radix Tree

Radix tree (also called radix trie) is a data structure which represents an order tree storing keywords [25]. The root node, leaf nodes and inner nodes of a radix tree are associated with special symbols \$, #, and normal symbols in the keywords, respectively. The special symbols \$ and # indicate the start and end of a keyword, respectively. We simply assume that \$ and # will never appear in any keyword; otherwise, we would choose other special symbols to indicate the start and end of a keyword. An example of a radix tree is given in Fig. 4(a), where keywords are "AH", "AN", "ANT", "AT", "BIG", "BIT" and "BITE".

Radix tree could be considered as a good candidate data structure for the hash table in search applications, since it does not have collision for different values like the hash table. It is also easy to see that once the radix tree is built, the time complexity for searching keyword w is $\mathcal{O}(|w|)$, where $|w|$ is the number of symbols in the keyword. In addition, the radix tree also enables us to achieve fuzzy queries (supporting operators "$*$" and "?"). When we apply the FPE technique on the radix tree, we can obtain an encrypted version of radix tree while still keeping its nice properties. For example, if we follow the private one-to-one mapping in Fig. 3, we can transform the radix tree in Fig. 4(a) into its encrypted version in Fig. 4(b). Note that, if the frequencies of letters in the original radix tree follow the English letter frequency, its encrypted version may suffer from the frequency analysis attack, i.e., some letters with high frequency will be identified. To resist against the frequency analysis attack, we can disturb the frequency by encrypting the same letter i into the different letters. The details can be found in Section V-A.

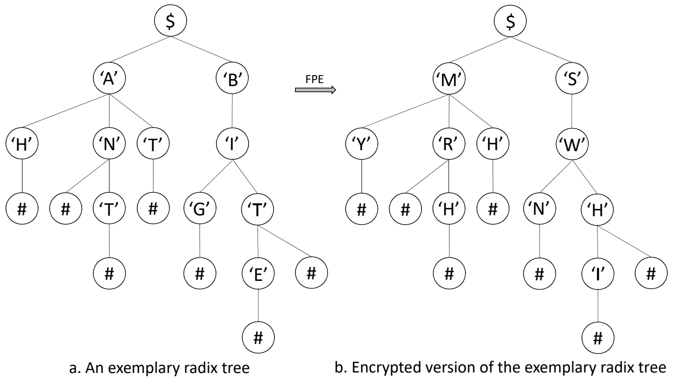

FPE
a. An exemplary radix tree
b. Encrypted version of the exemplary radix tree

Fig. 4. An example of radix tree with keywords "AH", "AN", "ANT", "AT", "BIG", "BIT" and "BITE" and its encrypted version with FPE technique.

IV. Our Proposed Verifiable Searchable Encryption for Cloud Storage

In this section, we present our new verifiable SE scheme, which mainly consists of the following four phases, namely User Preparation (`UserPrep`), Search Token Generation (`TokenGen`), Cloud Search (`CloudSearch`) and Verification & Decryption (`VerDec`). In `UserPrep` phase, the cloud user `CU` performs some preparations for cloud storage, including building the encrypted radix tree as the encrypted index, encrypting the files, and uploading the resultant data to the cloud server `CS`. Once `CU` wants to retrieve some data from `CS`, he/she could generate the search token for his/her search query in `TokenGen` phase. Upon receiving the search token from `CU` in `CloudSearch` phase, `CS` will find the encrypted files according to the search token, and return them to `CU` with the validity and completeness proof. Finally, in `VerDec` phase, `CU` will check the validity and completeness of the data received from `CS` and decrypt the encrypted files if the checking passes successfully. The details of them are described below.

A. Description of the Proposal

Our proposed verifiable SE scheme can support the search query containing one or more of operators "`OR`", "`AND`", "$*$",

"?". For simplicity, we only show four kinds of processes (search and verification) where queries contain only one of these four basic operators. It is easy to see that other processes can be easily extended from these four kinds of processes.

1) `UserPrep`: Assume that the files to be outsourced to `CS` are $\mathcal{F} = \{f_1, f_2, \cdots, f_{n_f}\}$, which have been sorted by some rules, such as the file size or created time. `CU` will run the following steps for the preparation.

- `CU` firstly chooses an `HMAC` scheme, an FPE scheme and a symmetric key encryption $E()$.
- `CU` also chooses three random keys k, k_0 and k_1, which will be used in `HMAC`, FPE and $E()$, respectively.
- After choosing the underlying algorithms and their associated keys, `CU` encrypts each file $f_i \in \mathcal{F}$ into $f'_i = E(k_1, f_i)$. We denote the resultant encrypted files as $\mathcal{F}' = \{f'_1, f'_2, \cdots, f'_{n_f}\}$.
- `CU` further extracts keywords $\mathcal{W} = \{w_1, w_2, \cdots, w_{n_w}\}$ from files $\mathcal{F} = \{f_1, f_2, \cdots, f_{n_f}\}$. The symbols composing of these keywords are $\{s_1, s_2, \cdots, s_{n_s}\}$.
- Based on the original keywords $\mathcal{W} = \{w_1, w_2, \cdots, w_{n_w}\}$, `CU` builds a corresponding radix tree. For example, when the keywords "AH", "AN", "ANT", "AT", "BIG", "BIT" and "BITE", the radix tree is shown in Fig. 4(a).
- `CU` obtains the encrypted symbols $\{s'_1, s'_2, \cdots s'_{n_s}\}$ by applying the underlying FPE scheme and k_0. Then, by replacing s_i with s'_i, `CU` can also get the encrypted keywords $\mathcal{W}' = \{w'_1, w'_2, \cdots, w'_{n_w}\}$. For example, by using the FPE scheme in Fig. 3, we can encrypt the keyword "BITE" into "SWHI". Furthermore, with the encrypted symbols, `CU` can transform the radix tree into its encrypted version, as shown in Fig. 4.
- `CU` adds a hash value h_{cp_i} into each node except the leaf nodes in the encrypted radix tree, where $h_{cp_i} = \texttt{HMAC}(k, \mathcal{A}_{p_i} || s'_{i_j}, \mathcal{S}'_{c_i})$, $\mathcal{A}_{p_i}$ is the encrypted symbol chain containing all the encrypted symbols in ancestor nodes of the current node, s'_{i_j} is the encrypted symbol in the current node, and $\mathcal{S}'_{c_i}$ is the encrypted symbol set containing all encrypted symbols in child nodes of the current node. Take the encrypted radix tree in Fig. 4(b) as an example, $\mathcal{A}_{p_i}$, s'_{i_j} and $\mathcal{S}'_{c_i}$ of the second left node in level two are \$"M", 'R' and $\{\#, \text{'H'}\}$, respectively. Regarding the root node of the radix tree, $\mathcal{A}_{p_r}$ is an empty encrypted symbol chain.

  The data structures of resultant root node and inner nodes are as shown in Fig. 5(a) and Fig. 5(b), respectively.

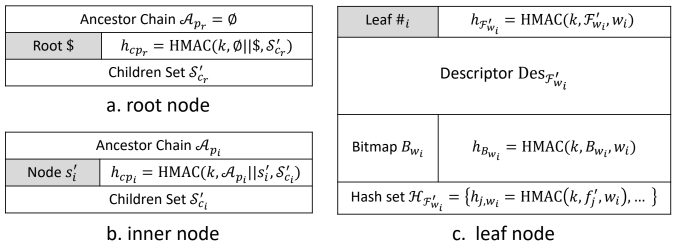

Ancestor Chain $\mathcal{A}_{p_r} = \emptyset$
Root \$
$h_{cp_r} = \text{HMAC}(k, \emptyset || \$, \mathcal{S}'_{c_r})$
Children Set $\mathcal{S}'_{c_r}$
a. root node
Ancestor Chain $\mathcal{A}_{p_i}$
Node s'_i
$h_{cp_i} = \text{HMAC}(k, \mathcal{A}_{p_i} || s'_i, \mathcal{S}'_{c_i})$
Children Set $\mathcal{S}'_{c_i}$
b. inner node
Leaf $\#_i$
$h_{\mathcal{F}'_{w_i}} = \text{HMAC}(k, \mathcal{F}'_{w_i}, w_i)$
Descriptor $\text{Des}_{\mathcal{F}'_{w_i}}$
Bitmap B_{w_i}
$h_{B_{w_i}} = \text{HMAC}(k, B_{w_i}, w_i)$
Hash set $\mathcal{H}_{\mathcal{F}'_{w_i}} = \{h_{j,w_i} = \text{HMAC}(k, f'_j, w_i), \ldots\}$
c. leaf node

Fig. 5. Data structures of nodes in the resultant encrypted radix tree

- Furthermore, `CU` adds the following values into each leaf node. Assume that the keyword corresponding to the current leaf node is w_i.
  - $h_{\mathcal{F}'_{w_i}}$: It is a hash value computed by $h_{\mathcal{F}'_{w_i}} = \texttt{HMAC}(k, \mathcal{F}'_{w_i}, w_i)$, where $\mathcal{F}'_{w_i}$ is the set of all encrypted files whose corresponding (original) files contain the keyword w_i.
  - $\text{Des}_{\mathcal{F}'_{w_i}}$: It is a descriptor of encrypted files in the set $\mathcal{F}'_{w_i}$. Note that the descriptor is just used for locating the encrypted files, and no information about the file content is involved.
  - B_{w_i}: It is an n_f-bit length number used to indicate which files in $\mathcal{F}$ contain the keyword w_i by using the bitmap index technique [24]. Recall all files are sorted, if file f_j contains the keyword w_i, the j-th bit of B_{w_i} is set to 1; and 0 otherwise. In the rest of this paper, we use $B_{w_i}[j]$ to indicate that the j-th bit of B_{w_i}.
  - $h_{B_{w_i}}$: It is a hash value computed by $h_{B_{w_i}} = \texttt{HMAC}(k, B_{w_i}, w_i)$.
  - $\mathcal{H}_{\mathcal{F}'_{w_i}}$: For each $f'_j \in \mathcal{F}'_{w_i}$, `CU` computes its hash value $h_{j,w_i} = \texttt{HMAC}(k, f'_j, w_i)$. The resultant hash values form the hash value set $\mathcal{H}_{\mathcal{F}'_{w_i}}$.

  The data structure of the resultant leaf nodes can be described in Fig. 5(c).
- Finally, `CU` sends all the encrypted files $\mathcal{F}'$ and the resultant encrypted radix tree T to `CS`.

2) `TokenGen`: To protect the privacy of keywords, `CU` in this phase needs to encrypt all the symbols except the operators in search query by using the underlying FPE scheme. For example, the query $w_i = s_{i,1}s_{i,2}\cdots s_{i,t_i}*$ will be encrypted as $w'_i = s'_{i,1}s'_{i,2}\cdots s'_{i,t_i}*$. We call the obtained result after encryption as the search token. In the end, `CU` sends the resultant search token to `CS`.

3) `CloudSearch`: In this phase, we show how `CS` responds the queries containing "`OR`", "`AND`", "$*$" or "?" one by one.

a) Operator "`OR`": Upon receiving the search token $w'_{i_1} \bigvee w'_{i_2}$ from `CU`, where $w'_{i_1} = s'_{i_1,1}s'_{i_1,2}\cdots s'_{i_1,t_1}$ and $w'_{i_1} = s'_{i_2,1}s'_{i_2,2}\cdots s'_{i_1,t_2}$, `CS` firstly searches w'_{i_1} and w'_{i_2} in the encrypted radix tree. Assume that $w'_{i_1,s} = s'_{i_1,1}s'_{i_1,2}\cdots s'_{i_1,\ell_1^*}$, ($\ell_1^* \le t_1$) and $w'_{i_2,s} = s_{i_2,1}s_{i_2,2}\cdots s_{i_2,\ell_2^*}$, ($\ell_2^* \le t_2$) are the longest substrings found in the encrypted radix tree for w'_{i_1} and w'_{i_2}, respectively. After that, `CS` does the following steps.

- For each $j \in \{1, 2\}$, if $\ell_j^* < t_j$ (case 1), or $\ell_j^* = t_j$ while none of the child nodes of the node corresponding to $w'_{i_j,s}$ is a leaf node (case 2), we know that w'_{i_j} cannot be found in the encrypted radix tree. Then, `CS` simply extracts $(h_{cp_{\ell_j^*}}, \mathcal{S}'_{c_{\ell_j^*}})$ from the node corresponding to $w'_{i_j,s}$.
- For each $j \in \{1, 2\}$, if $\ell_j^* = t_j$, and one of the child nodes of the node corresponding to $w'_{i_j,s}$ is a leaf node, we know that w'_{i_j} is found in the encrypted radix tree. `CS` extracts $\text{Des}_{\mathcal{F}'_{w_{i_j}}}$ and $h_{\mathcal{F}'_{w_{i_j}}}$ from the leaf node corresponding to w'_{i_j} in the encrypted radix tree. Furthermore, `CS` obtains the corresponding encrypted file set $\mathcal{F}'_{w_{i_j}}$ according to $\text{Des}_{\mathcal{F}'_{w_{i_j}}}$.

Eventually, CS sends the obtained $(\ell_j^*, h_{cp_{\ell_j^*}}, \mathcal{S}'_{c_{\ell_j^*}})$ and $(\mathcal{F}'_{w_{i_j}}, h_{\mathcal{F}'_{w_{i_j}}})$ to CU as the search result.

Note that $(\ell_j^*, h_{cp_{\ell_j^*}}, \mathcal{S}'_{c_{\ell_j^*}})$ is the proof of that no file satisfies keyword w_{i_j}, and $h_{\mathcal{F}'_{w_{i_j}}}$ is the proof of that all the encrypted files in set $\mathcal{F}'_{w_{i_j}}$ satisfy the keyword w_{i_j}.

b) Operator "AND": Upon receiving the search token $w'_{i_1} \bigwedge w'_{i_2}$ from CU, where $w'_{i_1} = s'_{i_1,1}s'_{i_1,2}\cdots s'_{i_1,t_1}$ and $w'_{i_1} = s'_{i_2,1}s'_{i_2,2}\cdots s'_{i_1,t_2}$, CS firstly searches w'_{i_1} and w'_{i_2} in the encrypted radix tree. Like in the case of operator "OR", we also assume that $w'_{i_1,s} = s'_{i_1,1}s'_{i_1,2}\cdots s'_{i_1,\ell_1^*}$, $(\ell_1^* \le t_1)$ and $w'_{i_2,s} = s_{i_2,1}s_{i_2,2}\cdots s_{i_2,\ell_2^*}$, $(\ell_2^* \le t_2)$ are the longest substrings found in the encrypted radix tree for w'_{i_1} and w'_{i_2}, respectively. After that, CS does the following steps.

- For $j \in \{1, 2\}$, if $\ell_j^* < t_j$ (case 1), or $\ell_j^* = t_j$ while none of the child nodes of the node corresponding to $w'_{i_j,s}$ is a leaf node (case 2), we know that w'_{i_j} can not be found in the encrypted radix tree, and no file satisfies the search query. At this point, CS simply extracts $h_{cp_{\ell_j^*}}$ and $\mathcal{S}'_{c_{\ell_j^*}}$ from the node corresponding to $w'_{i_j,s}$, returns $(\ell_j^*, h_{cp_{\ell_j^*}}, \mathcal{S}'_{c_{\ell_j^*}})$ to CU as the search result, and aborts this phase.
  Note that $(\ell_j^*, h_{cp_{\ell_j^*}}, \mathcal{S}'_{c_{\ell_j^*}})$ is the proof of that no file satisfies the search query.
- If for all $j \in \{1, 2\}$, we have that $\ell_j^* = t_j$, and one of the child nodes of node w_{i_j} is a leaf node, we know that both w'_{i_1} and w'_{i_2} are found in the encrypted radix tree. CS continues to do the following steps.
  - CS extracts $(\text{Des}_{\mathcal{F}'_{w_{i_1}}}, B_{w_{i_1}}, h_{B_{w_{i_1}}}, \mathcal{H}_{\mathcal{F}'_{w_{i_1}}})$ and $(\text{Des}_{\mathcal{F}'_{w_{i_2}}}, B_{w_{i_2}}, h_{B_{w_{i_2}}}, \mathcal{H}_{\mathcal{F}'_{w_{i_2}}})$ from the leaf nodes in the encrypted radix tree corresponding to w'_{i_1} and w'_{i_2}, respectively.
  - After the extraction, CS computes $B_i = B_{w_{i_1}} \& B_{w_{i_2}}$, where $\&$ is the bit AND operator. If $B_i[\ell] = 1$, then file f_ℓ satisfies the search query $w_{i_1} \bigwedge w_{i_2}$. We denote $\mathcal{F}'_{B_i}$ as the encrypted file set where the encrypted files are selected according to the bit position with 1 in B_i.
  - After obtaining the files, CS needs to get the corresponding values that show the files satisfy the search query. Specifically, for all $j \in \{1, 2\}$, CS obtains a hash value set $\mathcal{H}_{\mathcal{F}'_{w_{i_j}}, B_i} = \{h_{\ell, w_{i_j}} | B_i[\ell] = 1, 1 \le \ell \le n_f\}$.
  - Lastly, the cloud server CS sends $\mathcal{F}'_{B_i}$ and $\{B_{w_{i_j}}, h_{B_{w_{i_j}}}, \mathcal{H}_{\mathcal{F}'_{w_{i_j}}, B_i}\}_{j=1}^2$ to CU as the search result.

  Note that $\{B_{w_{i_j}}, h_{B_{w_{i_j}}}\}_{j=1}^2$ is the proof of that how many files satisfy the search query, and $\mathcal{H}_{\mathcal{F}'_{w_{i_j}}, B_i}$ is the proof of that all the files corresponding to the encrypted files in $\mathcal{F}'_{B_i}$ contain keyword w_{i_j}.

c) Operator "$*$": Upon receiving the search token $w'_i = s'_{i,1}s'_{i,2}\cdots s'_{i,t}*$ from CU, CS searches $s'_{i,1}s'_{i,2}\cdots s'_{i,t}$ in the encrypted radix tree. Assume that $w'_{i,s} = s'_{i,1}s'_{i,2}\cdots s'_{i,\ell^*}$, $(\ell^* \le t)$ is the longest substring found in the encrypted radix tree for $s'_{i,1}s'_{i,2}\cdots s'_{i,t}$. CS does the following steps.

- If $\ell^* < t$, we now that there is no file satisfying the search token w'_i. In this case, CS extracts $h_{cp_{\ell^*}}$ and $\mathcal{S}'_{c_{\ell^*}}$ from the node corresponding to $w'_{i,s}$, returns $(\ell^*, h_{cp_{\ell^*}}, \mathcal{S}'_{c_{\ell^*}})$ to CU as the search result, and aborts this phase.
  Note that $(\ell_j^*, h_{cp_{\ell_j^*}}, \mathcal{S}'_{c_{\ell_j^*}})$ is the proof of that no file satisfies the search query.
- If $\ell^* = t$, we know that there exist files satisfying the search query w_i. CS continues to do the following steps.
  - CS extracts a subtree $\bar{T}$ from the encrypted radix tree, and the node corresponding to $w'_{i,s}$ is the root node of the subtree $\bar{T}$.
  - After that, CS further extracts $(\text{Des}_{\mathcal{F}'_{w_{i_j}}}, h_{\mathcal{F}'_{w_{i_j}}})$'s from the leaf nodes of the subtree $\bar{T}$.
  - According to $\text{Des}_{\mathcal{F}'_{w_{i_j}}}$'s, CS obtains the corresponding encrypted file sets $\mathcal{F}'_{w_{i_j}}$'s.
  - Ultimately, CS sends $\mathcal{F}'_{w_{i_j}}$'s as well as a subtree $\tilde{T}$ to CU as the search result, where $\tilde{T}$ is almost the same as $\bar{T}$, except that the leaf nodes in $\tilde{T}$ only contain the symbol $\#$ and $h_{\mathcal{F}'_{w_{i_j}}}$'s.

  Note that the subtree $\tilde{T}$, especially the hash values in the subtree $\tilde{T}$, is the proof of that all the files corresponding to the encrypted files in $\mathcal{F}'_{w_{i_j}}$'s satisfying the search query.

d) Operator "?": Upon receiving the search token $w'_i = s'_{i,1}\cdots s'_{i,j}?s'_{i,j+2}\cdots s'_{i,t}$ from CU, CS searches $s'_{i,1}s'_{i,2}\cdots s'_{i,j}$ in the encrypted radix tree. Assume that $w'_{i,s} = s'_{i,1}s'_{i,2}\cdots s'_{i,\ell^*}$, $(\ell^* \le j)$ is the longest substring found in the encrypted radix tree for $s'_{i,1}s'_{i,2}\cdots s'_{i,j}$. CS does the following steps.

- If $\ell^* < j$, we have that there is no file satisfying the search token w'_i. CS directly returns $(\ell^*, h_{cp_{\ell^*}}, \mathcal{S}'_{c_{\ell^*}})$ to CU as the search result, and aborts this phase.
  Note that $(\ell^*, h_{cp_{\ell^*}}, \mathcal{S}'_{c_{\ell^*}})$ is the proof of that no file satisfies the search query.
- If $\ell^* = j$, CS continues to do the following steps.
  - CS extracts h_{cp_j} and $\mathcal{S}'_{c_j}$ from the node corresponding to $w'_{i,s}$.
  - CS further extracts one subtree for each child node of the node corresponding to $w'_{i,s}$ from the encrypted radix tree by treating each child node as the root node of the associated extracted subtree. The resultant extracted subtrees consist of a subtree set $\mathcal{T}_{w'_{i,s}}$.
  - For each subtree in $\mathcal{T}_{w'_{i,s}}$, CS checks whether $s'_{i,j+2}s'_{i,j+3}\cdots s'_{i,t}$ exists in the corresponding subtree. Assume that the encrypted symbol in the root node of the current subtree is $s'_{i_{\bar{j}}}$, and $w'_{i,\bar{s}} = s'_{i,j+2}s'_{i,j+3}\cdots s'_{i,\bar{\ell}^*}$, $(\bar{\ell}^* \le t)$ is the longest substring found in the current subtree for $s'_{i,j+2}s'_{i,j+3}\cdots s'_{i,t}$. Note that during the search process, the root node of the subtree is never compared as that in the encrypted radix tree.
    - If $\bar{\ell}^* < t$ (case 1), or $\bar{\ell}^* = t$ while none of child nodes of the node corresponding to $w'_{i,\bar{s}}$ is a leaf node (case 2), we know that $s'_{i,j+2}s'_{i,j+3}\cdots s'_{i,t}$ is not found in the subtree. CS extracts $h_{cp_{\bar{\ell}^*}}$ and $\mathcal{S}'_{c_{\bar{\ell}^*}}$ from the node corresponding to $w'_{i,\bar{s}}$.

    * If $\bar{\ell}^* = t$, and one of child nodes of the node corresponding to $w'_{i,\bar{s}}$ is a leaf node, we know that $s'_{i,j+2}s'_{i,j+3}\cdots s'_{i,t}$ is found in the subtree. CS extracts $\text{Des}_{\mathcal{F}'_{w_{i_{\bar{j}}}}}$ and $h_{\mathcal{F}'_{w_{i_{\bar{j}}}}}$ from the corresponding leaf node of the subtree. According to $\text{Des}_{\mathcal{F}'_{w_{i_{\bar{j}}}}}$, CS obtains the corresponding encrypted file set $\mathcal{F}'_{w_{i_{\bar{j}}}}$.

  – CS sends the obtained $(s'_{i_{\bar{j}}}, \bar{\ell}^*, h_{cp_{\bar{\ell}^*}}, \mathcal{S}'_{c_{\bar{\ell}^*}})$'s, $(s'_{i_{\bar{j}}}, \mathcal{F}'_{w_{i_{\bar{j}}}}, h_{\mathcal{F}'_{w_{i_{\bar{j}}}}})$'s, and h_{cp_j} to CU as the search result.

Note that h_{cp_j} is the proof of that all the possible encrypted symbols for operator "?" in the radix tree are listed in the search result, $(s'_{i_{\bar{j}}}, \bar{\ell}^*, h_{cp_{\bar{\ell}^*}}, \mathcal{S}'_{c_{\bar{\ell}^*}})$ is the proof of that there is no satisfied file if operator "?" is replaced by $s'_{i_{\bar{j}}}$, and $h_{\mathcal{F}'_{w_{i_{\bar{j}}}}}$ is the proof of that all the encrypted files in $\mathcal{F}'_{w_{i_{\bar{j}}}}$ satisfy the keyword when operator "?" is replaced by $s'_{i_{\bar{j}}}$.

4) `VerDec`: As in `CloudSearch` phase, we only give the processes of the basic four operators in `VerDec` phase. We also imply that if the validity and completeness checking pass successfully, CU would use k_1 to decrypt the received encrypted files. The detailed process of decryption is omitted in the following description.

a) Operator "`OR`*":* On receiving the search result from CS for the search token $w'_{i_1} \bigvee w'_{i_2}$, where $w'_{i_1} = s'_{i_1,1}s'_{i_1,2}\cdots s'_{i_1,t_1}$ and $w'_{i_1} = s'_{i_2,1}s'_{i_2,2}\cdots s'_{i_1,t_2}$, CU checks the validity and completeness as follows.

For the part $(\ell^*_j, h_{cp_{\ell^*_j}}, \mathcal{S}'_{c_{\ell^*_j}})$, CU checks whether

$$s'_{i_j,\ell^*_j+1} \notin \mathcal{S}'_{c_{\ell^*_j}} \tag{1}$$

and

$$h_{cp_{\ell^*_j}} = \texttt{HMAC}(k, \$||s'_{i_j,1}||\cdots||s'_{i_j,\ell^*_j}, \mathcal{S}'_{c_{\ell^*_j}}) \tag{2}$$

Note that if $\ell^*_j = t_j$, then we set $s'_{i_j,\ell^*_j+1} = \#$.

For the part $(\mathcal{F}'_{w_{i_j}}, h_{\mathcal{F}'_{w_{i_j}}})$, CU just checks whether $h_{\mathcal{F}'_{w_{i_j}}} = \texttt{HMAC}(k, \mathcal{F}'_{w_{i_j}}, w_{i_j})$ holds.

If all of above conditions are satisfied, then the returned search result is valid and complete; otherwise, it is invalid or incomplete.

b) Operator "`AND`*":* On receiving the search result from CS for the search token $w'_{i_1} \bigwedge w'_{i_2}$ from CU, where $w'_{i_1} = s'_{i_1,1}s'_{i_1,2}\cdots s'_{i_1,t_1}$ and $w'_{i_1} = s'_{i_2,1}s'_{i_2,2}\cdots s'_{i_1,t_2}$, CU can perform the checking process as follows.

- If the result is $(\ell^*_j, h_{cp_{\ell^*_j}}, \mathcal{S}'_{c_{\ell^*_j}})$, CU checks whether conditions (1) and (2) are satisfied. If both of them are satisfied, then the returned search result is valid and complete (there is indeed no file satisfying the query); otherwise, it is incomplete.
- If the result is $\mathcal{F}'_{B_i}$ and $\{B_{w_{i_j}}, h_{B_{w_{i_j}}}, \mathcal{H}_{\mathcal{F}'_{w_{i_j}},B_i}\}_{j=1}^2$, CU does the following steps.
  – Check whether $h_{B_{w_{i_j}}} = \texttt{HMAC}(k, B_{w_{i_j}}, w_{i_j})$ holds for all $j \in \{1, 2\}$. If one of them does not hold, the search result is invalid. Otherwise, do the next step.
  – Compute $B_i = B_{w_{i_1}} \& B_{w_{i_2}}$, and check whether the size of $\mathcal{F}'_{B_i}$ equals the number of bits with 1 in B_i. If it is not, the search result is incomplete. Otherwise, do the next step.
  – Check whether $h_{\ell,w_{i_j}} = \texttt{HMAC}(k, f'_\ell, w_{i_j})$ holds for all $h_{\ell,w_{i_j}} \in \mathcal{H}_{\mathcal{F}'_{w_{i_j}},B_i}$ and all $j \in \{1, 2\}$. If one of the equalities does not hold, the search result is invalid; otherwise, the search result is valid and complete.

c) Operator "$*$": On receiving the search result from CS for the search token $w'_i = s'_{i,1}s'_{i,2}\cdots s'_{i,t}*$, CU can perform the checking process as follows.

- If the result is $(\ell^*, h_{cp_{\ell^*}}, \mathcal{S}'_{c_{\ell^*}})$, CU checks whether $h_{cp_{\ell^*}} = \texttt{HMAC}(k, \$||s'_{i,1}||\cdots||s'_{i,\ell^*}, \mathcal{S}'_{c_{\ell^*}})$ and $s'_{i,\ell^*+1} \notin \mathcal{S}'_{c_{\ell^*}}$. If both of the above conditions are satisfied, then the returned search result is valid and complete (there is indeed no file satisfying the query); otherwise, it is incomplete.
- If the result is $\mathcal{F}'_{w_{i_j}}$'s and subtree $\tilde{T}$, CU does the following checking.
  – Check whether $h_{cp_\ell} = \texttt{HMAC}(k, \$||\mathcal{A}_{p_\ell}||s'_\ell, \mathcal{S}'_{c_\ell})$ holds for every node except the leaf nodes in subtree $\tilde{T}$.
  – Check whether $h_{\mathcal{F}'_{w_{i_\ell}}} = \texttt{HMAC}(k, \mathcal{F}'_{w_{i_\ell}}, w_{i_\ell})$ holds for every leaf node in subtree $\tilde{T}$.

  If all the above equalities hold, then the returned search result is valid and complete; otherwise, it is invalid or incomplete.

d) Operator "?": On receiving the search result from CS for the search token $w'_i = s'_{i,1}\cdots s'_{i,j}?s'_{i,j+2}\cdots s'_{i,t}$, CU can perform the checking process as follows.

- If the search result is $(\ell^*, h_{cp_{\ell^*}}, \mathcal{S}'_{c_{\ell^*}})$, CU can do the checking as that in the case of operator "$*$".
- If the search result is $(s'_{i_{\bar{j}}}, \bar{\ell}^*, h_{cp_{\bar{\ell}^*}}, \mathcal{S}'_{c_{\bar{\ell}^*}})$'s, $(s'_{i_{\bar{j}}}, \mathcal{F}'_{w_{i_{\bar{j}}}}, h_{\mathcal{F}'_{w_{i_{\bar{j}}}}})$'s and h_{cp_j}, CU does the following steps.
  – Check whether the equality $h_{cp_j} = \texttt{HMAC}(k, \$||s'_{i,1}||\cdots||s'_{i_j}, \{s'_{i_{\bar{j}}}\})$ holds. If it does not hold, the search result is invalid or incomplete; otherwise, do the next step.
  – For the part $(s'_{i_{\bar{j}}}, \bar{\ell}^*, h_{cp_{\bar{\ell}^*}}, \mathcal{S}'_{c_{\bar{\ell}^*}})$, CU checks whether $h_{cp_{\bar{\ell}^*}} = \texttt{HMAC}(k, \$||s'_{i,1}||\cdots||s'_{i,\bar{\ell}^*}, \mathcal{S}'_{c_{\ell^*}})$ and $s'_{i,\bar{\ell}^*+1} \notin \mathcal{S}'_{c_{\bar{\ell}^*}}$ are satisfied. Note that if $\bar{\ell}^* = t$, then we set $s'_{i,\bar{\ell}^*+1} = \#$.
  For the part $(s'_{i_{\bar{j}}}, \mathcal{F}'_{w_{i_{\bar{j}}}}, h_{\mathcal{F}'_{w_{i_{\bar{j}}}}})$, CU checks whether $h_{\mathcal{F}'_{w_{i_{\bar{j}}}}} = \texttt{HMAC}(k, \mathcal{F}'_{w_{i_{\bar{j}}}}, w_{i_{\bar{j}}})$ holds, where $w_{i_{\bar{j}}}$ is the keyword corresponding to $w'_{i_{\bar{j}}} = s'_{i,1}\cdots s'_{i,j}s'_{i_{\bar{j}}}s'_{i,j+2}\cdots s'_{i,t}$.

  If all of the above checking steps pass successfully, then the returned search result is valid and complete; otherwise, it is invalid or incomplete.

B. A Toy Example

In this section, we will give a toy example to further clarify our proposed verifiable SE scheme. Assume that there are seven keywords {"AH", "AN", "ANT", "AT", "BIG", "BIT", "BITE"} and eight files $\{f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8\}$ in our example. The relationship between keywords and files is given in Table I. Furthermore, the underlying FPE scheme in the example is the one in Fig. 3, and the corresponding encrypted radix tree is constructed as shown in Fig. 6. Note that the hash values in Fig. 6 are not computed by the real HMAC but just manipulated for illustration. Now we are ready to give the query examples.

Keyword	"AH"	"AN"	"ANT"	"AT"	"BIG"	"BIT"	"BITE"
File	f_1, f_4, f_5, f_8	f_2	f_1, f_4	f_1, f_3, f_7	f_2, f_6	f_5, f_8	f_1

TABLE I. The files and associated keywords in the toy example

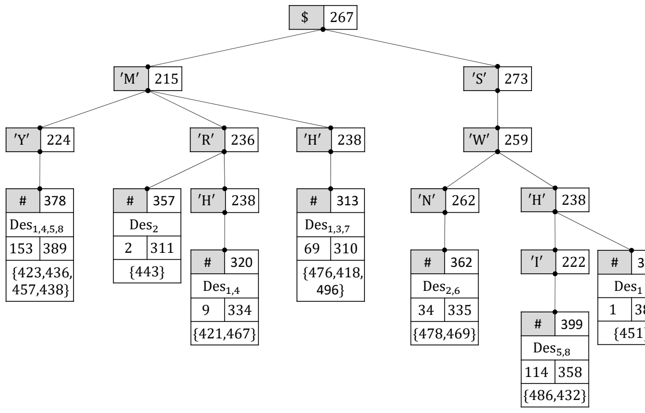


Fig. 6. The encrypted radix tree of the toy example

1) Operator "`OR`*":* Assume that the search query is $w_{i_1} \bigvee w_{i_2}$, where w_{i_1} = "SEC" and w_{i_2} = "BIT", the corresponding search token is $w'_{i_1} \bigvee w'_{i_2}$ = "JIE" $\bigvee$ "SWH".

For w'_{i_1} = "JIE", CS cannot find any child node of the root node in the encrypted radix tree equal to 'J', hence it obtains $(\ell_1^*, h_{cp_{\ell_1^*}}, \mathcal{S}'_{c_{\ell_1^*}}) = (0, 267, \{'\mathrm{M}', '\mathrm{S}'\})$. For w'_{i_2} = "SWH", CS can find it in the encrypted radix tree, hence it obtains $(\mathcal{F}_{w_{i_2}}, h_{\mathcal{F}'_{w_{i_2}}}) = (\{f'_1\}, 327)$. After the above steps, CS sends $(0, 267, \{'\mathrm{M}', '\mathrm{S}'\})$ and $(\{f'_1\}, 327)$ to the CU as the search result.

To check the validity and completeness of the search result, CU checks whether $'\mathrm{J}' \notin \{'\mathrm{M}', '\mathrm{S}'\}$, $267 = \mathtt{HMAC}(k, \$, \{'\mathrm{M}', '\mathrm{S}'\})$, and $327 = \mathtt{HMAC}(k, \{f'_1\}, \text{"BIT"})$. If one of the above conditions is not satisfied, the search result is invalid or incomplete; otherwise, it is valid and complete.

2) Operator "`AND`*":* In this case, we will give two examples, which leads to different processes.

a) No file matching the search query: Assume that the search query is $w_{i_1} \bigwedge w_{i_2}$, where w_{i_1} = "SEC" and w_{i_2} = "BIT", the corresponding search token is $w'_{i_1} \bigwedge w'_{i_2}$ = "JIE" $\bigwedge$ "SWH".

For w'_{i_1} = "JIE", CS can obtain $(\ell_1^*, h_{cp_{\ell_1^*}}, \mathcal{S}'_{c_{\ell_1^*}}) = (0, 267, \{'\mathrm{M}', '\mathrm{S}'\})$ as that in the previous example. In this case, CS simply returns $(0, 267, \{'\mathrm{M}', '\mathrm{S}'\})$ as the search result. Note that it is no need to search "SWH" in the encrypted radix tree due to the property of operator "AND".

To check the validity and completeness of $(0, 267, \{'\mathrm{M}', '\mathrm{S}'\})$, CU checks whether $'\mathrm{J}' \notin \{'\mathrm{M}', '\mathrm{S}'\}$, and $267 = \mathtt{HMAC}(k, \$, \{'\mathrm{M}', '\mathrm{S}'\})$. If one of the above conditions is not satisfied, the search result is invalid or incomplete; otherwise, it is valid and complete.

b) Files matching the search query: Assume that the search query is $w_{i_1} \bigwedge w_{i_2}$, where w_{i_1} = "AH" and w_{i_2} = "BITE", the corresponding search token is $w'_{i_1} \bigwedge w'_{i_2}$ = "MY" $\bigwedge$ "SWHI".

Clearly, CS can find "MY" and "SWHI" in the encrypted radix tree, and it obtains $(B_{w_{i_1}}, h_{B_{w_{i_1}}}) = (153, 389)$ and $(B_{w_{i_2}}, h_{B_{w_{i_2}}}) = (114, 358)$. After that, it computes $B_i = B_{w_{i_1}} \& B_{w_{i_2}} = 114 = (10010000)_2$. From $(10010000)_2$, CS know that f_5 and f_8 satisfy the search query $w_{i_1} \bigwedge w_{i_2}$. According to $\mathrm{Des}_{5,8}$, CS gets $\{f'_5, f'_8\}$. Furthermore, it also extracts $\mathcal{H}_{\mathcal{F}'_{w_{i_1}}, B_i} = \{457, 438\}$ and $\mathcal{H}_{\mathcal{F}'_{w_{i_2}}, B_i} = \{486, 432\}$ from the encrypted radix tree. In the end, CS returns $(\{f'_5, f'_8\}, (153, 389, \{457, 438\}), (114, 358, \{486, 432\}))$.

To verify the validity and completeness of $(\{f'_5, f'_8\}, (153, 389, \{457, 438\}), (114, 358, \{486, 432\}))$, CU first checks

$$389 = \mathtt{HMAC}(k, 153, \text{"AH"}), \quad 358 = \mathtt{HMAC}(k, 114, \text{"BITE"}).$$

If one of the above equalities does not hold, the search result is invalid; otherwise, CU computes $153\&114 = (10010000)_2$, and gets two bits with value 1 in $(10010000)_2$. If the size of $\{f'_5, f'_8\}$ is not two, then the search result is incomplete; otherwise, CU checks

$$457 = \mathtt{HMAC}(k, f'_5, \text{"AH"}), \quad 438 = \mathtt{HMAC}(k, f'_8, \text{"AH"}),$$
$$486 = \mathtt{HMAC}(k, f'_5, \text{"BITE"}), \quad 432 = \mathtt{HMAC}(k, f'_8, \text{"BITE"}).$$

If one of the above equalities does not hold, the search result is invalid or incomplete; otherwise, it is valid and complete.

3) Operator "$*$": As that in operator "AND" case, we have two query examples in this case.

a) No file matching the search query: Assume that the search query is w_i, where w_i = "SEC $*$ ", the corresponding search token is w'_i = "JIE $*$ ". It is easy to see that CS will return $(\ell^*, h_{cp_{\ell^*}}, \mathcal{S}'_{c_{\ell^*}}) = (0, 267, \{'\mathrm{M}', '\mathrm{S}'\})$ as the search result.

To check the validity and completeness of the search result, CU simply checks whether $'\mathrm{J}' \notin \{'\mathrm{M}', '\mathrm{S}'\}$ and $267 = \mathtt{HMAC}(k, \$, \{'\mathrm{M}', '\mathrm{S}'\})$. If one of the above conditions is not satisfied, the search result is invalid or incomplete; otherwise, it is valid and complete.

b) Files matching the search query: Assume that the search query is w_i, where w_i = "BIT $*$ ", the corresponding search token is w'_i = "SWH$*$". According to this search token, CS finds "SWH" in the encrypted radix tree, and gets a subtree $\tilde{T}$. In the subtree $\tilde{T}$, the root node is the node $\boxed{\text{'H'} \mid 238}$ in level three of the encrypted radix tree, the inner node is the node $\boxed{\text{'I'} \mid 222}$ in level four of the encrypted radix tree, and the leaf nodes are nodes $\boxed{\# \mid 399}$ and $\boxed{\# \mid 327}$ as shown in Fig. 6. Furthermore, CS fetches $\{f'_5, f'_8\}$ and $\{f'_1\}$ according to $\mathrm{Des}_{5,8}$ and Des_1 in the corresponding leaf nodes, respectively. After the above steps, CS sends $\{f'_5, f'_8\}$, $\{f'_1\}$ and $\tilde{T}$ to CU as the search result.

To check the validity and completeness of $\{f'_5, f'_8\}$, $\{f'_1\}$ and $\tilde{T}$, CU checks

$$\begin{aligned} 238 &= \texttt{HMAC}(k, \$\text{“SWH”}, \{\text{'I'}, \#\}), \\ 222 &= \texttt{HMAC}(k, \$\text{“SWHI”}, \{\#\}), \\ 399 &= \texttt{HMAC}(k, \{f'_5, f'_8\}, \text{“BITE”}), \\ 327 &= \texttt{HMAC}(k, \{f'_1\}, \text{“BIT”}). \end{aligned}$$

If one of the above equalities does not hold, the search result is invalid or incomplete; otherwise, it is valid and complete.

4) Operator “?”: In this case, we omit the query example where the symbol subchain before operator “?” cannot be found in the encrypted radix tree, since it is almost the same as in the case of operator “$*$”.

Assume that the search query is w_i = “BI?E”, and the corresponding search token is w'_i = “SW?I”. According to the search token, CS can get “SW" in the encrypted radix tree, and find ‘I’ in the subtree with root node $\boxed{\text{‘H’} \mid 238}$ in level two but not the subtree with root node $\boxed{\text{‘N’} \mid 262}$ in level two. As a result, the search result is $(\text{'N'}, 3, 262, \{\#\})$, $(\text{'H'}, \{f'_5, f'_8\}, 399)$ and 259.

To check the validity and completeness of the search result, CU checks

$$\begin{aligned} 259 &= \texttt{HMAC}(k, \$\text{“SW”}, \{\text{'N'}, \text{'H'}\}), \\ 262 &= \texttt{HMAC}(k, \$\text{“SWN”}, \{\#\}), \\ 399 &= \texttt{HMAC}(k, \{f'_5, f'_8\}, \text{“BITE”}). \end{aligned}$$

Note that CU can decrypt $'\text{H}'$ to get $'\text{T}'$ according to the relationship in Fig. 3. If one of the above equalities does not hold, the search result is invalid or incomplete; otherwise, it is valid and complete.

Notes. From the above toy example, we can see that every component in our design has its own functionality. The radix tree data structure is used to realize the fuzzy keyword search. The values of h_{cp_i}'s are used to guarantee the integrity of the radix tree, and each of them can authenticate the position of the corresponding encrypted symbol in the encrypted radix tree.

From the bitmap arrays B_i's, we can know how many files satisfying the search query, which is used to realize operator “AND”. The integrity of bitmap arrays is guaranteed by the values of h_{B_i}'s. However, the bitmap arrays cannot realize operator “AND” alone, we still need h_{j,w_i}'s to make sure which file satisfies the search query.

For operators “OR”, “$*$” and “?”, the search result should contain all the files satisfying one keyword associated to the search query. In this case, $h_{\mathcal{F}'_{w_i}}$'s are used to ensure all the files are included in the search result.

V. Security Analysis

In this section, we will analyze the security of our proposed verifiable SE scheme, especially the confidentiality of files and the verifiability of the search results.

A. Confidentiality of Files

Due to the simplicity, it is easy to obtain the following two theorems from the description of our proposal.

Theorem 1. *If the underlying FPE scheme is a secure pseudorandom permutation, then the algorithm for encrypting the radix tree is a secure pseudorandom permutation.*

Theorem 2. *If the underlying symmetric key scheme is cryptographically secure, then the algorithm for encrypting the files is cryptographically secure.*

However, as we mentioned in Section III-B, the encrypted radix tree may reveal some secret information if the symbol frequency in the encrypted radix tree reflects that in some keyword set. Fortunately, this frequency analysis attack can be mitigated by slightly modifying the proposed protocol as follows. In particular, the process $s'_i \Leftarrow \texttt{FPE}(k_0, s_i)$ is replaced by $s'_i \Leftarrow \texttt{FPE}(k_{i,w}, s_i)$, where $k_{i,w} = \texttt{KDF}(k_0, \mathcal{A}_{i,w})$, KDF is a cryptographically secure key derivation function, and $\mathcal{A}_{i,w}$ is the prefix of the current keyword w before symbol s_i. According to Theorem 3, we know that the symbol frequency is garbled.

Theorem 3. *If the underlying FPE and KDF are a pseudorandom permutation and a pseudorandom function, respectively; then the modified algorithm for encrypting radix tree cannot yield the same encrypted symbol for the same symbol with different prefixes.*

Proof: If the two prefixes are different, i.e., $\mathcal{A}_w \neq \mathcal{A}_{w'}$, then we have that $k_w (= \texttt{KDF}(k_0, \mathcal{A}_w)) \neq k_{w'} (= \texttt{KDF}(k_0, \mathcal{A}_{w'}))$ according to that the underlying KDF is a pseudorandom function. Furthermore, we have that $s'_i (\Leftarrow \texttt{FPE}(k_w, s_i)) \neq s''_i (\Leftarrow \texttt{FPE}(k_{w'}, s_i))$ with $k_w \neq k_{w'}$ according to that the underlying FPE is a pseudorandom permutation. ■

Furthermore, according to Theorem 4, we have that the correctness of our proposal won't be affected by the above mitigation.

Theorem 4. *Two different keywords will always yield two different encrypted keywords. That is, if $w_0 \neq w_1$, then $w'_0 \neq w'_1$.*

Proof: Without loss of generality, we assume that $w_0 = s_{0,1}s_{0,2} \cdots s_{0,\ell_0}$ and $w_1 = s_{1,1}s_{1,2} \cdots s_{1,\ell_1}$, and the i-th position is the first different position between w_0 and w_1, i.e., $s_{0,1}s_{0,2} \cdots s_{0,i-1} = s_{1,1}s_{1,2} \cdots s_{1,i-1}$ and $s_{0,i} \neq s_{1,i}$.

Firstly, since $s_{0,1}s_{0,2} \cdots s_{0,i-1} = s_{1,1}s_{1,2} \cdots s_{1,i-1}$, we can easily obtain that $k_{i,w_0} = k_{i,w_1}$ from the two equalities $k_{i,w_0} = \texttt{KDF}(k_0, s_{0,1}s_{0,2} \cdots s_{0,i-1})$ and $k_{i,w_1} = \texttt{KDF}(k_0, s_{1,1}s_{1,2} \cdots s_{1,i-1})$. Secondly, since FPE is a secure permutation, we can easily obtain that $s'_{0,i} \neq s'_{1,i}$ from $s'_{0,i} \Leftarrow \texttt{FPE}(k_{i,w_0}, s_{0,i})$ and $s'_{1,i} \Leftarrow \texttt{FPE}(k_{i,w_1}, s_{1,i})$. Hence, we have that $s'_{0,1}s'_{0,2} \cdots s'_{0,\ell_0} \neq s'_{1,1}s'_{1,2} \cdots s'_{1,\ell_1}$, i.e., $w'_0 \neq w'_1$. ■

Note that, it is also possible for an adversary to corrupt the cloud server CS and use the information of either access pattern or search pattern to expose the content of files. The access patten refers to the information on those (encrypted) files contain the queried (encrypted) keywords, and the search patten is the information as to whether two search tokens are associated to the same search query. According to the analysis in [32] and [33], our proposal suffers from the access pattern attack and the search attack. Luckily, two generic solutions for these two attacks have also been proposed in [32] and [33], respectively. These two solutions are suitable for any searchable solutions while at the expense of a few false positives and some dummy queries, respectively. We can directly apply them in our proposal. Since the main contribution of this paper is the verifiability of the search result, we omit this part in this paper. We refer the interested readers to [32], [33] for more details.

B. Verifiability of the Search Results

Theorem 5. *If the underlying HMAC is unforgeable under chosen message attacks, then no probabilistic polynomial time adversary can break the verifiability of our proposal with a non-negligible probability.*

Proof: Assume there exists an adversary that can break the verifiability of our proposal; then we can build the following algorithm interacting with the adversary to break the unforgeability of the underlying HMAC. That is, our algorithm aims to generate a valid MAC, while the corresponding message has never been queried.

Query: We can use the oracle for generating MAC of the underlying HMAC to respond the MAC queries issued by the adversary.

Output: When the adversary successfully breaks the verifiability of our proposal, one of the following conditions should be satisfied.

- The tuple $(h_{cp_i}, \mathcal{A}_{p_i}, s'_i, \mathcal{S}'_{c_i})$ has never been queried by the adversary, but it satisfies the condition $h_{cp_i} = \mathtt{HMAC}(k, \mathcal{A}_{p_i}||s'_i, \mathcal{S}'_{c_i})$.
- The tuple $(h_{\mathcal{F}'_{w_i}}, \mathcal{F}'_{w_i}, w_i)$ has never been queried by the adversary, but it satisfies the condition $h_{\mathcal{F}'_{w_i}} = \mathtt{HMAC}(k, \mathcal{F}'_{w_i}, w_i)$.
- The tuple (h_{j,w_i}, f'_j, w_i) has never been generated by the adversary, but it satisfies the condition $h_{j,w_i} = \mathtt{HMAC}(k, f'_j, w_i)$.
- The tuple $(h_{B_{w_i}}, B_{w_i}, w_i)$ has never been generated by the adversary, but it satisfies the condition $h_{B_{w_i}} = \mathtt{HMAC}(k, B_{w_i}, w_i)$.

It is easy to see that any of the above four cases is a valid output to break the unforgeability of the underlying HMAC. ■

VI. Performance Evaluation

In this section, we will evaluate the performance of our proposed verifiable SE scheme in terms of communication overhead and computational costs.

Communication Overhead The communication overhead of queries from `CU` to `CS` is the same as that in the normal cloud storage systems due to the property of format preserving encryption, i.e., no ciphertext expansion. While the communication overhead of responses from `CS` to `CU` varies from one operator to another, and it is relatively small if we do not take the part of encrypted files into consideration for evaluation, as the communication costs for returning the encrypted files are the same in all encrypted cloud storage systems. In particular, it is proportional to the number of keywords for the case of operator "`OR`". For each keyword, it costs $(\ell^*, h_{cp}, \mathcal{S}'_{c_{\ell^*}})$ or $h_{\mathcal{F}'}$ that is less than 500bits under the situation that $|\ell^*| < 34$bits, $|h_{cp}| = |h_{\mathcal{F}'}| = 256$bits, and $|\mathcal{S}'_{c_{\ell^*}}| \leq 26 * 8 = 208$bits. Regarding the case of operator "`AND`", it is proportional to the number of keywords and the number of satisfied files due to $\{B, h_B, \mathcal{H}\}$. It is easy to see that it costs one hash value for each keyword/satisfied file pair, and it also costs n_f bits and one hash value for each keyword. Finally, for the cases of operators "$*$" and "?", it is proportional to the number of branches of the radix tree satisfying the query, which is similar with the case of operator "`OR`".

Computational Cost For the computational cost, we would like to give the analysis based on experimental results without counting the time of decryption on the encrypted files. Concretely, we first implemented our scheme (with SHA256 as the underlying hash function) using Java without multithread or any other parallel techniques, and then ran the cloud-side programs (the search process) on a server and the user-side programs (the search result verification) on both a laptop and a smart phone. The server is deployed with Intel Xeon E5-2603v4 1.7 GHz and 32 GB memory running Ubuntu 16.04 LTS, the laptop is equipped with Intel Core i5-4210U 1.7 GHz and 8 GB memory running Windows 10 Professional, and the smart phone is implemented with Kirin 960 and 6 GB memory running Android 7.0.

The underlying dataset in our experiment is composed by 15,000 random documents larger than 100 bytes from Wikivoyage [34], and the underlying radix tree is constructed based on 8,000 corresponding keywords parsed from these 15,000 documents. The resultant keyword length (the number of symbols in the keyword) distribution can be found in Fig. 7. Every point in Figs. 8-13 is derived from average result of 100 runs with the same query. For the conveniences of description, we denote `qs` and `kl` respectively as the query size (the number of keywords in the query) and the keyword length in the following experiments.

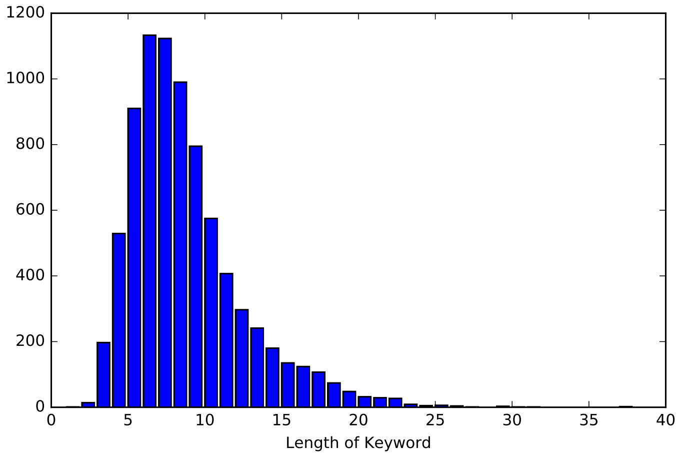


Fig. 7. Keyword length distribution.

A. Operator "`OR`"

In the experiment for operator "`OR`", we have ten different query sizes ranging from one to ten.

1) Search Efficiency: Due to the property of the radix tree and operator "`OR`", `CS` would do `kl` comparisons at most for one keyword with length `kl`, and `qs` · `kl` comparisons at most for one query containing `qs` keywords with length `kl`. The keyword length of each keyword varies due to the random keyword generation, which will cause quite different experimental results even for the queries with the same query size. To remove this influence, we not only ran the same kind of query for 100 times with randomly generated keywords, but also make use of the box-and-whisker diagram to show the experimental result distribution. In Fig. 8, we plot the search efficiency for queries with operator "`OR`" varying with keyword count from 1 to 10. In the figure, the box-and-whisker diagram used in this paper has the following properties: i) The bottom and top of the box, and the band inside the box denote the first, third, and second quartile of the data, respectively; ii) the lower (resp. upper) whisker denotes the lowest (resp. highest) datum within 1.5 interquartile range (IQR) of the lower (resp. upper) quartile; ii) outliers that are not included between the whiskers are plotted as individual points. From the figure, we can see that the computational cost increases as the query size increases as expected.

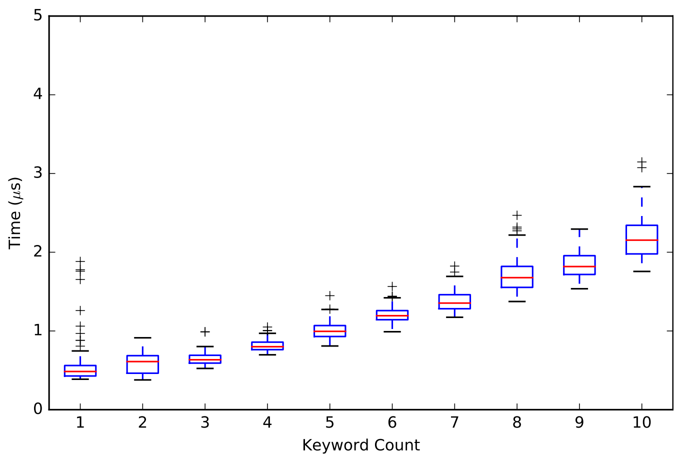
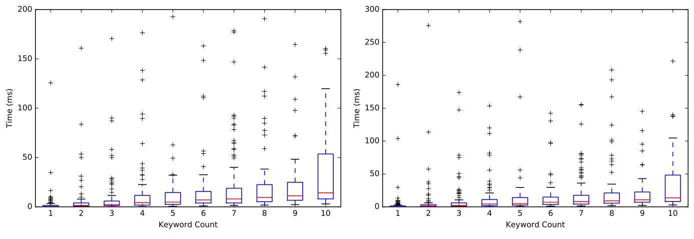


Fig. 8. Search efficiency for queries with operator "`OR`"

2) Verification Efficiency: For queries only containing operator "`OR`", the returned result would contain several $(\mathcal{F}'_{w_{i_j}}, h_{\mathcal{F}'_{w_{i_j}}})$'s and $(\ell^*_j, h_{cp_{\ell^*_j}}, \mathcal{S}'_{c_{\ell^*_j}})$'s, and one of them corresponds to one keyword. On the other hand, no matter $(\mathcal{F}'_{w_{i_j}}, h_{\mathcal{F}'_{w_{i_j}}})$ or $(\ell^*_j, h_{cp_{\ell^*_j}}, \mathcal{S}'_{c_{\ell^*_j}})$, only one `HMAC` computation is required to do the verification. Hence, for queries with size `qs` in the case of operator "`OR`", we need to perform `qs` `HMAC` computations to verify the returned search result. Corresponding to the search experiments, we ran every kind of query for 100 times and make use of the box-and-whisker diagram to show the experimental result. From Figs. 9(a) and 9(b), we can see that the computational cost increases as the query size increases as expected.

B. Operator "`AND`"

Similar with the experiment of operator "`OR`", we have nine different query sizes ranging from two to ten in this experiment. Note that the keywords in this experiment are randomly



(a) Verifying in a laptop (b) Verifying in a smart phone

Fig. 9. Verification efficiency for queries with operator "`OR`"

chosen from the 8,000 keywords parsed from our dataset. With this keyword choice, there would be no short-circuiting of operator "`AND`" happens during the search process, and the computational cost of the search process would be maximized for each kind of query. If the maximum computational cost is still low, then we can claim that our proposal is efficient for search with operator "`AND`".

1) Search Efficiency: Since all the keywords are chosen from the 8,000 keywords of our dataset, `CS` has to do the search for every keyword in the query. Similar with that in the case of operator "`OR`", `CS` would do `qs`·`kl` comparisons at most. Again, to obtain a convictive experimental result, we ran the same kind of query 100 times with randomly generated keywords and make use of the box-and-whisker diagram to show the experimental result. From Fig. 10, we can see that the computational cost increases as the query size increases as expected.

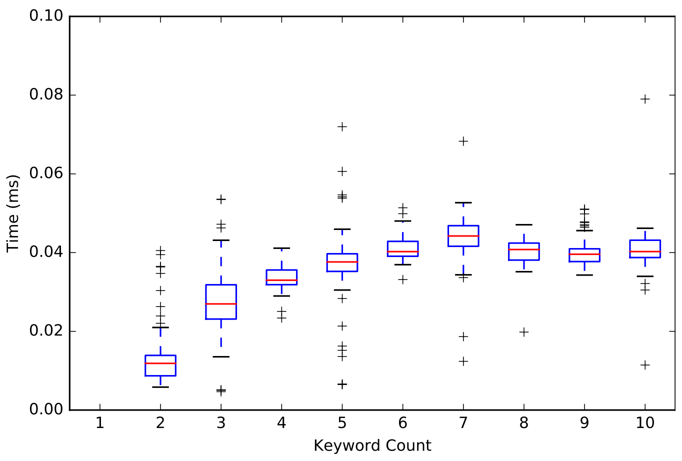
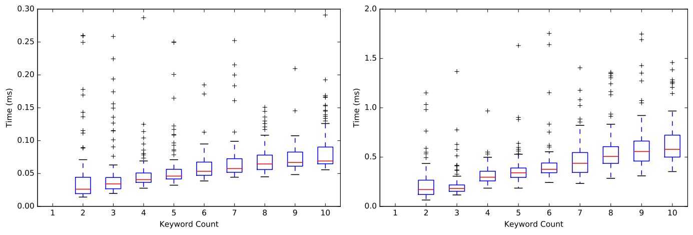


Fig. 10. Search efficiency for queries with operator "`AND`"

2) Verification Efficiency: To verify the completeness and correctness of the returned search result, `CU` needs to finish the following three tasks. i) Verify the integrity of bitmap corresponding to each keyword, ii) compute the "Bit AND" result of these bitmaps and count the number of bit 1 in the obtained result, ii) compute `HMAC`'s for each satisfied file with every keyword in the query. Hence, the computational cost in this experiment is related to not only the query size, but also the number of satisfied files that varies due to the underlying keywords and dataset. Corresponding to the search experiment, we ran the same kind of query 100 times and make use of the box-and-whisker diagram to show the experimental result as previous experiments. From Figs. 11(a) and 11(b), we can see that the computational cost increases as the query size increases as expected.



(a) Verifying in a laptop (b) Verifying in a smart phone

Fig. 11. Verification efficiency for queries with operator "AND"

At first glance, the experimental results in Figs. 11(a) and 11(b) are wrong. Theoretically, the number of HMAC computation of operator "AND" is a lot more than that of operator "OR". However, the experimental result shows that the verification of operator "AND" is more efficient than that of operator "OR". This conflict is mainly because that the input of HMAC in the case of operator "OR" is a lot larger than that of operator "AND".

C. Operator "$*$"

Similar with the previous experiments, we have ten different kinds of queries with ten different prefix lengths ranging from one to ten, where the prefix is the symbols before "$*$". Furthermore, we require that all the encrypted prefixes can be found in the radix tree, which makes the computational cost of search process in this experiment reach maximum.

1) Search Efficiency: The main search step for operator "$*$" is to search the encrypted prefix in the radix tree. Hence, the computational cost is quite similar to the case of operator "OR" with one keyword. Here, we still ran the same kind of query for 100 times with randomly generated keywords and use the box-and-whisker diagram to show the experimental result as previous experiments. Fig. 12 indicates that the computational cost remains almost the same as the prefix length increases, which is against the theoretical analysis. The reason of this conflict is mainly because the computational cost of an encrypted symbol comparison in the radix tree is quite low and ten comparison times are the maximum number in our experiment.

2) Verification Efficiency: Since CS always returns a subtree of the underlying radix tree in this experiment, the computational cost for the verification is mainly to the verification of the returned subtree. Again, we ran the same kind of query for 100 times with randomly generated keywords, while we use 3-dimension figures to show the relationship among the prefix length, the size of the returned tree and the verification time as in Figs. 13(a) and 13(b). The cross points in these figures are the projection of the circle points on the coordinate plane. The projection on the left plane shows that the computational cost increases as the returned subtree becomes larger, while the projection on the bottom plane shows that shorter prefix, larger subtree.

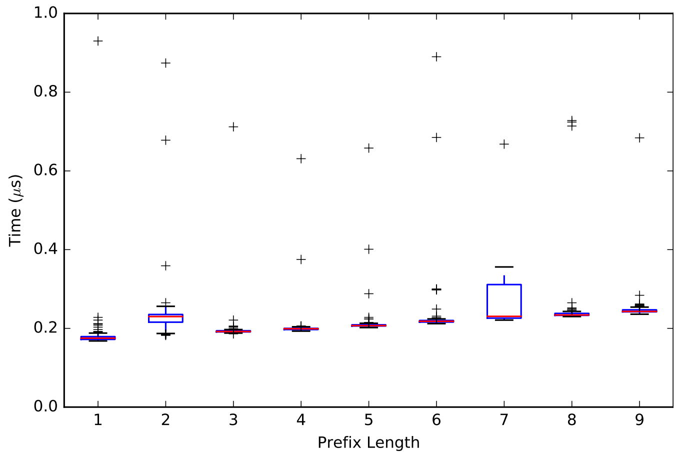
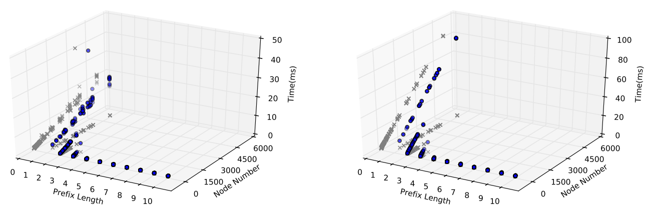


Fig. 12. Search efficiency for queries with operator "$*$"



(a) Verifying in a laptop (b) Verifying in a smart phone

Fig. 13. Verification efficiency for queries with operator "$*$"

D. Operator "?"

In this experiment, we have 100 kinds of queries with different lengths of prefix and suffix, where the prefix and suffix of a query are the symbols before "?" and that after "?", respectively. The lengths of prefix and suffix range from zero to ten, and the length of the query (excluding "?") is always ten. Furthermore, we required that the encrypted prefix can be always found in the radix tree, which makes the computational cost of search on the prefix reach maximum.

1) Search Efficiency: There are two main parts for the search of operator "?". One is for the search on the encrypted prefix in the radix tree, the other is for the search on the encrypted suffix in the radix tree starting from the second generation descendants of the node corresponding to the encrypted prefix. For the first part, CS would do $\texttt{len}_p$ comparisons when the prefix length is $\texttt{len}_p$. While regarding the second part, CS would do $\texttt{len}_s * n$ comparisons at most, if the suffix length is $\texttt{len}_s$ and there are n child nodes of the node corresponding to the encrypted prefix. As previous experiments, we ran the same kind of query for 100 times with randomly generated keywords, while we use a 3-dimension figure to show the relationship among prefix length, suffix length and the verification time. Fig. 14 shows that longer prefix consumes less time, which coordinates with the situation that the shorter the prefix, the more characters that can be connected behind it in general.

2) Verification Efficiency: CS can always find satisfied files according to the properties of queries in this experiment. Hence, the computational cost of verification is mainly related to the number of child nodes of the node corresponding to

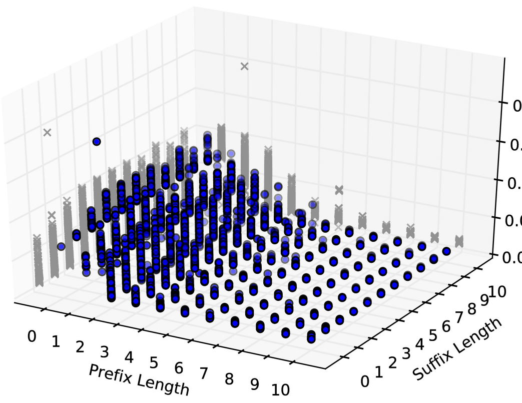


Fig. 14. Search efficiency for queries with operator “?”

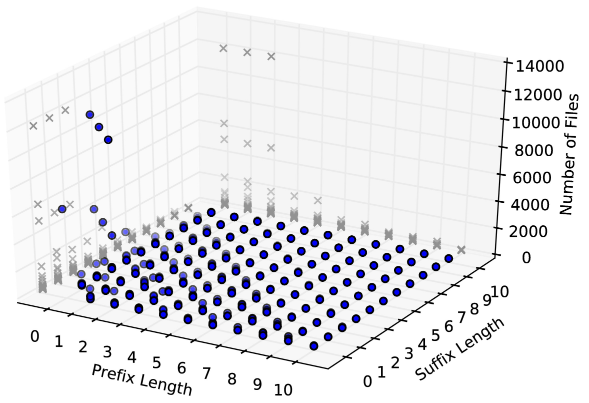
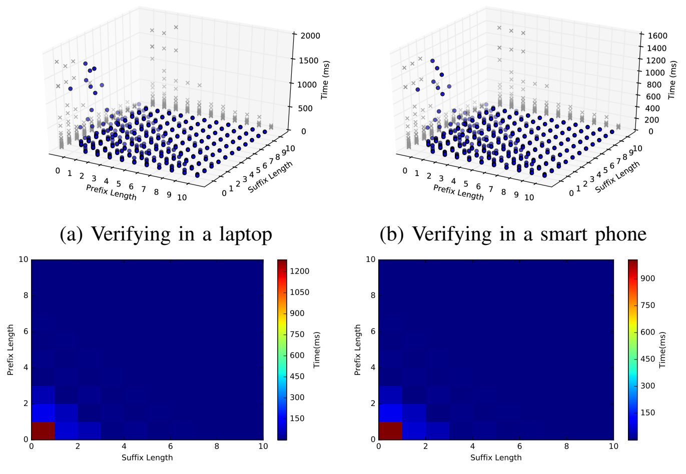


Fig. 16. Number of returned files for queries with operator “?”

the encrypted prefix and the number of files contained in the response. Figs. 15(a) and 15(b) demonstrate the relationship among prefix length, suffix length, and verification time, while Fig. 16 shows the relationship among prefix length, suffix length, and the number of returned files. With these figures, we can deduce that the verification time is mainly related to the number of returned files as the theoretical analysis. Moreover, Figs. 16(c) and 16(d) show the average verification time among different prefix and suffix length. As illustrated in these two figures, the time for the client to verify most of the search responses is less than 20ms.



(c) Average verification time for a laptop (d) Average verification time for a smart phone

Fig. 15. Verification efficiency for queries with operator “?”

From Figs. 8-15, we can easily see that all the experimental processes including the one running in mobile devices can be completed in less than 33 ms (according to the average results). Hence, we can conduct that our proposal is efficient and suitable for the use in resource-constrained devices.

VII. Related Work

The problem of verifiability of search results in searchable encryption was firstly investigated by Chai and Gong [9]. By using the radix tree, bitmap index, hash function and symmetric key encryption, they proposed the first verifiable searchable symmetric key encryption (VSSKE) that only support one keyword search. Since then, many different VSSKE schemes with different properties have been proposed. Wang et al. [10] proposed a VSSKE scheme to support fuzzy keyword search. Zhu et al. [12] extended VSSKE supporting fuzzy keyword search to the dynamic case by using locality sensitive hashing and Bloom filter. Later, Zhu et al. [13] proposed another VSSKE scheme supporting dynamic fuzzy keyword search with a rigorous security proof. Based on m-best tree and term similarity tree, Fu et al. [11] proposed a new VSSKE scheme supporting sematic search. Chase and Shen [35] proposed a new VSSKE scheme supporting substring search by applying similar techniques used in our proposal. However, the client in their scheme cannot verify whether all valid results are returned. Bost and Fouque [14] proposed a new VSSKE scheme supporting dynamic keyword search and forward security. Recently, Ogata and Kurosawa [15] proposed a more efficient VSSKE scheme with forward security by using Cuckoo hashing [36], pseudo-random function and symmetric key encryption. Very recently, Wang et al. [37] proposed a new VSSKE scheme supporting conjunctive keyword search and large-scale database.

Sun et al. [16] and Zheng et al. [17] independently extended the concept of verifiability in searchable symmetric key encryption into searchable public key encryption. Two verifiable searchable public key encryption (VSPKE) schemes were proposed in [16], [17] respectively. One [16] supports conjunctive keyword search, the other [17] supports one keyword search. Later on, Sun et al. [18] proposed a new VSPKE scheme to support conjunctive keyword search and dynamic data. Miao et al. [20] proposed a similar VSPKE scheme without secure channels. Sun et al. [19] and Miao et al. [21] further extended the setting of VSPKE by allowing more data owners (contributors). Recently, Jiang et al. [22] proposed a new VSPKE scheme with public verifiability.

Nevertheless, all of the above mentioned verifiable searchable encryption (VSE) scheme cannot support query operator “OR”, “AND”, “$*$” and “?”, simultaneously. To the best of our knowledge, only a few of the existing VSE schemes [38], [39] hold this property. However, all of these VSE schemes require many time-consuming operations, such as bilinear maps. From a practical point of view, it is desired to design an efficient VSE scheme supporting various queries that contain one or many of operators "OR", "AND", "$*$" and "?", especially for the use in mobile devices.

Although the basic techniques applied in our proposal also appear in other schemes, we uniquely integrate them together to get a new VSSKE scheme that can support operators "OR", "AND", "$*$" and "?" without any time-consuming operations for the first time. Furthermore, confidentiality of the data and verifiability of the search result in our proposal can also be well achieved.

VIII. Conclusion

In this paper, by using the bitmap index, radix tree, keyed-hash message authentication code, format preserving encryption and symmetric key encryption, we have proposed a new searchable encryption scheme that has the following properties: 1) Supporting various queries containing one or many of query operators "OR", "AND", "$*$" and "?"; 2) verifiability of search results; 3) no time-consuming operation required during the whole process. Furthermore, we have also implemented a prototype of the proposed verifiable SE scheme and tested its effectiveness and efficiency on the real dataset from Wikivoyage. The experimental results indicate that our proposal is efficient and suitable for the use in resource-constrained devices.

As a future research effort, we plan to refine our verifiable SE scheme to support dynamic data, queries with "NOT" operator, and the setting where cloud users are granted with different access rights. Furthermore, the similarity search over encrypted data has been studied extensively recently [40]–[42], we also plan to add the result verifiability into the related works.

Acknowledgement

The source code and dataset in this paper can be downloaded from http://github.com/guanyg/vsse.

References

[1] Research and Markets, "Cloud storage market - forecasts from 2017 to 2022," Online, May 2017, https://www.researchandmarkets.com/research/lf8wbx/cloud_storage.

[2] Wikipedia, "icloud leaks of celebrity photos," Online, Aug. 2014, https://en.wikipedia.org/wiki/ICloud_leaks_of_celebrity_photos.

[3] S. Khandelwal, "Download: 68 million hacked dropbox accounts are just a click away!" Online, Oct. 2016, https://thehackernews.com/2016/10/dropbox-password-hack.html.

[4] M. K. McGee, "Blood test results exposed in cloud repository," Online, Oct. 2017, https://www.databreachtoday.com/blood-test-results-exposed-in-cloud-repository-a-10382.

[5] C. Zuo, S. Sun, J. K. Liu, J. Shao, and J. Pieprzyk, "Dynamic searchable symmetric encryption schemes supporting range queries with forward (and backward) security," in *ESORICS*, 2018, pp. 228–246.

[6] H. Li, D. Liu, Y. Dai, T. H. Luan, and S. Yu, "Personalized search over encrypted data with efficient and secure updates in mobile clouds," *IEEE Trans. Emerging Topics Comput.*, vol. 6, no. 1, pp. 97–109, 2018.

[7] H. Blodget, "Amazon's cloud crash disaster permanently destroyed many customers' data," Online, Apr. 2011, http://www.businessinsider.com/amazon-lost-data-2011-4.

[8] R. Jennings, "Oops: Google 'loses' your cloud data (sky falling; film at 11)," Computerworld, Aug. 2015, https://www.computerworld.com/article/2973600/cloud-computing/google-cloud-loses-data-belgium-itbwcw.html.

[9] Q. Chai and G. Gong, "Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers," in *IEEE ICC*, 2012, pp. 917–922.

[10] J. Wang, X. Chen, H. Ma, Q. Tang, J. Li, and H. Zhu, "A verifiable fuzzy keyword search scheme over encrypted data," *Journal of Internet Services and Information Security*, vol. 2, no. 1/2, pp. 49–58, 2012.

[11] Z. Fu, J. Shu, X. Sun, and N. Linge, "Smart cloud search services: Verifiable keyword-based semantic search over encrypted cloud data," *IEEE Transactions on Consumer Electronics*, vol. 60, no. 4, pp. 762–770, 2014.

[12] X. Zhu, Q. Liu, and G. Wang, "Verifiable dynamic fuzzy search over encrypted data in cloud computing," in *ICA3PP*, 2015, pp. 655–666.

[13] ——, "A novel verifiable and dynamic fuzzy keyword search scheme over encrypted data in cloud computing," in *IEEE Trustcom/BigDataSE/ISPA*, 2016, pp. 845–851.

[14] R. Bost and P.-A. Fouque, "Verifiable dynamic symmetric searchable encryption optimality and forward security," 2016, https://eprint.iacr.org/2016/062.pdf.

[15] W. Ogata and K. Kurosawa, "Efficient no-dictionary verifiable searchable symmetric encryption," in *Financial Cryptography and Data Security*, 2017, pp. 498–516.

[16] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li, "Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 11, pp. 3025–3035, 2014.

[17] Q. Zheng, S. Xu, and G. Ateniese, "VABKS: Verifiable Attribute-based Keyword Search over Outsourced Encrypted Data," in *IEEE INFOCOM*, 2014, pp. 522–530.

[18] W. Sun, X. Liu, W. Lou, Y. T. Hou, and H. Li, "Catch you if you lie to me: Efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data," in *IEEE INFOCOM*, 2015, pp. 2110–2118.

[19] W. Sun, S. Yu, W. Lou, Y. T. Hou, and H. Li, "Protecting your right: Verifiable attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud," *IEEE Transactions on Parallel and Districuted Systems*, vol. 27, no. 4, pp. 1187–1198, 2016.

[20] Y. Miao, J. Ma, F. Wei, Z. Liu, X. A. Wang, and C. Lu, "Vcse: Verifiable conjunctive keywords search over encrypted data without secure-channel," *Peer to Peer Network Application*, vol. 10, pp. 995–1007, 2017.

[21] Y. Miao, J. Ma, X. Liu, Q. Jiang, J. Zhang, L. Shen, and Z. Liu, "Vcksm: Verifiable conjunctive keyword search over mobile e-health cloud in shared multi-owner settings," *Pervasive and Mobile Computing*, vol. 40, pp. 205–219, 2017.

[22] S. Jiang, X. Zhu, L. Guo, and J. Liu, "Publicly verifiable boolean query over outsourced encrypted data," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp. 1–1, 2017.

[23] J. Steven, "Internet stats & facts for 2019," Online, Dec. 2018, https://hostingfacts.com/internet-facts-stats/.

[24] C.-Y. Chan and Y. E. Ioannidis, "An efficient bitmap encoding scheme for selection queries," in *SIGMOD*, 1999, pp. 215–226.

[25] D. Knuth, *The Art of Computer Programming, Volume 3: (2nd ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., 1998.

[26] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *CRYPTO*, 1996, pp. 1–15.

[27] M. Bellare, T. Ristenpart, P. Rogaway, and T. Stegers, "Format-preserving encryption," in *SAC*, 2009, pp. 295–312.

[28] J. Li, Z. Liu, X. Chen, F. Xhafa, X. Tan, and D. S. Wong, "L-encdb: A lightweight framework for privacy-preserving data queries in cloud computing," *Knowledge-Based Systems*, vol. 79, pp. 18–26, 2015.

[29] J. Shao, R. Lu, and X. Lin, "Fine: A fine-grained privacy-preserving location-based service framework for mobile devices," in *IEEE INFOCOM*, 2014, pp. 244–252.

[30] C. Zuo, J. Shao, J. K. Liu, G. Wei, and Y. Ling, "Fine-grained two-factor protection mechanism for data sharing in cloud storage," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 1, pp. 186–196, 2018.

[31] J. Black and P. Rogaway, "Ciphers with arbitrary finite domains," in *CT-RSA*, 2002, pp. 114–130.

[32] M. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: Ramification, attack and mitigation," in *NDSS*, 2012.

[33] C. Liu, L. Zhu, M. Wang, and Y. Tan, "Search pattern leakage in searchable encryption: Attacks and new construction," *Information Sciences*, vol. 265, pp. 176–188, 2014.

[34] wikivoyage, https://www.wikivoyage.org/, accessed Jan. 2018.

[35] M. Chase and E. Shen, "Substring-searchable symmetric encryption," in *PoPETs*, vol. 2015, no. 2, 2015, pp. 263–281.

[36] R. Pagh and F. F. Rodler, "Cuckoo hashing," in *ESA*, 2001, pp. 121–133.

[37] J. Wang, X. Chen, S. Sun, J. K. Liu, M. H. Au, and Z. Zhan, "Towards efficient verifiable conjunctive keyword search for large encrypted database," in *ESORICS*, 2018, pp. 83–100.

[38] R. Cheng, J. Yan, C. Guan, F. Zhang, and K. Ren, "Verifiable searchable symmetric encryption from indistinguishability obfuscation," in *AsiaCCS*, 2015, pp. 621–626.

[39] J. Alderman, C. Janson, K. M. Martin, and S. L. Renwick, "Extended functionality in verifiable searchable encryption," in *BalkanCryptSec*, 2015, pp. 187–205.

[40] X. Yuan, H. Cui, X. Wang, and C. Wang, "Enabling privacy-assured similarity retrieval over millions of encrypted records," in *ESORICS*, 2015, pp. 40–60.

[41] Z. Xia, Y. Zhu, X. Sun, Z. Qin, and K. Ren, "Towards privacy-preserving content-based image retrieval in cloud computing," *IEEE Trans. Cloud Computing*, vol. 6, no. 1, pp. 276–286, 2018.

[42] H. Cui, X. Yuan, Y. Zheng, and C. Wang, "Towards encrypted in-network storage services with secure near-duplicate detection," *IEEE Transactions on Services Computing*, 2018.

Jun Shao received the Ph.D. degree from the Department of Computer Science and Engineering at Shanghai Jiao Tong University, Shanghai, China in 2008. He was a postdoc in the School of Information Sciences and Technology at Pennsylvania State University, USA from 2008 to 2010. He is currently a professor of the School of Computer Science and Information Engineering at Zhejiang Gongshang University, Hangzhou, China. His research interests include network security and applied cryptography.

Rongxing Lu (S'09-M'11-SM'15) has been an assistant professor at the Faculty of Computer Science (FCS), University of New Brunswick (UNB), Canada, since August 2016. Before that, he worked as an assistant professor at the School of Electrical and Electronic Engineering, Nanyang Technological University (NTU), Singapore from April 2013 to August 2016. Rongxing Lu worked as a Postdoctoral Fellow at the University of Waterloo from May 2012 to April 2013. He was awarded the most prestigious "Governor General's Gold Medal", when he received his PhD degree from the Department of Electrical & Computer Engineering, University of Waterloo, Canada, in 2012; and won the 8th IEEE Communications Society (ComSoc) Asia Pacific (AP) Outstanding Young Researcher Award, in 2013. He is presently a senior member of IEEE Communications Society. His research interests include applied cryptography, privacy enhancing technologies, and IoT-Big Data security and privacy. He has published extensively in his areas of expertise (with citation 14,900+ and H-index 60 from Google Scholar as of April 2019), and was the recipient of 8 best (student) paper awards from some reputable journals and conferences. Currently, Dr. Lu serves as the Vice-Chair (Publication) of IEEE ComSoc CIS-TC (Communications and Information Security Technical Committee). Dr. Lu is the Winner of 2016-17 Excellence in Teaching Award, FCS, UNB.

Yunguo Guan is a master student of the School of Computer Science and Information Engineering at Zhejiang Gongshang University. His research interests include applied cryptography and game theory.

Guiyi Wei is a professor of the School of Computer Science and Information Engineering at Zhejiang Gongshang University. He obtained his Ph.D. in Dec 2006 from Zhejiang University, where he was advised by Cheung Kong chair professor Yao Zheng. His research interests include wireless networks, mobile computing, cloud computing, social networks and network security.