Abstract—This paper proposes an inverted-index-based, practical, encrypted search engine, including encrypted index creation, search, and maintenance methods. Recently, more and more patients are placing their medical data at online third-party services, such as Google Health. If such sensitive data are stored at a semi-trusted third-party, security guarantee is necessary for assurance of data confidentiality and access pattern privacy. However, most service providers only provide such assurance by policy means. The new method proposed in this paper can efficiently resolve both the medical data confidentiality and sharing privacy issues. Based on our encrypted inverted-index table structure, key derivation and distribution methods, it is convenient for a patient to authorize others to access his encrypted medical data without any loss of data confidentiality and access pattern privacy to an unauthorized person. Further with the help of virtual deletion technology, our method resolves the encrypted index table maintenance issue when medical data are deleted. The experimental results show that the overhead of our solution is small compared to indexing and search times of traditional search engines. The security analysis shows that our solution assures data confidentiality and access pattern privacy.

I. INTRODUCTION

Today, population movement becomes more frequent. When a person moves from one city to another, he may want to move his personal medical data to a medical provider in the new city. But currently, each hospital keeps its own patients’ medical data. The questions come up: how to manage the patients’ medical data in different hospitals, and how such data can be viewed by other doctors in different hospitals. Internet-based online archive service is such a kind of solution to provide huge amounts of storage space to the end users. Based on the online storage space, some new medical services have been proposed. In February 2008, Google announced a new online service, Google Health, to help patients manage their online medical information [1].

Because of the confidentiality of the medical data, the online medical services must offer the assurance of data confidentiality and access pattern privacy to the patients. HIPAA [2] also requires that electronic private health information be transmitted and stored in an encrypted format. However, web medical information storage service providers cannot satisfy strict privacy requirements. They can only assure patients’ privacy by policy means, not by technology means. What makes the situation even more serious is that the management of the encrypted data, such as the key management and distribution, the encrypted data sharing, etc., becomes very challenging given the continuous expansion of the data scale. One of the most difficult challenges is how to effectively share the encrypted data with an authorized person without revealing the access pattern to others.

Consider such a scenario: Patients store their medical data at a semi-trusted third-party storage service provider. Here semi-trusted means the service provider can assure customers’ data privacy by policy means, not by technology means. Such data includes ordinary medical information such as a patient’s allergy symptoms. The patients may want to share the medical data with various people, such as their ophthalmologists, dentists, or medical researchers. In the example shown in Fig.1, a patient would like to make his allergies information and eye disease records accessible to the ophthalmologists while keeping other data secret to them. Similarly, he would like to make some of his data accessible to dentists and some medical researchers, respectively, while access to other data is denied.

So we need data security and sharing methods for these requirements. Currently, any encryption methods can meet the data security requirement easily. However, the current access control methods [3] cannot meet the encrypted data sharing and

![Figure 1. Medical Application Scenario.](image-url)
access pattern privacy requirements, especially for the situation that the patient does not know which person should access which portions of the medical data when he encrypts them.

In this paper, we describe an encrypted search method to share online medical data. With authorized search, a patient can easily control searchers to access the desired data. In addition, all of the medical data will be encrypted for confidentiality purpose. In this situation, there are following challenges for us to overcome: how to enable others to search encrypted medical data; how to disseminate data decryption keys to the searchers; how to distinguish different data access rights with respect to different searchers; how to assure search pattern privacy; and how to make it an efficient solution in terms of computation and communication resources.

In order to solve these problems, we propose an inverted-index-based encrypted search engine scheme. Our solution provides the following important security dimensions.

- Confidentiality. The medical data being stored in a third-party server is not decipherable by the server.
- Privacy of search. The keyword concerned in a search as well as the search results will not be revealed to others during or after the client-server conversation.
- Fine-grained multi-level retrieval. No searcher can obtain and decipher documents beyond his privacy level.
- Ciphertext-based data-driven access control. The medical data access is controlled by the search results, while the authorization of search is controlled in accordance with not only privacy levels but also keywords.

The rest of this paper is structured as follows. Section II briefly introduces ciphertext search technologies. Section III proposes encrypted search engine solution based on the inverted index. Section IV introduces our prototype implementation, experiment analysis and security analysis. The conclusion and future work are drawn in the last section.

II. RELATED WORK

In this section, we will introduce searchable encryption technologies.

The state of the art in search engine field has solved full-plaintext-based keyword search issues very well. However, supporting full-ciphertext-based search in a secure remote storage scenario is difficult, and often tends to significantly compromise either security or performance, or both.

To address this problem, various forms of searchable encryption have been proposed [4–6]. In order to provide secure searchable encryption, most of these schemes encrypt the content of data and attach related keywords to it in encrypted form. i.e.,

\[
\left[ E_{\text{key}_E}(D), \text{SKE(key}_1, (W_1, W_2, \ldots, W_n)) \right]
\]

where the first component is an encryption of document information \( D \) and the second one is called a searchable keyword encryption (SKE) of keywords \( (W_1, W_2, \ldots, W_n) \). A key \( \text{key}_E \) can be a symmetric key or a public key, depending on the encryption algorithm in use. When a user wants to retrieve documents containing a keyword \( kw \), he should possess a trapdoor for keyword \( kw \) and send it to the semi-trusted remote server for testing the existence of the keyword within the associated encryptions. The trapdoor for \( kw \) reveals only which encryptions contain keyword \( kw \) and no other information, even the keyword itself is not disclosed to the server.

In [4], Song, Wagner, and Perrig proposed a symmetric key scheme in which the same key was used to make SKEs and trapdoors. Afterwards, several schemes were proposed to improve and extend this scheme [5, 6]. In [5], Goh proposed an efficient symmetric key scheme using Bloom filters. The scheme can determine whether a document contains a keyword in a constant time. Both of these schemes [4, 5] are symmetric key schemes, so they are not applicable to a public key system such as the email gateway introduced by Boneh et al. [6]. For the public key systems, Boneh et al. proposed the Public Key Encryption with Keyword Search (PEKS) whose ciphertexts were created with a public key.

Unfortunately, almost all of the current searchable encryption schemes are impractical because they are based on forward index technology. In forward index, when the number of documents is very large, the query cost is intolerable. Specifically, the computation complexity that the storage server spends in searching is \( O(msn) \) due to the nature of encryption, where \( m \) is the number of documents, \( n \) is the average number of distinct words in each document.

Another disadvantage of these solutions is that even though the storage server returns matching results, since the results are pointing to encrypted documents, the searcher has to contact the data owner for the decryption keys.

III. ENCRYPTED SEARCH ENGINE

In this section, we propose the main idea to solve the encrypted data search problem.

A. Inverted-index based Searchable Encryption Scheme

Because the inverted-index structure is more efficient than the forward index, especially in the search process [7], we adopt it to realize the encrypted search engine in our solution.

Usually, a search engine contains two key phases, involving index creation and index search. In this section, we will introduce our solution exactly in these two phases.

1) Encrypted Index Creation

During the encrypted index creation phrase, besides a master encryption key (MEK) owned by data owner, for each keyword \( KW_j \), more than one keyword encryption key KEKs (\( KEK_{kw_j}, MKE_{KW_j}, HKE_{KW_j} \)) is needed. The KEKs are used for documents with different privacy level, including low, middle and high level, respectively. Besides MEK, for each document \( D_j \),
a file encryption key \( K_{\text{file}_j} \) is also needed. Then, the data owner uses different keys to encrypt \((\text{flag || } \text{FileInfo } K_{\text{file}_j})\), where 

FileInfo is the \( j \)-th document’s information, which includes document path (FilePath\_), in the storage server, document’s cipher filename (CFN\_), and document’s plain filename (PFN\_) (FileInfo = FilePath\_||PFN\_||PFN\_), and flag is helpful for a searcher in decrypting. Flag is a random number chosen by data owner during the encrypted index creation phase, and can be shared by various items in tuples. The encrypted inverted index consists of a set of tuples. The form of \( i \)-th keyword item set \((\text{KIS}_i)\) is \(<\text{H}(\text{MEK}, \text{KW})\_i: \text{LKey}_{\text{KW}}_i, \text{flag || FileInfo } K_{\text{file}_j} >\), \( \ldots <\text{E}_i(\text{HKey}_{\text{KW}}_i, \text{flag || FileInfo } K_{\text{file}_j} >, \ldots <\text{E}_n(\text{HKey}_{\text{KW}}_n, \text{flag || FileInfo } K_{\text{file}_j} >\rangle\), where \( H(X, Y) \) is a hash function with key \( X \). \( E_i(X, Y) \) is an encryption function using key \( X \). Furthermore, an existing inverted index structure such as B-tree and Hash can be adopted as a basis of the encrypted index, and the encrypted information is registered to such an existing index structure. Fig. 2 illustrates the index build process.

With our structure, we can easily realize the key management for huge documents. All of the matched documents’ decrypted keys are hidden in the encrypted index items. How can a searcher retrieve these keys? It depends on the right keyword authorization in the encrypted index search process.

2) **Encrypted Index Search**

Firstly, the data owner authorizes a searcher to query keyword \( \text{KW} \) by a token \( T_{\text{kw}} = <\text{A} = \text{H}(\text{MEK}, \text{KW}), \text{B} = \text{!Key}_{\text{kw}} \_i >\), where \( ? \) means L or M or H. Secondly, the searcher sends part \( A \) in \( T_{\text{kw}} \) to the storage server and keeps part \( B \) secret. On receiving part \( A \), the storage server can quickly locate the matching KIS \(<\text{H}(\text{MEK}, \text{KW})\_i: \text{E}_i(\text{LKey}_{\text{KW}}_i, \text{flag || FileInfo } K_{\text{file}_j} >, \ldots <\text{E}_n(\text{HKey}_{\text{KW}}_n, \text{flag || FileInfo } K_{\text{file}_j} >\rangle\) in the encrypted inverted index, then the storage server returns the matching KIS to the searcher.

Now, we suppose only \( \text{LKey}_{\text{kw}} \) is revealed to the searcher, which can only decrypt the corresponding encrypted items that are encrypted with \( \text{LKey}_{\text{kw}} \). Other obtained results cannot be decrypted correctly since the data owner doesn’t grant the searcher \( \text{MKey}_{\text{kw}} \) and \( \text{HKey}_{\text{kw}} \). Note that the server tests whether the decryption is correct or not by verifying that the holding flag is equal to the flag extracted from the decryption of the items. Fig. 3 illustrates the search process.

![Figure 2. Encrypted Index Creation Process.](image)

**B. Encrypted Index Table Maintenance -- Virtual Deletion**

As we know, it’s relatively complex to update the index after deletion of one or more documents, especially for the encrypted index. Generally, it takes a large amount of computation resources and time to perform this operation, but the deletion is relatively fast and easy to perform.

A naive solution for the deletion operation includes several sub-operations, which are downloading the encrypted document, decrypting the encrypted document, re-creating the encrypted index for this document, deleting the document’s encrypted index from the encrypted index table, and deleting the encrypted document from the storage server.

This solution is too complex to use, so it is necessary to update the index at a lower frequency, such as on a daily or weekly basis. It is also required that the index update may be scheduled so as to reduce the duration and effects of service outage. For example, the operation is performed at a time when fewer searchers are accessing to the search service, such as at midnight.

However, to ensure correctness of search after deletion, it is necessary to screen out the deleted encrypted documents from the search results before the encrypted index is updated. We call such operation as virtual deletion.
Next, we will introduce two processes under virtual deletion. One is how to search a keyword. The other is how to actually delete the deleted documents' encrypted index.

1) Search Process under Virtual Deletion

By filtering out some documents according to certain condition in providing encrypted documents to the searcher, the server can provide the ability of virtual deletion. For example, the data owner sends a list of deleted encrypted document identifiers \{CFN_i\} to the server, and the server deletes the corresponding encrypted documents. After that, when the server receives a list of search results identifiers \{CFN_i, CFN_j, CFN_k\}, he firstly filters out the deleted documents \{CFN_i\}. Then, he only returns the filter-out encrypted documents results \{CFN_j, CFN_k\} to the searcher. That means the storage server will keep a deletion list for the deleted documents. Before the search results are returned, he will use the list to filter out the deleted results.

2) Deletion Process under Virtual Deletion

With the virtual deletion, our encrypted search engine becomes more practical during the deletion operation. However, another problem rises for the virtual deletion. That is how to actually delete the deleted documents’ encrypted index from the encrypted index table.

To solve this problem, our encrypted inverted index structure is extended. In our new index structure, KIS, is \(<I=H_{(MEK,\text{KW})},<I_{<j,k>=E_{(?\text{Key}_{\text{KW}},\text{flag||FileInfo||K_{file}})}, I_{<j,k>=\text{Hash} (I_{<j,k>}, x_{j})}>;\text{where} x_{j}=\text{Hash}(\text{CFN}_j, sk), sk \text{ is a secret key held by the data owner, which is equal to MEK. By this structure, storage server can locate the corresponding encrypted items easily from the encrypted inverted index after the encrypted document is deleted, and then the server removes the invalid ones with the help of the data owner. The concrete process of locating specific items can be described as follows:}

- Data owner computes \(x' = \text{Hash}(\text{CFN}', sk)\), and sends it to storage server securely.
- After receiving \(x'\), the storage server computes \(I'_{<j,k>}=\text{Hash}(I_{<j,k>}, x')\) one by one based on \(I\) and \(I_{<j,k>}\).
- Then the server tests whether \(I'_{<j,k>}\) is equal to \(I_{<j,k>}\) or not.
- If \(I'_{<j,k>}\) is equal to \(I_{<j,k>}\), it means a target item appears. Otherwise, skip it and compute the next.

Obviously, this process continues as above until all items have been tested.

IV. Prototype Implementation and Experiment

In this section, our prototype for the encrypted search engine, secure public space (SPS) system, will be introduced, and then we will give the evaluation results for it.

A. Prototype Implementation

There are two core modules in SPS system. One is encrypted search engine module, another is document management module.

In the encrypted search engine module, we implement a fast searchable encryption scheme based on an open source code search engine, Lucene. All of the operations principles are same as in Lucene, including the index creation process, index merging process, and index search process. The only difference is the index items and the search items. The entire index items and search items are encrypted as mentioned in Section IIIA.

In order to manage the online documents, we adopt WebDav to deal with the document uploading, downloading, deletion, list operations, and so on. The WebDav server uses Apache server, and its client uses an open source code -- neon.

We design our system under the C/S model. The SPS server stores the encrypted index and encrypted documents. The encrypted index search and merging process are also performed on the server. While all the information, including documents, search results and index items, is encrypted/decrypted in the SPS client. By this way, the storage server only provides a storage service without revealing any data privacy (Fig. 4).

B. Evaluation Results

In this section, an experiment for comparing several index operations between our ciphertext information and plaintext information is described. The experiment environment is introduced in Tbl.I and the experiment results in Tbl.II.

We can see that there is almost no performance difference in the index search process between the encrypted search engine and the traditional ones. In the index creation process, our encrypted search engine needs some encryption and hash operations. So there is a little more latency than the traditional one, only 0.75 seconds for a 10MB documents. In addition, in our solution, each the encrypted index item (such as document information) is different. It is hard to combine the same index items like plaintext index. So the index size in the ciphertext index is larger than the plaintext index.

C. Security Analysis

1) Confidentiality

In our solution, all the medical data are encrypted by different symmetric encryption keys using AES in the patient’s party. All the keys are derived from a private key, which is only kept by
In our solution, before medical data are encrypted, the patient needs to keep a private key, and then uses this key to generate the corresponding level decryption key in the keyword authorization. So, the searcher can only decrypt the FileInfo in the search results. In this search process.

Privacy of Search

In our solution, the authorization for a search keyword combines two parts, an encrypted keyword and a decryption key. The searcher uses the encrypted keyword to search the encrypted index. All the search results are only decrypted in the searcher party. So, nobody, except the patient and the searcher, can know which keyword is used and what the search results are returned in this search process.

Fine-grained Multi-level Retrieval

In our solution, before medical data are encrypted, the patient should define the medical data’s privacy level. Then he will use the corresponding level key to encrypt the FileInfo in the encrypted index. When a searcher wants to search a keyword, the patient also defines the searcher’s privacy level. Then he will generate the corresponding level decryption key in the keyword authorization. So, the searcher can only decrypt the corresponding level key’s FileInfo in the search results.

Ciphertext based Data-driven Access Control

In our solution, access control is based on search process. All the medical data decryption keys are hiding in the encrypted index. It is very convenient for the patient to manage his encrypted data and share them to others. Because the patient only needs to keep a private key, and then uses this key to generate the keyword authorization to share encrypted data to others.

V. Conclusions and Future Work

In this paper, we present a practical encrypted search engine scheme, which enables not only the online medical data owner but also a searcher to search the encrypted full-text data without revealing the data privacy and the search pattern to any other third-parties. We can easily manage our sensitive medical data at an online third-party server by our solution. We can also solve the encrypted medical data sharing issue with our solution when a patient visits different hospitals. The security analysis shows that our solution assures data confidentiality and access pattern privacy. Results of experiments show that the full-ciphertext inverted-index-based structure makes our solution very efficient in that the overhead of our solution is small compared to the indexing and search times of traditional search engines.

In addition, there are still some open problems to be resolved. Among them the conjunctive keyword search is probably the most challenging one. Conjunctive keyword search is a common operation in the traditional search engines. But for an encrypted search engine, it is a difficult problem. Several solutions already addressed the conjunctive keyword search issue for encrypted search engine [8, 9], but none of them are practical, because they are all based on the forward index, and work under the database index model. This means their index structures are composed of several fields, such as name, gender, birthday, and so on, and each field in one document only has one value. These index structures can only deal with special documents. We reiterate that no existing solution can deal with conjunctive keyword search in a full-ciphertext index structure. It is a big challenge for researchers in this field.

REFERENCES