Salam *et al. Hum. Cent. Comput. Inf. Sci.* (2015) 5:19
DOI 10.1186/s13673-015-0039-9

Human-centric Computing and Information Sciences
a SpringerOpen Journal

RESEARCH Open Access

Implementation of searchable symmetric encryption for privacy-preserving keyword search on cloud storage

Md Iftekhar Salam[1], Wei-Chuen Yau[2*], Ji-Jian Chin[2], Swee-Huay Heng[3], Huo-Chong Ling[4], Raphael C-W Phan[2], Geong Sen Poh[5], Syh-Yuan Tan[3] and Wun-She Yap[6]

*Correspondence: wcyau@mmu.edu.my
[2] Faculty of Engineering, Multimedia University, Cyberjaya, Selangor, Malaysia
Full list of author information is available at the end of the article

Abstract

Ensuring the cloud data security is a major concern for corporate cloud subscribers and in some cases for the private cloud users. Confidentiality of the stored data can be managed by encrypting the data at the client side before outsourcing it to the remote cloud storage server. However, once the data is encrypted, it will limit server's capability for keyword search since the data is encrypted and server simply cannot make a plaintext keyword search on encrypted data. But again we need the keyword search functionality for efficient retrieval of data. To maintain user's data confidentiality, the keyword search functionality should be able to perform over encrypted cloud data and additionally it should not leak any information about the searched keyword or the retrieved document. This is known as privacy preserving keyword search. This paper aims to study privacy preserving keyword search over encrypted cloud data. Also, we present our implementation of a privacy preserving data storage and retrieval system in cloud computing. For our implementation, we have chosen one of the symmetric key primitives due to its efficiency in mobile environments. The implemented scheme enables a user to store data securely in the cloud by encrypting it before outsourcing and also provides user capability to search over the encrypted data without revealing any information about the data or the query.

Keywords: Searchable encryption, Data confidentiality, Cloud storage, Keyword search

Background

With the rapid development of cloud computing and mobile networking technologies, users tend to access their stored data from the remote cloud storage with mobile devices. The main advantage of cloud storage is its ubiquitous user accessibility and also its virtually unlimited data storage capabilities. Despite such benefits provided by the cloud, the major challenge that remains is the concern over the confidentiality and privacy of data while adopting the cloud storage services [1]. For instance, unencrypted user data stored at the remote cloud server can be vulnerable to external attacks initiated by unauthorized outsiders and internal attacks initiated by the untrustworthy cloud service providers (CSPs) [2]. There are several reports that confirm data breaches related to cloud servers, due to malicious attack, theft or internal errors [3]. This raises concern for many users/

organizations as the outsourced data might contain very sensitive personal organization/information.

Several researches have addressed the issue of ensuring confidentiality and privacy of cloud data without compromising the user functionality. Here, confidentiality refers to the secrecy of the stored data so that only the client can read the contents of the stored data. To solve the problem of confidentiality, data encryption schemes can come in handy to provide the users with some control over the secrecy of their stored data. This has been adopted by many recent researches which allow users to encrypt their data before outsourcing to the cloud [4–9]. However, standard encryption schemes will dampen users' searching ability over the stored data, since after encryption a user simply cannot use a plaintext keyword to perform a search anymore and therefore cannot retrieve the contents in an efficient way.

The keyword search functionality enables the user to search for a certain keyword on the remote cloud data. Consider a cloud application that consists of a cloud service provider (CSP), and users who store their data on the cloud storage. The users can use a traditional encryption scheme to ensure the confidentiality of the contents. The naïve approach for retrieving encrypted contents related to a certain keyword would require the user to download all the stored data, and then decrypt and perform the search on the local machine. However, this solution is infeasible from a practical point of view, as the user needs to download all the contents rather than the contents containing the searched keyword. For example, consider a scenario where the cloud storage contains 1 GB of user's data, but only 1 MB of data is related to the searched keyword. Using the naïve solution, it is required to retrieve all the 1 GB data, which is inefficient. Alternatively, the user can store a plaintext keyword index in the cloud server and use it while retrieving the data. However, this approach will allow the CSPs to know about the keyword which is not desirable either. Therefore, to ease the data retrieval from a secure cloud, we need a scheme which enables user capability to search over encrypted contents.

To provide a secure and efficient retrieval of data, one needs to ensure that the user can perform a search over the encrypted data without revealing the contents and the searched keyword to the server. The cryptographic primitive that provides this feature is widely known as searchable encryption (SE). This research aims to study the searchable encryption schemes in detail and implements a solution that enables privacy preserving data storage and retrieval system in cloud computing (aka PrivCloud). For our implementation, we have chosen an existing searchable symmetric key encryption algorithm. To enable the privacy preserving keyword search, this scheme will generate an encrypted keyword index which will be outsourced to the cloud server along with the encrypted data set. The encrypted keyword index lists out the encrypted keyword and pointer to the corresponding document containing that keyword. To search a keyword, client can simply encrypt the keyword to generate the search token and send it to the remote cloud server. The server can retrieve pointer to the corresponding document by matching the search token with the encrypted keyword index table. For our proposed solution, we have chosen to index all the words from the document instead of a specific keyword set. This will allow the client to search for any word in the document rather than any specific keyword. Also, indexing all the words of the document make sure that the user does not need to maintain a keyword index table at the client side since it can

search for any words. However, indexing all the words will come with the trade-off of a slightly larger index size. One of the limitations of the proposed solution is that it does not support the addition of new files since the index update is static.

Background on searchable encryption

This section provides a detailed background on the searchable encryption scheme. In particular, we discuss about the architecture, security requirements and design approaches for searchable encryption scheme.

Searchable encryption: architecture

Searchable encryption (SE) enables the users to generate a search token from the searched keyword in such way that given a token, the cloud server can retrieve the encrypted contents containing the searched keyword. Basically, the search token represents an encrypted query over the encrypted data and can be generated only by users with the appropriate secret key. Figure 1 shows the basic architecture and working principle of a searchable encryption scheme. The architecture comprises mainly four entities: data owner, data user, cloud service provider and key generator. A brief description of the entities and their operations are given below.

A. *Data owner*: The data owner is the entity which generates and encrypts the data and uploads them to the cloud server. It can be either an organization or an individual. To use the service, the data owner uses its application which consists of a data processor

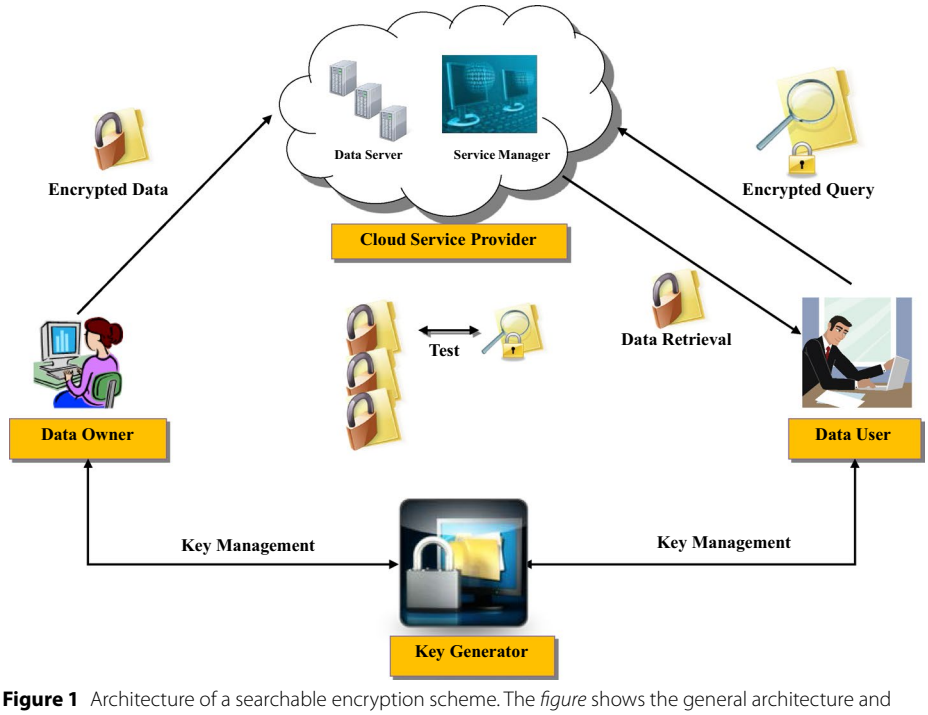

Figure 1 Architecture of a searchable encryption scheme. The *figure* shows the general architecture and components of a searchable encryption scheme. It describes the workflow of the searchable encryption scheme.

for uploading new contents to the cloud. It encrypts the data and metadata with a cryptographic scheme [1] that enables searching capability.

B. *Data user*: This entity is also a subscriber to the cloud storage which sends encrypted queries to the cloud service provider to search for a specific encrypted data. There may be more than one data user in the system and in some scenario, the data owner and the data user might be the same entity.
C. *Cloud service provider*: This entity provides the data storage and retrieval service to the subscribers. The cloud service provider consists of cloud data server and cloud service manager. The first entity is used to store the outsourced encrypted data whereas the latter one is used for data management in the cloud. Upon receiving the encrypted search queries from the data user, the cloud service provider tests on the encrypted queries and encrypted metadata in the cloud storage. The encrypted data that satisfies the search criteria is retrieved and sent back to the data owner upon completion of the test. The cloud service provider should not learn any information from the operation.
D. *Key generator*: This entity is considered to be a trusted third party which is responsible for the generation and management of the encryption/ decryption keys. User specific keys are generated and distributed during the setup of the system.

Searchable encryption: security requirements

In general, the following requirements should be satisfied when constructing a searchable encryption scheme.

- *Retrieved data*: Server should not be able to distinguish between documents and determine search contents.
- *Search query*: Server should not learn anything about the keyword being searched for. Given a token, the server can retrieve nothing other than pointers to the encrypted content that contains the keyword.
- *Query generation*: Server should not be able to generate a coded query. The query can be generated by only those users with the relevant secret key.
- *Search query outcome*: Server should not learn anything about the contents of the search outcome.
- *Access patterns*: Server should not learn about the sequences and frequency of documents accessed by the user.
- *Query patterns*: Server should not learn whether two tokens were intended for the same query.

Searchable encryption: design approaches

Searchable encryption scheme can be built using either a non-keyword based approach or an index/keyword based approach. In the non-keyword based approach, the scheme scans the entire document word by word to find out the word W of interest. This provides the functionality to search any words in the document. However, it takes a long search time for a large number of document set. On the other hand, index/keyword based solution builds up an index [10], for each word W of interest and lists out the

corresponding documents that contain W. This provides a faster search operation when the document set is large. However, storing and updating the index can be an overhead.

From the viewpoint of cryptographic algorithm selection, the SE scheme can mainly be modeled using either asymmetric/public key or symmetric/secret key setting. In the following, we briefly discuss the difference between these two settings.

A. *Asymmetric searchable encryption (ASE)*: In this setting, a user encrypts the data using asymmetric/public key encryption schemes (e.g. RSA) before outsourcing it to the cloud server. This setting is appropriate for a scenario where the user searching over the data is different from the user who generates it. For example, multiple users can use the public key of a certain user to encrypt and upload the data, however; only the owner with the corresponding private key can generate the search token and therefore can perform a search over the encrypted data. The main advantage of ASE is its functionality whereas the drawback is inefficiency. ASE schemes can be used in a larger number of settings since the reader and writer can be different for this case. On the other hand, all known ASE schemes require the evaluation of pairings on elliptic curves which is a relatively slow and costly operation compared to the hash functions or block ciphers [1]. Several researches have been conducted in developing SE scheme using public key cryptography [11–16].
B. *Symmetric searchable encryption (SSE)*: In this setting, a user encrypts the data using symmetric/private key encryption schemes (e.g. AES) before outsourcing it to the cloud server. This setting is appropriate when the user that searches over the data is also the one who generates it. The main advantage of this setting is the efficiency, but it lacks of functionality as it can only be used for a single user scenario. Moreover, most of the SSE schemes leaks the access patterns. The encryption is efficient because most SSE schemes are based on symmetric primitives like block-ciphers and pseudo-random functions and requires very less computational overhead. The SSE scheme was first proposed by Song et al. [17] which provides techniques for remote searching over encrypted data using symmetric key primitives. Later, security notions of SSE schemes were revisited and stronger security definitions were provided by Goh [10], Chang et al. [18] and Curtmola et al. [19].

Related work

In this section, we present a brief summary of related works dealing with the searchable encryption schemes. Searchable encryption scheme can be designed based on either public key or symmetric key algorithm. The first searchable encryption scheme based on public key algorithm was proposed by Boneh et al. [11]. This is known as the PEKS scheme which uses the public key of a user to encrypt and store the data in the server, and allows an authorized user with the private key to search and decrypt the corresponding content. This is a keyword based scheme ensuring faster search functionality; however, limiting the search capability. Also, the scheme is computationally expensive and it reveals the user access pattern. An extension of the PEKS scheme was proposed by Liu et al. [14]. This also uses the public key primitive to support the keyword searching on encrypted data. This scheme allows the cloud service provider to participate in the partial decipherment and claims to have reduced computational overhead on the client

due to this partial decipherment. Another variant of the PEKS scheme called iPEKS was proposed by Tseng et al. [15]. This scheme aims to accelerate the search time by looking into the previously searched keywords. For this, the cloud service provider caches the previously searched keywords to avoid the search on all the stored ciphertexts. However, this comes with the tradeoff of large storage overhead. The first searchable encryption scheme based on symmetric key primitive was proposed by Song et al. [17]. This provides a non-keyword based solution. However, this scheme can search for only fixed length words and also the search time is linear in document size, since the it needs to scan the whole document to complete the search. The scheme is too slow when searching for a large number of documents. Goh [10] addresses some of the issues of the above scheme by introducing the concept of a secure index. This scheme generates a search index which can be used to locate the encrypted content. Later, the security notions of searchable symmetric encryption were revisited and stronger security definitions were provided by Curtmola et al. [19]. This is a very simple scheme based on keyword indexing approach.

Proposed solution

This research aims to design and develop a privacy preserving data storage and retrieval system in cloud computing. The scopes involve the use of searchable encryption algorithms to search for specific keywords within an encrypted content, i.e., without requiring the user to download the database and decrypt its contents before searching can be performed. The proposed solution delegates searching on encrypted data to CSP but with privacy preserved. For the searchable encryption algorithm, we have chosen to implement the adaptively secure Searchable Symmetric encryption (referred as SSE-2 scheme in the original paper) scheme proposed by Curtmola et al. [19]. For the rest of the paper, we will use the term SSE-2 scheme or adaptively secure SSE scheme to refer to the proposed method by Curtmola et al. The SSE-2 scheme provides a simple but efficient method to enable searching over encrypted data while preserving the data privacy. The reason behind selecting a private/symmetric key primitive for our implementation mainly lies in the fact that it results in significantly lesser computational overhead when compared to its public/asymmetric key counterpart and therefore will be more suitable for mobile devices.

Adaptively secure searchable symmetric encryption

In this section, we describe the adaptively secure SSE algorithm proposed by Curtmola et al. [19]. As the name suggests, the scheme is based on the symmetric key encryption setting and therefore mostly suitable for a single reader/single writer scenario. It uses the index based approach [10], where user has to pre-process the contents to generate a keyword index to provide for the search capability. Figure 2 shows the construction of the adaptively secure searchable symmetric encryption scheme.

As shown in Figure 2, a user U encrypts a set of data $D = \{D_1, D_2, \cdots, D_n\}$ and creates an encrypted index file I which contains a set of m encrypted keywords extracted from the data set D. In order to conduct a search over the encrypted data, user U outsources the index I and the encrypted data set D to the cloud server. During a search, U creates an encrypted query and sends it to the server. Cloud server takes this encrypted query as input and then uses the encrypted index located at the server to retrieve pointers to the

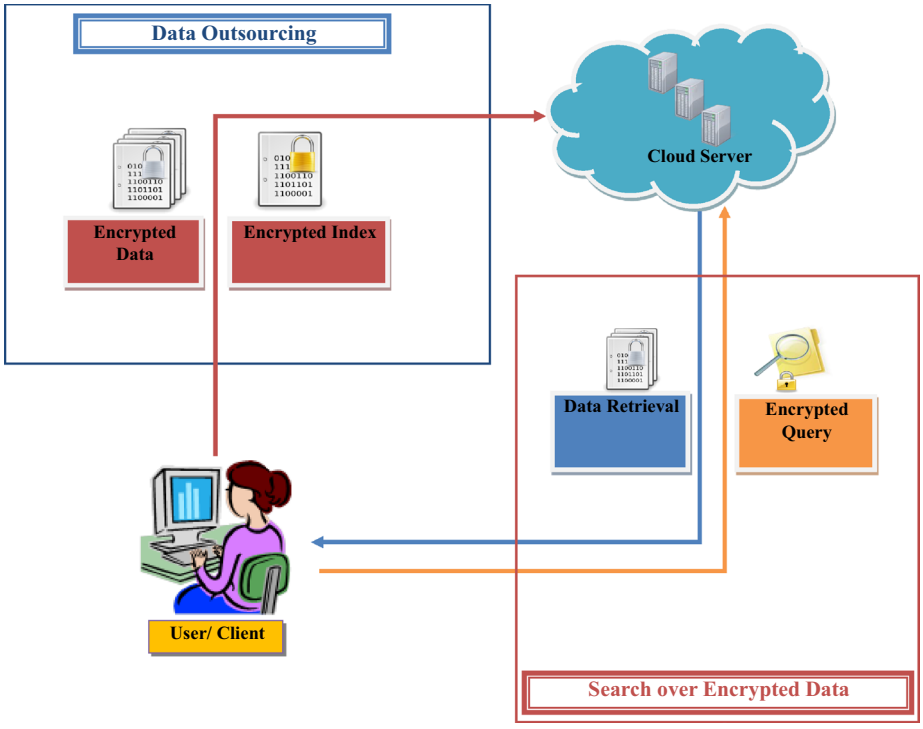

Figure 2 Working principle of adaptively secure SSE scheme to search over encrypted data. The *figure* shows the work-flow of the searchable symmetric encryption scheme. This scheme generates and stores an encrypted index at the cloud server to enable search over encrypted data.

document(s) containing the searched keyword. Once the search result is obtained, the encrypted document(s) containing the searched keyword is returned to the client.

The scheme mainly consists of six functional entities: key generation, document pre-processing, encryption, search token generation, search and decryption. Functions of each entity/ block are described as follows:

A. *Key generation*: It generates the keys for encryption and decryption using a symmetric key primitive. Two keys K_1 and K_2 are generated for encryption of index contents and documents respectively. Both keys are generated and stored securely at the user device. Note that this is a symmetric key primitive and therefore same key is used for both encryption or decryption purposes.

B. *Document re-processing*: This block provides the necessary function to pre-process a set of documents and initialize the encryption procedure. Before encrypting the set of documents, the user needs to pre-process the document set to pull out the keyword and build an index. Let D defines a set of document to be encrypted and uploaded in the cloud server. The user has to list out all the keywords from each document in D and build an index table listing the documents and the corresponding keywords. The scheme will also assign a content ID for each of the documents in the set. For example, consider a set of 5 documents $D = \{D_1, D_2, D_3, D_4, D_5\}$ to be uploaded in the cloud. At the pre-processing step, the user will pre-process these set of documents to build an index table as shown in Table 1. Here, $w_i\,(i = 1, 2, 3, \cdots, m)$

Table 1 Document index creation

Content/ document	Content ID	Keyword
D_1	1	w_1, w_5
D_2	2	w_1, w_3, w_8
D_3	3	w_2, w_9, w_5
D_4	4	w_6, w_7, w_9
D_5	5	w_1, w_4, w_{10}

represents the keyword and m is the total number ($m = 10$ for this example) of keywords. The content IDs are assigned sequentially starting from 1. In the next step, it will create an inverted index table which lists out the respective content IDs for each keyword w_i. For the above example, the inverted index is shown in Table 2.

C. *Encryption*: This block provides the functionality to encrypt the index and document set. The encryption is performed using the encryption keys generated at the key generation step. Basically, it consists of index encryption and document encryptions and performs as described below: *Index encryption* This encrypts the keyword set generated at the pre-processing step and creates an encrypted index/lookup table. The keyword encryption is computed as $ENC_{K_1}(w_i||n_i)$, where ENC_{K_1} represents encryption with key K_1, w_i is the keyword i and n_i is the corresponding document ID containing keyword w_i. For each of the encrypted keywords, the encrypted index table lists out the corresponding document ID. For our example document set, the encrypted index table is shown in Table 3. *Document encryption*: This encrypts each document from the document set D with key K_2, and stores it in the database. The document encryption is computed as $ENC_{K_2}(D_i)$ which represents encryption of document D_i with key K_2. For the above example, the encrypted document lists are shown in Table 4.

D. *Search token generation*: This block is used when a user wants to search for a document containing a certain keyword. It provides the user/client with the functionality of generating a search token/ trapdoor which can be used at the server to perform a search over encrypted documents. To carry out the search operation, the user will input the search keyword and then compute the search token. The search token for a keyword w_q is computed as follows: Search Token, $t = (t_1, t_2, \cdots, t_n) = ENC_{K_1}(w_q||1), ENC_{K_1}(w_q||2), \cdots, ENC_{K_1}(w_q||n)$, where n is

Table 2 Inverted index

Keyword	Content ID
w_1	1, 2, 5
w_2	3
w_3	2
w_4	5
w_5	1, 3
w_6	4
w_7	4
w_8	2
w_9	3, 4
w_{10}	5

Table 3 Encrypted index

Encrypted keyword	Content ID
$ENC_{K_1}(w_1\|\|1)$	1
$ENC_{K_1}(w_1\|\|2)$	2
$ENC_{K_1}(w_1\|\|5)$	5
$ENC_{K_1}(w_2\|\|3)$	3
$ENC_{K_1}(w_3\|\|2)$	2
$ENC_{K_1}(w_4\|\|5)$	5
$ENC_{K_1}(w_5\|\|1)$	1
$ENC_{K_1}(w_5\|\|3)$	3
$ENC_{K_1}(w_6\|\|4)$	4
$ENC_{K_1}(w_7\|\|4)$	4
$ENC_{K_1}(w_8\|\|2)$	2
$ENC_{K_1}(w_9\|\|3)$	3
$ENC_{K_1}(w_9\|\|4)$	4
$ENC_{K_1}(w_{10}\|\|5)$	5

Table 4 Encrypted document list

Content ID	Encrypted Document
1	$ENC_{K_2}(D_1)$
2	$ENC_{K_2}(D_2)$
3	$ENC_{K_2}(D_3)$
4	$ENC_{K_2}(D_4)$
5	$ENC_{K_2}(D_5)$

the total number of documents in the document set D. For our example of 5 document sets, the search token for keyword w_q will be $t = (t_1, t_2, t_3, t_4, t_5)$. Once computed, the search token is sent to the server to find out the corresponding document containing the searched keyword.

E. *Search*: The search function takes the search token and the encrypted index table as input and outputs the document list containing the searched keyword. For a search token $t = (t_1, t_2, \cdots, t_n)$ it will compare with the encrypted index if any of the values in t matches with the encrypted keyword. It outputs the corresponding document IDs which match the search token and then send back the respective documents to the client.

F. *Decryption*: The client decrypts back the document once it has obtained the encrypted document set containing the searched keyword. The document decryption is computed as $DEC_{K_2}(D_i)$ which represents decryption of document D_i with key K_2.

Implementation of PrivCloud system

This section provides the detailed description for our implementation of the adaptively secure SSE scheme, aka PrivCloud. This implementation considers a single writer/ single

reader scenario where the user that searches over the data is also the one who generates it. First of all, we provide the basic framework for our implementation.

Framework of the PrivCloud system

The PrivCloud system implements the SSE-2 scheme [19] which consists of the following modules: encryption, search and decryption. These modules are integrated with a Key Generator block which provides the encryption/ decryption keys.

A. *Encryption*: This module provides the document and index encryption functionality. Figure 3 shows the framework for the implementation of the SSE-2 Encryption module. First, the key generator is initiated to generate keys. Since symmetric key generation does not require much computational power, we assume that the key generator is located at the user device. This resolves the key distribution issue. The key generation is a one-time process to generate and store the keys. After the key generation, a user inputs a set of document for the encryption and index creation. When user inputs the document set, the Encryption module uses the Keyword Extractor block to pull out the keyword and builds an index of all the words in the document. After keyword extraction, input documents and the document index are fetched to the Encryption block. The Encryption block uses the keys generated by the key generator to create the encrypted document set and encrypted document index. Concurrently, the Encryption block also creates a filename mapping object/ filename index which contains the mapping of document IDs to the corresponding original filenames and extensions. This is stored in plaintext at the user device. Finally, the encrypted files are uploaded to the server.

B. *Search*: The search module provides the functionality of searching a keyword over the encrypted data stored at the cloud server. Figure 4 shows the framework for the implementation of the SSE-2 search module. When user inputs the searched

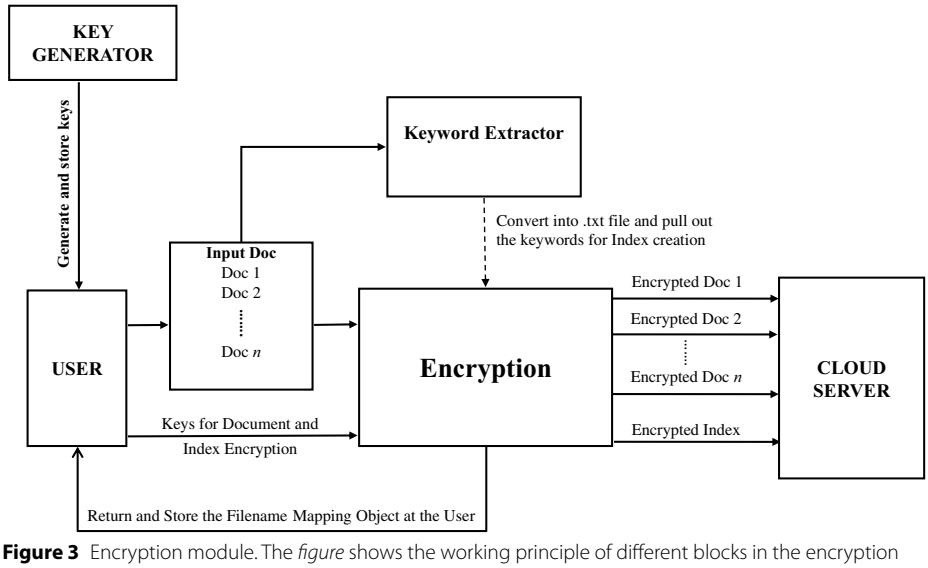

Figure 3 Encryption module. The *figure* shows the working principle of different blocks in the encryption module of the PrivCloud system. The encryption module provides the document and index encryption functionality.

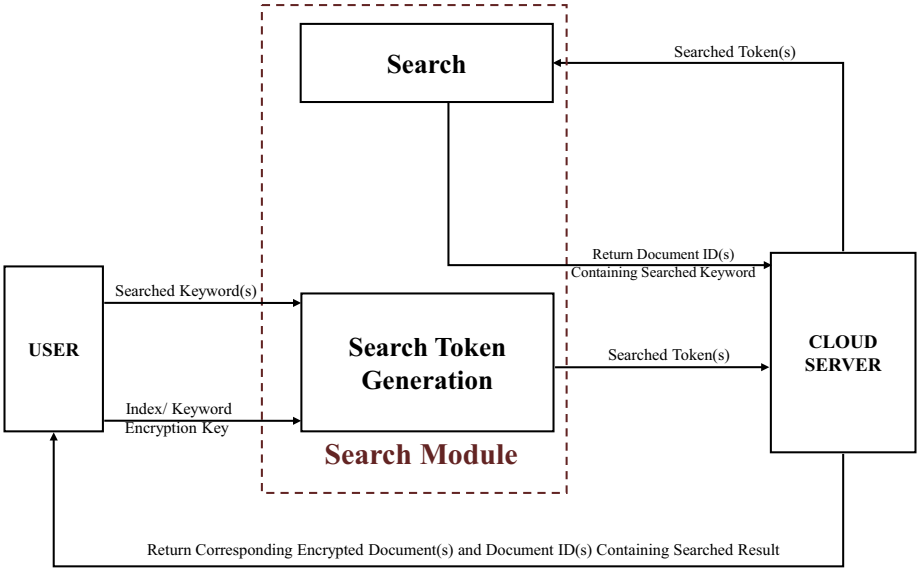

Figure 4 Search module. The *figure* shows the working principle of different blocks in the search module of the PrivCloud system. This module enables a cloud user to generate an encrypted query to the server. The cloud server can use this encrypted query to search over the encrypted data without knowing the contents of the data or the query.

keyword(s), Search Token Generation computes a search token(s)/ trapdoor(s) using the keyword/index encryption key. This generated search token is then sent to the cloud server to perform the search operation. Upon receiving the search token, the cloud server find out pointer(s) to the appropriate encrypted documents by comparing the search token with the encrypted index values. Following this, the cloud server retrieves the encrypted document(s) containing the searched keyword(s) and sends it back to the client along with the corresponding document ID(s).

C. *Decryption*: The Decryption module that is located at the user's device provides the decryption functionality for a given set of encrypted documents. Figure 5 shows the basic construction for the SSE-2 Decryption module. The Decryption module requires Encrypted Document, corresponding Document ID, Document Decryption Key and Filename Mapping Object as input. First, the Decryption module uses the filename mapping object and the document ID to retrieve the original filename and extension of the encrypted document. Then it decrypts the document using the document decryption key and stores it to the user device.

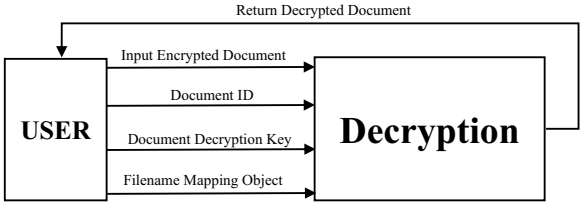

Figure 5 Decryption module. The *figure* shows the working principle of different blocks in the decryption module of the PrivCloud system. This module provides the decryption functionality.

Implementation details

The scheme is implemented and tested under the following environment: Java 1.7.0_51 , Windows 8.1 64-bit operating system. Based on the framework, we divide our implementation into four main class structures: Key Generation, Encryption, Search and Decryption.

A. *Key generation class*: We use the standard javax.crypto library to generate the AES keys. Two 128-bit AES keys (i.e., keyword encryption key, K_1 and document encryption key, K_2) are generated and stored. The key generation class has 3 methods:

   - *generate* () : This method generates two 128-bit AES keys.
   - *store* () : This method converts the generated keys (encoded) into strings, writes into a Key object and then stores it to a location at the user device.
   - *read* () : This provides the functionality to read the keys from user. The method *read* () takes the Key object as input and reads the key values. Following this, it will convert the key string to an array of bytes and then construct the corresponding secret key from the given byte of array.

B. *Encryption class*: The encryption is performed at user device before outsourcing the data to the cloud server. This creates the database containing the encrypted document set and encrypted index. Before performing encryption one needs to pre-process the data to retrieve the keyword and create an index. The SSE-2 scheme extracts only a set of distinct keywords to create an index. However, in our implementation we create an index of the whole document rather than a finite number of distinct keyword set. This gives a user the flexibility to search for any words in the document rather than a selected keyword and also user does not need to keep the keyword index stored at its own device. We consider the initializing/pre-processing of documents within the Encryption block. The pre-processing step includes:

   - *Document ID allocation*: For each document, it assigns a unique document ID. The document ID will be assigned starting from 0 and will increase sequentially.
   - *Keyword extraction*: The keyword extraction process searches for consecutive sequences of non-blank and non-punctuation characters. It can easily be tweaked to search for consecutive sequences of ASCII characters. This provides adequate results for binary files containing uncompressed English text. To read out the characters from an input document, we need to convert it to a text file first and for this file conversion we have used the Apache Tika. The keyword extractor is implemented by pulling characters from stream, checking if they are acceptable, and accumulating consecutive acceptable characters into a keyword. An additional functionality is also provided where one can limit the number of total keywords by selecting a minimum length of characters in a keyword. For example, if we select the minimum keyword length to be 4 characters, then the keyword extraction algorithm will pull out only those keywords which have 4 or more ASCII characters.

After pre-processing process the encryption module outputs the encrypted files. We use AES/CBC/PKCS5 for the document and index encryption. Encryption class consists of mainly two methods: *encrypt* () and *witeDocIDFileNameMapping* ().

- *encrypt* () : This function takes as input all the files from user given path and encrypt all the files and also creates an encrypted index. For each input file the function retrieves the keywords first and for each keyword it computes the 128-bit AES encryption of the keyword and document ID. Then, it creates an index entry in the encrypted index table to list out the encrypted keyword and the corresponding document ID. It uses $put(key, value)$ specified in Java TreeMap interface to associate the specified value (document ID) with the specified key (encrypted keyword). Finally, it saves the encrypted index in the database. The input documents are ready for encryption once the indexing is finished. For the encryption of a document, this method first creates a new document in the database and writes the encrypted stream in this. For each encrypted document, the corresponding document ID is used as the new filename of the encrypted document.
- *writeDocIDFileNameMapping* () : This method is used to create and store a filename mapping object /filename index which stores the ID of an encrypted document and the corresponding filename. It uses hash table to map the keys (document ID) into values (filename). This object is stored at client and utilized during decryption for the mapping of document ID to retrieve original filename and extension of the document.

Figure 6 shows the detail flowchart of the encryption procedure.

C. *Search class*: The search operation is initiated at the user device and performed in the cloud server. The operation consists of two methods: *searchToken* () and *search* ().

- *searchToken* () : This takes the keyword and document ID as input and computes the search token at the client device. The keyword encryption key is used to compute the encrypted value of the user given keyword and returns the generated search token. The search token is then sent to the server.
- *search* () : This takes place at the server when it receives a search request from the user. This method takes the search token and the user database as input to find out the document IDs containing the search result. It uses *get*() defined in the Java TreeMap class, which returns the value to which the specified key is mapped, or null if this map contains no mapping for the key. Finally, server retrieves the corresponding document and sends back to the client for decryption.

D. *Decryption class*: This initiates the *decrypt* () method when the client receives the encrypted document from the server.

- *decrypt* () : This method decrypts back the data in its plaintext form by using the user decryption key stored at the client device. Once the user has input the encrypted document, the program reads the corresponding document ID from the filename of the encrypted document. Then, it uses the filename index stored at the user device and gets the original filename and extension for an encrypted document by using its ID. Then, it reads the input stream from the encrypted

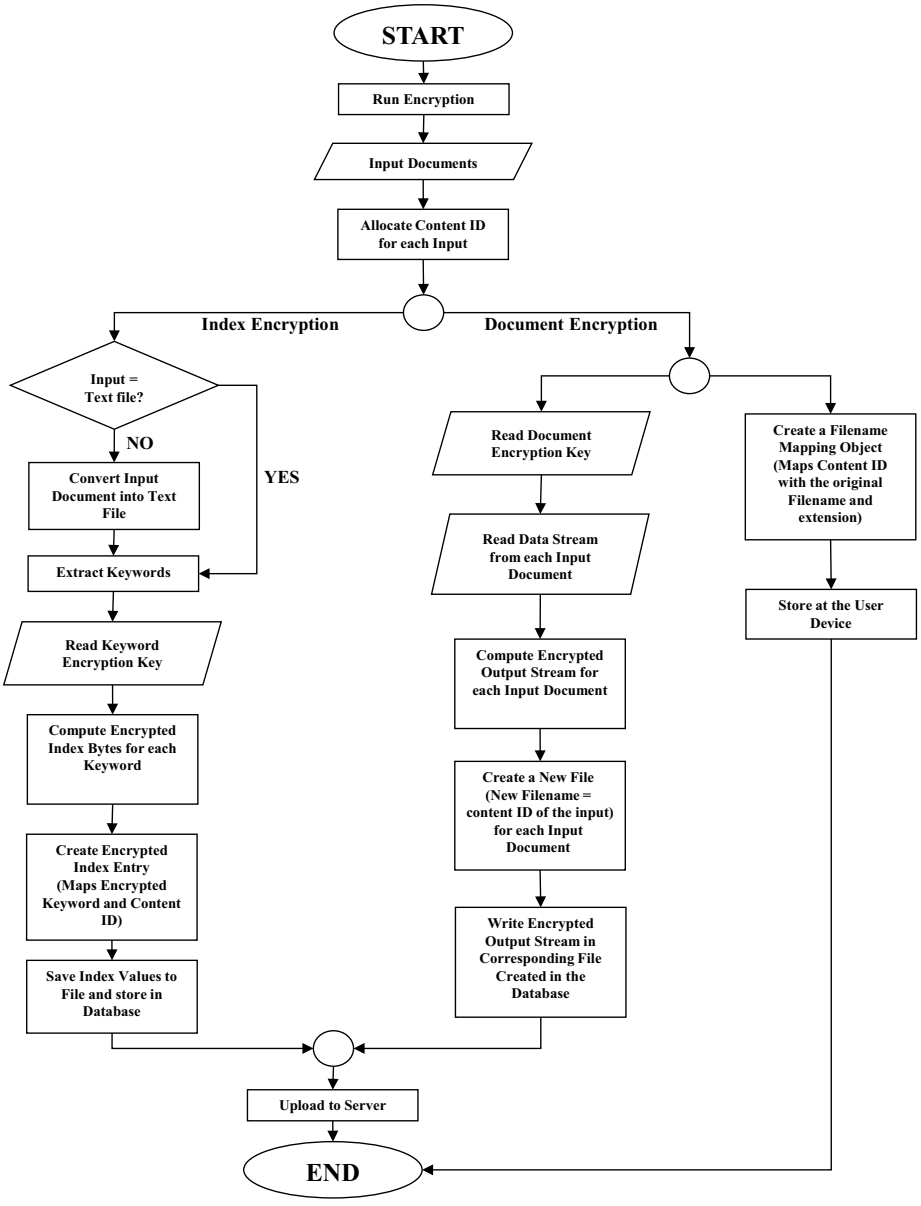

Figure 6 Flowchart for the document and index encryption process. The *figure* shows the flowchart of the encryption module for the PrivCloud system. It provides detail work-flow of the module to compute the encrypted document and encrypted index.

document and decrypts back the plaintext content using the corresponding decryption key. Lastly, this method generates a new document with the retrieved original filename and extension and then writes the decrypted plaintext stream into this file and stores it in the user device.

Conclusion

This article provides the details of our implementation of Curtmola's SSE scheme for the PrivCloud System. For our implementation, we have chosen the AES/ CBC mode for the encryption/decryption purpose. Our implementation indexes the whole document

rather than a set of keyword from each document as proposed in the SSE-2 scheme, which provides the user capability to search any keyword from the document with the trade-off of a slightly larger index size. Also, the client does not need to maintain a keyword index on its side. However, the index update process is static in our implementation, which does not allow the addition of new files or updating files. On the other hand, our encryption module requires slightly longer time since this also includes the conversion of documents into text file for the keyword extraction process. A faster encryption process can be obtained if the keyword extraction module can work without the document conversion process.

Author's contributions

This work was carried out as a part of the TM R&D project: Privacy Preserving Data Storage and Retrieval System in Cloud Computing. All the authors were part of this project and contributed to this manuscript. All authors read and approved the final manuscript.

Author details

[1] Information Security Institute, Queensland University of Technology, Brisbane, Australia. [2] Faculty of Engineering, Multimedia University, Cyberjaya, Selangor, Malaysia. [3] Faculty of Information Science and Technology, Multimedia University, Melaka, Malaysia. [4] Faculty of Engineering and Science, Curtin University, Miri, Sarawak, Malaysia. [5] University Malaysia of Computer Science & Engineering, Putrajaya, Malaysia. [6] Lee Kong Chian Faculty of Engineering and Science, Universiti Tunku Abdul Rahman, Sungai Long, Selangor, Malaysia.

Acknowledgements

This research was supported by TM R&D Fund (RDTC/130827).

Compliance with ethical guidelines

Competing interests

The authors declare that they have no competing interests.

Received: 4 December 2014 Accepted: 5 July 2015
Published online: 17 July 2015

References

1. Kamara S, Lauter K (2010) Cryptographic cloud storage. In: Sion R, Curtmola R, Dietrich S, Kiayias A, Miret JM, Sako K, Sebé F (eds) Financial Cryptography and Data Security, LNCS 6054. Springer, Berlin, Heidelberg, pp 136–149
2. Hacigümüş H, Iyer B, Li C, Mehrotra S (2002) Executing sql over encrypted data in the database-service-provider model. In: Proceedings of SIGMOD, ACM, pp 216–227
3. Subashini S, Kavitha V (2011) A survey on security issues in service delivery models of cloud computing. J Netw Comput Appl 34:1–11
4. Kubiatowicz J, Bindel D, Chen Y, Czerwinski S, Eaton P, Geels D et al (2000) Oceanstore: an architecture for global-scale persistent storage. In: Architectural support for programming languages and operating systems, ACM, pp 190–201
5. Muthitacharoen A, Morris R, Gil TM, Chen B (2002) Ivy: a read/write peer-to-peer filesystem. In: Proceedings of the 5th symposium on Operating System Design and Implementation, vol. 36, pp 31–44
6. Adya A, Bolosky WJ, Castro M, Cermak G, Chaiken R, Doucer JR et al (2002) Farsite: federated, available, and reliable storage for an incompletely trusted environment. In: Proceedings of the 5th Symposium on Operating systems design and implementation, vol. 36, pp 1–14
7. Benaloh J, Chase M, Horvitz E, Lauter K (2009) Patient controlled encryption: ensuring privacy of electronic medical records. In: Proceedings of the 2009 ACM workshop on Cloud computing security, ACM, pp 103–114
8. Li M, Lou W, Ren K (2010) Data security and privacy in wireless body area networks. IEEE Wireless Communications Magazine, vol. 17, IEEE, pp 51–58
9. Li M, Yu S, Ren K, Lou W (2010) Securing personal health records in cloud computing: patient-centric and fine-grained data access control in multi-owner settings. In: Jajodia S, Zhou J (eds) Security and Privacy in Communication Networks, LNCS 50. Springer, Berlin Heidelberg, pp 89–106
10. Goh EJ (2003) Secure indexes. In: Cryptology ePrint Archive: Report 2003/216
11. Boneh D, Crescenzo GD, Ostrovsky R, Persiano G (2004) Public-key encryption with keyword search. In: Cachin C, Camenisch JL (eds) Advances in Cryptology EUROCRYPT, LNCS 3027. Springer, Berlin Heidelberg, pp 506–522
12. Boneh D, Waters B (2007) Conjunctive, subset, and range queries on encrypted data. In: Salil Vadhan P (ed) Theory of cryptography, LNCS 4392, Springer, Berlin Heidelberg, pp 535–554
13. Liu Q, Wang G, Wu J (2009) An efficient privacy preserving keyword search scheme in cloud computing. In: International Conference on Computational Science and Engineering (CSE), Vol. 2, pp 715–720

14. Liu Q, Wang G, Wu J (2012) Secure and privacy preserving keyword searching for cloud storage services. J Netw Comput Appl (JNCA) 35(3):927–933
15. Tseng FK, Chen RJ, Lin BS (2013) iPEKS: Fast and secure cloud data retrieval from the public-key encryption with keyword search. International Conference on Trust. Security and Privacy in Computing and Communications, IEEE, pp 452–458
16. Lee SH, Lee IY (2013) A secure index management scheme for providing data sharing in cloud storage. J Inform Process Syst 9(2):287–300
17. Song DX, Wagner D, Perrig A (2000) Practical techniques for searches on encrypted data. In: Proceedings of the IEEE Symposium on Security and Privacy, IEEE, pp 44–55
18. Chang YC, Mitzenmacher M (2005) Privacy preserving keyword searches on remote encrypted data. In: Ioannidis J, Keromytis A, Yung M (eds) Applied Cryptography and Network Security, LNCS 3531. Springer, Berlin Heidelberg, pp 442–455
19. Curtmola R, Garay J, Kamara S, Ostrovsky R (2006) Searchable symmetric encryption: improved definitions and efficient constructions. In: Proceedings of the 13th ACM conference on Computer and communications security, ACM, pp 79–88