ISSN: 2664-9640

A Safe and Verifiable Fuzzy Keyword Search Method for Searchable Encryption

Bangying Wu[1], Yuanyuan Xu[2] and Hengru Zhang[2,*]

[1]Network and Information Center, Southwest Petroleum University, Chengdu, China.

[2] School of Computer Science, Southwest Petroleum University, Chengdu, China.

* Corresponding Author

Abstract

Searchable encryption uses keywords to retrieve encrypted data on the server. Data privacy disclosure and search results cannot be verified are two major difficulties of searchable encryption. In this paper, we propose an safe and verifiable fuzzy keyword search (SVFKS) method to solve the two difficulties. To ensure data privacy and security, we utilize several secret keys and the one-way trapdoor function to build a symbol tree. And we employ the collision resistance property of hash function to against attacks from an adversary who wants to steal data privacy. To verify the search results, we record every route from the root node to the current node in symbol tree. And we construct a verifiable strategy by comparing real search route with these records. The performing analysis results indicate our method is more safe and reliable than the-state-of-arts approaches.

Keywords

Searchable encryption, fuzzy keyword, privacy, verifiable, symbol tree.

1. Introduction

The emergence of cloud computing provides considerable opportunities for academia, IT industry and global economy. Increasing amounts of sensitive information is being centralized into the cloud, such as emails, personal health records, government documents and etc. By storing these data into the cloud, the data owners can be relieved from the burden of data storage and maintenance so as to enjoy the on-demand high quality data storage service. However, the cloud server cannot be fully trusted, they may be curious about the outsourced data. To protect data privacy, sensitive data has to be encrypted by data owners before outsourcing, which brings the problem about how to search the data we need without downloading and decrypting all the data. Therefore, exploring privacy-assured and effective search service over encrypted cloud data is equivalent essential and meaningful. Exact Keyword Search and Fuzzy Keyword Search are two widely accepted mainstream technologies in searchable encryption area. As textual information is ubiquitous in data management systems, traditional searchable encryption schemes solve the problem of searching over encrypted data through exact keyword matching.

In Exact Keywords Search, Asymmetric Searchable Encryption and Symmetric Searchable Encryption are two popular methods to process data encryption. In Asymmetric Searchable Encryption, Boneh et al. [2] construct a mechanism called Public-Key Encryption with Keyword Search (PEKS) based public key cryptosystem. It shows private key owner can provide a key to a server that enables the server to only check whether the key in the encrypted file which was encrypted by the related public key holders. In Symmetric Searchable Encryption, Song et al. [17] take the form of probabilistic searching and provide a controlled searching technique which support hidden queries that guarantee a server even know nothing about the search

ISSN: 2664-9640

keywords. Furthermore, Reza Curtmola et al. [5] make some safety and efficient improvements based on Symmetric Searchable Encryption and take a multi-user SSE scenario into consideration.

The query keyword is a single and exact word in Exact Keyword Search, thus misspelled keyword in a query may give rise to retrieving nothing. Consequently, many applications have an increasing need to support Fuzzy Keyword Search over encrypted data set. All these applications amplify the crucial importance of enabling the Fuzzy Keyword Search in outsourced cloud data management systems with high system usability and better user search experience. Fuzzy Keyword Search can greatly enhance system usability by returning the matching files when the query inputs just exactly match the predefined keywords or the closest possible matching files based on keyword similarity semantics. Li et al. [12] exemplified the Fuzzy Keyword Search scheme over encrypted data, their scheme has a wildcard-based fuzzy set construction. In the scheme, wildcards are put into consideration to form a fuzzy set whose size is drastically large. Liu et al. [13] improved the scheme by constructing dictionary-based fuzzy set construction to get smaller index size. Zhang et al. [6] through inner product compute for similarity-based ranking to achieve multi-keyword search in privacy-preserving conditions. Other Fuzzy Keyword Search schemes over encrypted data optimized the data structure index with various tree structures such as R-tree, multi-way tree, and symbol tree to improve the searching efficiency.

However, most schemes above do not regard the cloud server as a semi-honest-but-curios one and they overlook the case that the cloud server may return only part of the searching results in order to save its own computation resource. To address the problem of dishonest server, Chai et al. [3] put forward a verifiable SSE (VSSE) scheme, which ensures users can verify the correctness and completeness of the search results but only be applied in the exact keywords search. Qing et al. [19] presented Verifiable Attribute-based Keyword Search which uses Attribute-based Encryption as an access control policy.

Despite the fact that these schemes take the verification into consideration can enable users who satisfy the access control policy to verify the search results, they can only solve exact keyword matching and their solution algorithms are complex as well. In our work, we propose a scheme called SVFKS and our main contributions can be summarized as four-fold:

Our scheme employees the idea of String edit distance and supports dictionary based Fuzzy Keyword Search.

Searching results can be verified in our scheme on encrypted data.

We use one-way trapdoor function and the collision resistance of hash function to protect data privacy.

With symbol index tree, our scheme reduces the verifying computation cost to $O(1)$. The search time complexity of our scheme is almost equal to the scheme with multi-way tree.

2. Related Work

A variety of searchable encryption schemes have been designed to perform secure and effective search both without decrypting the whole data files. These searchable encryption schemes can be classified into two large categories: Exact Keyword Search and Fuzzy Keyword Search. To improve the efficiency of search, Multiple Keywords Search is investigate in the searchable encryption research. And considering the accuracy of search results, a Verifiable Keyword Search based on Exact Keyword Search and Fuzzy Keyword Search is derived.

Exact Keyword Search. The first construction of searchable encryption [17] support the view that each word in the document is encrypted separately under a special two-layered encryption construction. This construction needs to scan the entire file collection to obtain the complete

ISSN: 2664-9640

query results. Goh [7] outlines Bloom filters to construct the indexes for the data files. The keyword is inserted into the corresponding position of the Bloom filter through the hash function. For achieving more efficient search, Chang et al. [4] and Curtmola et al. [5] both propose a similar ‘‘index'' approache, where a single encrypted hash table index is built for the entire file collection.

Multiple Keywords Search. However, the above methods are all based on single keyword retrieval form. Single keyword search may return redundant file collection due to lack of accuracy. According to the dimension of data retrieval keywords, searchable encryption can be classified into single keyword search and multiple keywords search. A bucketizaiton based approach [8] that an attribute domain is partitioned into sets of buckets to support range queries is proposed. Their strategy is to process the decryption and other query processing at client site to save the computation power of server and protect against data thefts from the untrusted server. Considering the privacy threats arising from the creation of bucketization-based indices, a privacy-efficiency trade-offs for the bucketization scheme [9] based on controlled diffusion is designed. A K-anonymity techniques [16] which utilize generalization and suppression is proposed to solve the privacy disclosure of relationship between attributes in bucketizaiton scheme.

Fuzzy Keyword Search. Nevertheless, Exact Keyword Search can realize the retrieval on ciphertext, there is no tolerance of minor typos or format inconsistencies. The frequent user searching behavior makes the research of Fuzzy Keyword Search particularly necessary. A wildcard-based technique [12] which utilize edit distance to quantify keywords similarity is explored for the construction of fuzzy keyword sets. Unfortunately, their technique requires large data storage space. With considering scalable for large data sources and efficient similarity search, locality sensitive hashing technique is introduced in Fuzzy Keyword Search [11].

Verifiable Keyword Search. The authenticity of the results returned by the server is not considered in all the above methods. The problem is first address in [3] that query results returned by the cloud server cannot be fully trusted in the presence of a semi-honest-but-curious server. And they offer a verifiable symmetric searchable encryption method. Moreover, a two-phase search method [10] which provides versatility by allowing arbitrary and dynamic matching of phrases is presented against the same type of cloud server scenarios.

In this paper, we incorporate with modifications the valuable ideas and techniques in the above methods, and make some improvements to achieve our method SVFKS. Compared with other searchable encryption scheme, our solution is more efficient. Besides, the correctness and completeness of the search results can be verified to achieve more stringent information security in our scheme.

Organization: This paper is composed of six sections: Section 1 is the introduction. Related work is presented in Section 2. Section 3 exhibits the system model of our method. The design of safe and verifiable fuzzy keyword search method is described in detail in Section 4. In Section 5, we evaluate the security of our method and analyse verifiable ability. Finally, we make the conclusion of this paper in Section 6.

3. System Model

Assuming there exists a cloud server who is ‘‘honest-but-curious". Specifically, the server will follow our predefined protocol, but still try to find out as much secret information as possible based on its possessions. Moreover, we notice that the cloud server may be selfish in the aspect of saving computation ability or bandwidth as well, and data security for this type of server is significantly beyond our conventional ‘‘honest-but-curious'' server model. We consider this stronger adversary as a ‘‘semi-honest-but-curious" one. In other words, this server is un-

ISSN: 2664-9640

trusted and it may execute only a fraction of the search and/or return part of the searching results honestly.

The data owner represents either an individual or an enterprise customer who has a collection of *n* data files to be stored in the cloud server. We define the collection of data files as:

$$F=\left\{\left\langle fid, file_i\right\rangle \mid 1\le i\le n\right\}. \tag{1}$$

where $file_i$ is the *i*-th file identifier of *F*, *fid* is the file identifier of $file_i$ and every identifier is unique. To keep sensitive data confidential from unauthorized entities, the file collection *F* stored on server should be encrypted with a secret key. We define *C* as a collection of encrypted files of *F*,

$$C=E_{k_1}\left(F\right). \tag{2}$$

where k_1 is a secret key generated by the data owner.

For our scheme, we suppose the system model in the ‘‘semi-honest-but-curious'' server scenario. The architecture of our system model in a cloud computing scenario with the ‘‘semi-honest-but-curious'' cloud servers is presented in Figure1. The system model consists of three different entities: data owner, cloud servers and data users. In our scheme, we assume the ‘‘semi-honest-but-curious'' cloud servers satisfy the following properties:

The cloud server is a storage provider who follows our proposed protocol and does not modify/destroy the stored documents;

The cloud server tries to derive sensitive information from the stored documents, query pattern and search results;

The cloud server may execute only a fraction of search operations honestly and return part of the outcomes to user.

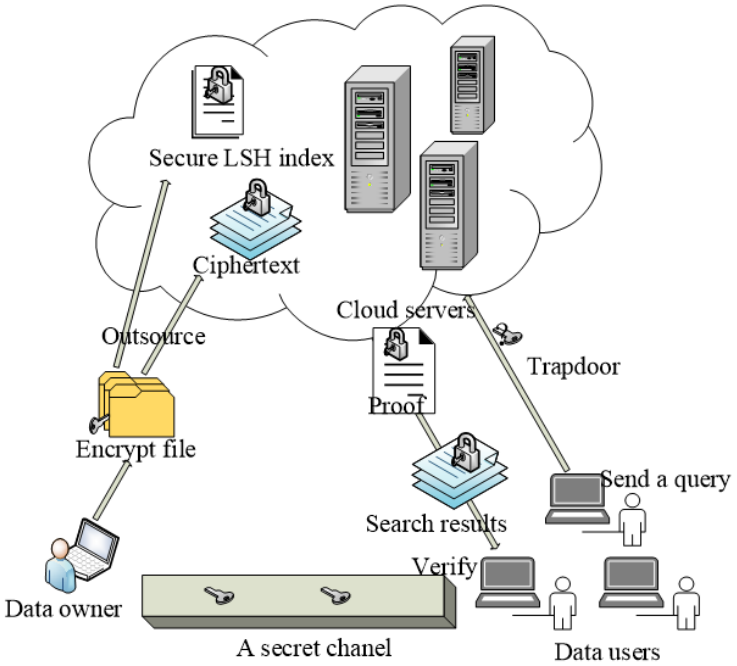

Figure 1: The architecture of our system model in a cloud computing scenario with the “semi-honest-but-curious” cloud servers

To achieve effective utilization of cloud data, such service architecture should be equipped with Fuzzy Keyword Search capability on encrypted cloud data. In addition, this architecture needs to also needs to ensure that the data stored on the server is secure. Moreover, supporting the function of results verification is necessary for the dishonest properties of the cloud servers.

ISSN: 2664-9640

When the data users get the results from the cloud server, they could verify the correctness and completeness of these returned results.

4. Safe and Verifiable Fuzzy Keyword Search

4.1. Notations

The notations are used in our method as shown in Table1.

Table 1: Notations

Symbol	Description
n	the number of files outsourced on cloud server.
$D=(e_1,e_2,\cdots,e_m)$	a predefined dictionary which contains a set of distinct legal English words.
p	the number of distinct keywords which can be extracted from the file collection F.
$W=(w_1,w_2,\cdots,w_p)$	a dictionary which contains a set of distinct keywords in D extracted from the file collection F.
$f_{k_2}(w)$	a one-way function with secret key k_2 for a keyword w.
T_w	the trapdoor of a search request contained a keyword w.
$g_{k_1}(\ )$	a secure encryption function based on semantic with secret key k_1.
$E_{k_1}(\ ),\ D_{k_1}(\ )$	a secure encryption/decryption function based on semantic with secret key k_1.
$h_{k_3}(\ )-\{0,1\}^*\rightarrow\{0,1\}^l$	a keyed hash function with secret key k_3, defined as $key \times \{0,1\}^* \rightarrow [0,1]^l$, l is length of the output.

4.1.1. String edit distance

Definition4.1. String operations [12] can be defined as follows:

Insertion: Insert one character in the string.

Deletion: Delete one character in the string.

Substitution: Replace a character in the string.

String Edit Distance is the minimum number of insertions, deletions, and substitutions required to transform one string into another. In this paper, we denote $ed(str_1, str_2)$ as the string edit distance between str_1 and str_2.

4.1.2. Dictionary

Dictionary D limits the scope of fuzzy keywords. In other words, all fuzzy keywords should belong to the dictionary. For example, we can make the dictionary contains all legal English words. But in most circumstance, stop words such as “a”, “the”, “about” will be excluded from the dictionary for the reason that these words are not usually used as keywords.

4.1.3. Alternative fuzzy keywords

Definition4.2. Given a keyword w and a dictionary D, the fundamental dictionary fuzzy keywords set of w can be defined as:

$$F_{w,d}^{D}=\{w_1,\cdots,w_i,\cdots w_m\}\ . \tag{3}$$

where all $w_i \in D$ and $0 \le ed(w,w_i) \le d$. The parameter d is a flexible, we can adjust the scope of fuzzy keyword D by adjusting the size of d.

In Eq.3, we obtain all approximate keywords of the the w with distance less than d. The length of the fuzzy keywords set of w is m, we denoted as $\left\|F_{w,d}^{D}\right\|$. That is,

ISSN: 2664-9640

$$\left\| F_{w,d}^{D} \right\| = m\left(w_i \in D, 1 \le i \le m \right) . \tag{4}$$

And we can build a secondary dictionary fuzzy keywords set W of keyword w which is defined as:

$$W = F_{w_1,d}^{D} \bigcup \cdots \bigcup F_{w_i,d}^{D} \bigcup \cdots \bigcup F_{w_m,d}^{D} . \tag{5}$$

Suppose the fuzzy keywords in the secondary dictionary fuzzy keywords set W denoted as w_W, while in the fundamental dictionary fuzzy keywords set denoted as w_F. It is not hard to figure out that

$$ed\left(w_W, w \right) \le ed\left(w_F, w \right) . \tag{6}$$

Therefore, W can be used as an alternative set in case no search results are found when use $F_{w,d}^{D}$.

4.2. Algorithms

Five polynomial time algorithms: ***KeyGen***, ***Buildindex***, ***Trapdoor***, ***Search***, and ***Verify*** constitute our SVFKS.

1. ***KeyGen*** $\left(\lambda \right) \to \left(k_1, k_2, k_3 \right)$:

KeyGen is a randomized key generation algorithm. It inputs security parameter λ and outputs three secret keys: k_1, k_2 and k_3 respectively. Secret key k_1 is used for files encryption and verification, secret key k_2 is used for trapdoor and index, secret key k_3 is used for the keyed hash function.

2. ***Buildindex*** $\left(W, k_2, k_3 \right) \to \left(G_{index} \right)$:

Buildindex is run by the data owner to create the index. It inputs keyword set W and the index secret key k_2 then outputs the symbol tree G_{index}. Considering the privacy disclosure from the index, the content of the symbol tree G_{index} will be hashed with the secret key k_3. The outline of symbol tree G_{index} is presented in Figure2.

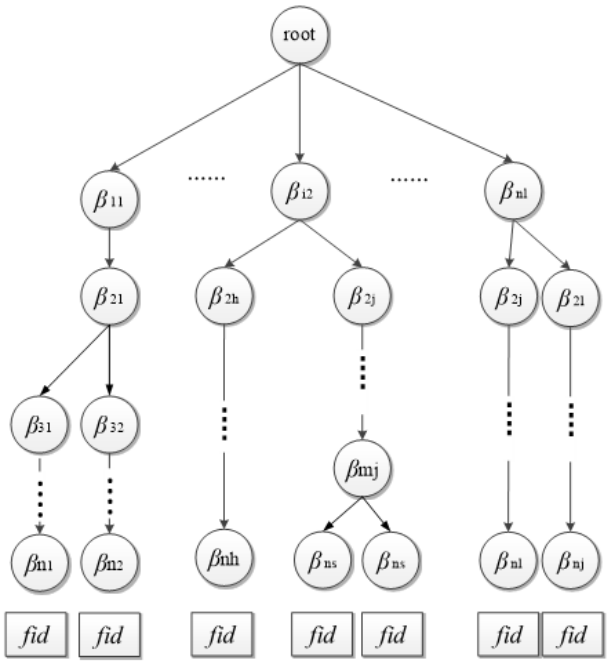

Figure 2: The architecture of our system model in a cloud computing scenario with the "semi-honest-but-curious" cloud servers

ISSN: 2664-9640

3. ***Trapdoor*** $\left(k_2, F_{w,d}^D\right) \to \{T_{w'}\}$:

This algorithm takes the trapdoor key k_2 and the fuzzy keyword set $F_{w,d}^D$ as inputs, then outputs the trapdoor set $T_{w'}$. For each keyword w in $F_{w,d}^D$, we employ a one-way function to calculate the T_w which is defined as:

$$T_{w'} = T_{w_1} \cup \cdots \cup T_{w_i} \cup \cdots \cup T_{w_m} \tag{7}$$

where $w' \in F_{w,d}^D$. $T_{w'}$ can be generated in two scenario: when data user sends a query to cloud server or at the data owner builds index tree time. In this paper, we suppose the keyword w comes from a query of the data users.

4. ***Search***$\left(G_{index}, \{T_{w'}\}\right) \to \{fid_{set}, proof\}$:

The cloud server execute ***Search*** over the encrypted files C, in order to find out the data which may contain keyword w'. Algorithm ***Search*** has two input parameters, they are index tree G_{index} and the trapdoor set $T_{w'}$ which is generated by data users query, respectively. If the search process goes successfully, then the cloud servers will output the set of file identifiers fid_{set} and *proof*. Both of them will be utilized in next stage for verification purpose. If the search process goes wrong, the cloud servers will output *proof* only. The unique *proof* will still be used for validation.

5. ***Verify*** $\left(k_1, k_3, fid_{set}, proof\right) \to \{True, False\}$:

The final steps in our method is validate the returned results. Algorithm ***Verify*** is executed by the data users to test whether the cloud server is honest or not. ***Verify*** takes secret key k_1, k_3 and the fid_{set} & *proof* from previous process as its input parameters. It is worth mentioning that if the cloud servers find nothing about the fuzzy keyword of query or the cloud servers are dishonest leading to retrieve nothing, the input parameter fid_{set} will become unnecessary. If the verification process is successful then ***Verify*** will output *True* (which means the returned results from cloud servers can be trusted) or else output *False* (which means the could servers are dishonest in the search results).

4.3. Symbol Tree Structure

We build a symbol tree to improve the query efficiency of our searching scheme. To build a symbol tree, the key idea behind this construction is that all trapdoors sharing a common prefix have a common node. The root is associated with an empty set and the symbols in a trapdoor can be recovered in a search from the root to the leaf that ends a trapdoor. All the dictionary fuzzy keywords in the tree can be found by the depth-first search.

Assume $\Delta = (\alpha_i)$ is a predefined symbol set, each symbol $\alpha_i \in \Delta$ is denoted by a t-bit binary vector and the maximum number of different symbols in Δ is $\|\Delta\| = 2^t$. The Δ has no practical significance and can be regarded as a codebook.

In Eq.7, the data users compute $T_{w'} = f\left(k_2, w_i\right)$ by taking the index generation secret key k_2 and the keyword $w_i \in F_{w,d}^D$ of related query as input. Then they divide each trapdoor value into $\left\lceil \left\| f_{k_2}(w) \right\| / t \right\rceil$ parts (from high-order position). Note that, each part is a t-bit binary number except the last part. The length of the last part is in the range of [1, t - 1). For consistency, a unified and special character is required to complete the last part to a t-bit string.

ISSN: 2664-9640

The data owner takes each part as a symbol and utilizes these symbols with some ingenious strategies (which we will introduce later) to construct an index tree G_{index} with covering all the fuzzy keywords of w_i in $F_{w,d}^{D}$.

The structure of G_{index} is shown in Figure2. Each node of the G_{index} is associated with a *t*-th bits binary symbol, where *t* is a flexible parameter and can be adjusted for a better performance purpose.

In the G_{index}, we set β_{ij} as the presentation of the node and define it as the *i*-th node from left to right of depth *j*. And we attach the corresponding $g_{k_1}(fid)$ of the searching route to each leaf node.

4.4. The Implement of SVFKS

4.4.1. Execution flow overview

The lifecycle of SVFKS can be summarized into three stages in our searchable encryption strategy.

The first execution flow, the data owner employs ***KeyGen*** to generate secret keys first. Then he/she employs the one of the secret keys and ***Buildindex*** to create encrypted file and the corresponding hashed index, respectively. Finally, these files which the data owner built will be outsourced to the cloud servers together.

In the second stage, data users employ ***Trapdoor*** to produce the trapdoor set of the related query with a given keyword. Then they send the trapdoor set of the keyword to the cloud servers for next ***Search***. The cloud servers execute ***Search*** and then return the search results & proof file back to the data users. For data safety, the returned search results & proof are still encrypted.

The last stage is the process of getting the plaintext results of the query. After the data users obtaining the search results returned by the cloud servers, they will perform ***Verify*** to validate the correctness and completeness of search results. The returned results from cloud servers can be classified into two categories: find the queried data or not. In the first scenario where the queried data is found, the ***Verify*** will help the data users to examine whether the outcome can be trusted or not. In the second scenario where the queried data is not found, the data users can execute the ***Verify*** to check whether there is really no outcomes of the query can be retrieved on the cloud servers.

4.4.2. Build symbol-tree index structure

The structure of our proposed symbol tree is shown in Figure2. Each node in the symbol index tree G_{index} is a tuple with three elements (r_0, r_1, r_2).

r_0 stores the *t* bit binary symbol of β_{ij}. As a consequence, r_1 has 2^t different results which represent the set of children of the current node.

r_1 is a bit stream which length is 2^t. From the root to current node in G_{index}, if current node's parents whose r_0 is the *i*-th symbol in Δ, then the *i* -th bit of current node r_1 is set to 1, while other bit positions in r_1 are all set to 0. Note that, the r_2 is only used in the ***Verify*** when the search route cannot reach the leaf node.

r_2 saves two results of the keyed hash value. One is $h_{k_3}(mem)$ where *mem* means symbol sequence from the root to the current node. And the other one is r_3. In our design, r_3 is defined differently according to the following two scenarios.

If current node is an intermediate node, $r_3 = r_1$.

If current node is a leaf node, $r_3 = h_{k_3}(fid_w)$.

ISSN: 2664-9640

w is a keyword associated with the current leaf node and fid_w is a set of file identifiers where these files contain keyword w.

The data owner outsources G_{index} with encrypted files to the cloud server. Meanwhile, he/she attaches $g_{k_1}(r_4)$ to every leaf node for further verification.

4.4.3. Verification

Verification is executed in every inquired route. $proof_i$=($path_i$, R_i) can be utilized by user to examine the correctness and completeness of search results.

When the search reaches the leaf node for the i-th route.

First, the data users obtain fid_{w_i} by decrypting $g_{k_1}(fid_{w_i})$ with k_1, that is

$$fid_{w_i} = D_{k_1}\left(g_{k_1}(fid_{w_i})\right) \tag{8}$$

Then they use the result fid_{w_i} in Eq.10 to compute $h_{k_3}(fid_w)$ in order to ascertain whether it is equal to the fid_w in R_i. If matches successfully then verification process will go to the second stage; otherwise the process indicates the cloud servers cannot be trusted.

Second, the data users estimate whether *path* in *proof* is in accordance with *mem* in R_i. They use k_3 to compute $h_{k_3}(path)$ first, then they compare the $h_{k_3}(path)$ with *mem* in R_i. If there is no difference, it means the cloud servers can be trusted; otherwise the cloud severs are dishonest.

When the search failure.

First, the data users only investigate the *proof* set to evaluate whether *path* in *proof* is matched precisely with *mem* in R_i. If the match is failure, the cloud sever will be un-trusted; otherwise go to the second step.

Second, the data users obtain Δ by decrypting $g_{k_1}(\Delta)$ with secret key k_1, the operation is noted as

$$\Delta = D_{k_1}\left(g_{k_1}(\Delta)\right) \tag{9}$$

Then the symbol sequence (the route from root to current node) of current node can be determined by r_1 and Δ. Finally, the data users check whether the symbol sequence is equal to the *path* in *proof*. If it is equal, the cloud servers can be trusted. Otherwise it indicates the cloud servers are dishonest.

5. Security and Performance Analysis

5.1. Security Analysis

Data Privacy. The documents stored on the cloud server are encrypted separately with k_1, and their confidentiality is essentially ensured by the underlying cipher. By using a cryptographic strong cipher, it fully guarantees these encrypted documents leaks zero information (except their respective lengths). Besides, the privacy-preserving query can be understood as a collection of prefix signatures. Its confidentiality and one-way properties are guaranteed by the underlying hash function. Note that, more focus should be placed on the confidentiality of the index. To be specific, each node in G_{index} has a triplet (r_0, r_1, r_2). r_1 and r_2 are hashed value. Therefore, even an attacker obtains (r_1, r_2), he cannot derive related plain-text information directly and easily.

Nonetheless, a dishonest cloud server may learn statistic information according to (r_1, r_2) by taking advantage of the mutual information among nodes in G_{index}. Considering these potential threats, we make the following essential observation in an outsourcing environment.

ISSN: 2664-9640

Proof. *In terms of the search privacy, the SVFKS method is adequate secure.*

Suppose SVFKS cannot achieve the index privacy which guarantees the encrypted information is indistinguishable under the chosen keyword attack (INDCPA). It means there may exist an algorithm A which can obtain the underlying information of a keyword from the index. In the worst situation, an adversary knows the entire SVFKS method includes G_{index}. Then he or she chooses some keywords randomly and initiates corresponding queries. However, even the adversary generates these trapdoor successfully, he cannot compute a new index $G_{index'}$ which is matched G_{index} perfectly by using these keywords are randomly chosen. Moreover, due to the adversary cannot obtain a probabilistic estimate of the true value with high degree of confidence. He or she cannot deprive the underlying keywords set W from the G_{index} directly either. Furthermore, ***Buildindex*** is an one-way function, which means utilizing W and k_2 to compute G_{index} can be easy while the reverse direction to calculate W will be hard. In conclusion, there is no algorithm can deprive the underlying keywords from G_{index} without the secret key k_2.

5.2. Verifiability

Verifiable Searchability. To prove the verifiability, we need to prove that the adversary cannot forge a valid *proof* which be denoted as δ. To tamper the search results, there are three ways for the adversary to forge the δ:

Generate a δ with different parameter *path* and/or *mem*;

Randomly generate a δ to replace the original real one;

Take the *proof* of another node as the δ of current node and return it back to the user.

In the first two ways, considering the collision resistance properties of hash function, each node in G_{index} has a unique *proof*. Therefore, without the secret key k_1, the adversary can cheat successfully in a negligible probability. He or she cannot return part of the search results or some fault one.

In the last way, there is just one unique *mem* from the root to the current node in G_{index} according to the symbol tree structure. In other words, the *mem* of any node can be regards as a signature of the current node. The δ which should be the *proof* of other node will be rejected in the ***Verify*** process.

Based on the above analysis, the adversary cannot construct a valid proof without the secret key k_1. We can conclude our proposed SVFKS is secure based on the assumption of collision resistance of hash function.

We compare our scheme with previous VSSE schemes as shown in Table2. Our scheme supports Ciphertext No-distinguish, Trapdoor No-distinguish and Fuzzy Keyword Search.

Table 2: Comparison between SVFKS and other schemes

Scheme	Ciphertext No-distinguish	Trapdoor No-distinguish	Secure Channel	Fuzzy Keyword Search
[2]	√	×	√	×
[1]	√	×	×	×
[14]	√	√	√	×
[18]	√	√	×	×
[12]	√	×	√	√
ours	√	√	√	√

ISSN: 2664-9640

6. Conclusion

In this paper, we propose a SVFKS method supports Fuzzy Keyword Search in the cloud computing environment with the ‘‘semi-honest but curious'' cloud servers. Our method provides data privacy security by introducing one-way trapdoor function and hash function with several secret keys. Our method also offers the verifiable searchability that can verify the integrity and correctness of returned results from cloud servers. Moreover, our method is efficiency with the structure of our proposed index based symbol tree. The rigorous security and verifiable searchability analysis confirm that our method is feasible to realize the data privacy-usability trade-off in a data outsourcing scenario.

References

[1] Baek, J.; Safavi-Naini, R.; and Susilo, W. 2008. Public key encryption with keyword search revisited. In International conference on Computational Science and Its Applications, 1249–1259. Springer.

[2] Boneh, D.; Di Crescenzo, G.; Ostrovsky, R.; and Persiano, G. 2004. Public key encryption with keyword search. In International conference on the theory and applications of cryptographic techniques, 506–522.

[3] Chai, Q.; and Gong, G. 2012. Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers. In 2012 IEEE International Conference on Communications (ICC), 917–922. IEEE.

[4] Chang, Y.-C.; and Michael, M. 2005. Privacy preserving keyword searches on remote encrypted data. In International conference on applied cryptography and network security, 442–455. Springer.

[5] Curtmola,R.; Garay,J.; Kamara,S.; andOstrovsky, R. 2011. Searchable symmetric encryption: improved definitions and efficient constructions. Journal of Computer Security, 19(5): 895–934.

[6] Fu, Z.-J.; Sun, X.-M.; Xia, Z.-H.; Zhou, L.; and Shu, J.-G. 2013. Multi-keyword ranked search supporting synonym queryoverencrypteddataincloudcomputing. In2013IEEE 32nd International Performance Computing and Communications Conference (IPCCC), 1–8. IEEE.

[7] Goh, E.-J. 2003. Secure indexes. IACR Cryptol. ePrint Arch., 2003: 216.

[8] Hacigümüs¸, H.; Iyer, B.; Li, C.; and Mehrotra, S. 2002. Executing SQL over encrypted data in the database-service-provider model. In Proceedings of the 2002 ACM SIGMOD international conference on Management of data, 216–227.

[9] Hore, B.; Mehrotra, S.; and Tsudik, G. 2004. A privacy-preserving index for range queries. In Proceedings of the Thirtieth international conference on Very large data bases Volume 30, 720–731.

[10] Kissel, Z. A.; and Wang, J. 2013. Verifiable phrase search over encrypted data secure against a semi-honest-but-curious adversary. In 2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops, 126–131. IEEE.

[11] Kuzu, M.; Islam, M. S.; and Kantarcioglu, M. 2012. Efficient similarity search over encrypted data. In 2012 IEEE 28th International Conference on Data Engineering, 1156–1167. IEEE.

[12] Li, J.; Wang, Q.; Wang, C.; Cao, N.; Ren, K.; and Lou, W.-J. 2010. Fuzzy keyword search over encrypted data in cloud computing. In 2010 Proceedings IEEE INFOCOM, 1–5. IEEE.

[13] Liu, Q.; Wang, G.-J.; and Wu, J. 2009. An efficient privacy preserving keyword search scheme in cloud computing. In 2009 International Conference on Computational Science and Engineering, volume 2, 715–720. IEEE.

[14] Rhee, H. S.; Park, J. H.; Susilo, W.; and Lee, D. H. 2010. Trapdoor security in a searchable public-key encryption scheme with a designated tester. Journal of Systems and Software, 83(5): 763–771.

[15] Samarati, P.; and Sweeney, L. 1998. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression.

[16] Song, D. X.; Wagner, D.; and Perrig, A. 2000. Practical techniques for searches on encrypted data. In Proceeding 2000 IEEE symposium on security and privacy. S&P 2000, 44–55. IEEE.

ISSN: 2664-9640

[17] Zhao, Y.-J.; Chen, X.-F.; Ma, H.; Tang, Q.; and Zhu, H. 2012. A new trapdoor-indistinguishable public key encryption with keyword search. J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl., 3(1/2): 72–81.

[18] Zheng, Q.-J.; Xu, S.-H.; and Giuseppe, A. 2014. VABKS: Verifiable attribute-based keyword search over outsourced encrypted data. In IEEE INFOCOM 2014-IEEE conference on computer communications, 522–530. IEEE.