A Dynamic Scheme for Secure Searches over Distributed Massive Datasets in Cloud Environment using Searchable Symmetric Encryption Technique

Irene Getzi*‡, Christopher D Durairaj**

* Department of Computer Science, Manonmaniam Sundaranar University, Abishekappati, Tirunelveli, 627 012, Tamil Nadu, India

**Research Centre, School of Computer Science, VHNSN College (Affiliated to Madurai Kamaraj University, Madurai), Virudhunagar, 626 001, Tamil Nadu, India

‡ Irene Getzi; Tel: +91 948 014 4908, e-mail: igetzis@gmail.com

ORCID ID: 0000-0003-0812-6938, 0000-0002-6071-1133

Research Paper Received: 16.06.2018 Revised: 25.07.2018 Accepted: 10.09.2018

Abstract- Cloud computing has produced a paradigm shift in large-scale data outsourcing and computing. As the cloud server itself cannot be trusted, it is essential to store the data in encrypted form, which however makes it unsuitable to perform searching, computation or analysis on the data. Searchable Symmetric Encryption (SSE) allows the user to perform keyword search over encrypted data without leaking information to the storage provider. Most of the existing SSE schemes have restrictions on the size and the number of index files, to facilitate efficient search. In this paper, we propose a dynamic SSE scheme that can operate on relatively larger, multiple index files, distributed across several nodes, without the need to explicitly merge them. The experiments have been carried out on the encrypted data stored in Amazon EMR cluster. The secure searchable inverted index, organized as a HashMap, is created instantly using Hadoop MapReduce framework during the search process, thus significantly eliminate the need to store keyword-document pairs on the server. The scheme allows dynamic update of existing index and document collection. The parallel execution of the pre-processing phase of the present research work reduces the processing time at the data owner. An implementation of our construction has been provided in this paper. Experimental results to validate the efficacy of our scheme are reported.

Keywords- Secure searches; searchable symmetric encryption; keyword search; multiple index; dynamic update.

1. Introduction

The cloud computing paradigm provides a sophisticated environment for storage, processing and distribution of large-scale data among different users in an efficient manner. More enterprises and users started adopting cloud-based services and there is an upsurge on "Data-as-a-Service" (DaaS) applications [1]. However, there are still many security and privacy challenges impeding the wide adoption of cloud computing in this domain. The concern here is about losing control over sensitive data while storing them on the untrusted third-party server. For example, in a healthcare cloud setting, even the existence of a record for a given patient should be kept private. i.e., the untrusted server should not even learn that a given person has a health complication or is a patient at a hospital.

Encrypting the data before outsourcing could provide confidentiality, but at the same time it makes traditional data utilization services such as

searching and retrieving the data a difficult task. To enable searching over the data and recover matching records, the user must either store an index locally, or download the entire encrypted datasets, decrypt it locally, and then search for the desired results. Since indexes can grow large, the first approach obviously negates the benefits of cloud storage, while the second has high communication complexity. Another method lets the server decrypt the data, runs the query on the server side, and sends only the results back to the user. This allows the server to learn the data being queried and hence makes encryption less useful. Hence, it is desirable to provide end-end protection to the data.

The question of performing computations on encrypted data has instigated several fascinating techniques. The problem of searching on encrypted data was first considered explicitly by Song, Wagner and Perrig [2]. The search on symmetrically encrypted data can achieve optimal security guarantee by considering the work on oblivious RAMs [4], and fully homomorphic encryption [5]. Unfortunately, these approaches require high computational overhead and thus remains impractical for large scale data outsourcing applications. The *Searchable Symmetric Encryption* (SSE) scheme strikes a good balance between security guarantees and practical performance with the smallest possible loss of data confidentiality and is best suitable for the design of searchable cryptographic cloud storage systems [6].

At a high level, a Searchable Encryption scheme employs a prebuilt encrypted search index at setup that lets users with appropriate tokens securely search over the encrypted data. Most of the existing SSE schemes mentioned in [7,8,9,11,14,15,17], are based on the notion that the searchable index is a single file stored in a centralized location. This assumption may not be suitable for many cloud-based applications that have large scale data stored in multiple locations. For example, in a healthcare domain the data may arise from different hospitals, diagnostic centers, embedded devices, sensors or mobile phones, thus makes it difficult to integrate and build a single searchable index at once. It is not reasonable to require the user to move all the data into a single location. Liu, Chu and Chen [21], proposed a SSE scheme with support for multiple data sources. The scheme requires the index files to be merged by the server before performing the search. Recent SSE schemes [23,24,25,26] leverages distributed search over multiple servers in an attempt to mimize leakage. But none of the schemes addressed the problem of dynamic datasets which require frequent updates.

Our Contributions: The present work deviates from existing inverted index-based techniques found elsewhere [27]. The proposed SSE scheme aims to build an efficient privacy preserving keyword search over highly dynamic and largescale cloud-based storage. The scheme uses a forward index to store the encrypted document-keyword pairs. This approach allows for straightforward index building as well as dynamic update operations. The parallel implementation of the search phase using Hadoop MapReduce concept enables to maintain the search complexity sublinear. An inverted index organized as HashMap is created during the search phase which allows for subsequent multiple searches by the user during a session with constant search time.

The proposed construction extends SSE over distributed data strored across multiple servers. In addition, the MapReduce framework provide mechanisms to divide and randomly distribute index to multiple servers. Hence no single server can have the possesion of the entire document or index and thus have the capability to minimize leakage. In addition, in the present work the index files and the document collection are encrypted in parallel to minimize the setup time considerably.

The rest of the paper is organized as follows: The next section describes the related work done in this area. Section 3 includes the general working model of the SSE scheme and its security requirements. The detailed description of our construction is presented in Section 4. The implementation of the work and the results are discussed in section 5 and concluded in Section 6.

2. Related Work

Over a decade, the problem of searching on encrypted data is gaining momentum and created a new research area. The state-of-the-art

constructions achieve different trade-off between security, efficiency and query expressiveness.

The searchable symmetric encryption scheme proposed by Song in [2], uses stream cipher for encryption and supports sequential search. The complexity of encryption and search algorithms are linear in time with the size of data files. Goh [3], introduced a secure index-based search technique using Bloom filters. Subsequent SSE schemes use different index structures that include linked list, array, look-up tables, and binary trees to extend the functionality and efficiency of SSE schemes. The schemes primarily used only static indexes that prevents user to dynamically add to the existing index or document collection. If update is required, the index file must be rebuilt.

The adoption of inverted index approach proposed in [7], reduced the search time to sublinear time. The notion of dynamic updates introduced by Kamara, Papamanthou, and Roeder in [9] and the work that followed [10,11,12], uses additional data structures such as search tables, arrays, tree structures to manage newly added data or deleted data thus making their construction very complex. In addition, the update operations reveal a non-trivial amount of information. The notion of *Blind Storage* introduced in [13], precludes the remote server from learning the number of files uploaded and their sizes.

Some of the recent works attempt to improve search efficiency and minimize leakage by using different approaches. The scheme described in [14], utilizes inverted matrix for efficient index construction that hides the number of keywords. An optimal index size using bloom filters and integer arrays has been achieved in [15]. However, both the schemes are not suitable for dynamic updates. A search efficient scheme using relevance score was proposed for medical cloud data [16]. To allow authorized access to multiple users, the scheme uses attribute-based encryption and secure k-Nearest Neighbor algorithm. The scheme requires that for each dynamic update, the vector with a relevance score has to be computed and send to all the authorized search users. Yang, Li, Yan, Zhang, and Cui [17], introduced an elegant search method using inner product of vectors. However, the scheme uses a different key for each document and thus makes it impractical for largescale data that requires large number of keys and trapdoors to be generated.

The ultimate challenge when it comes to SSE is to enable secure computing on massive data sets [18]. SSE schemes for large-scale data are proposed in the recent years. Cash et al. [20], proposed a scheme to work with very-large scale datasets of terabyte-scale data. The scheme required additional storage and very high computation capabilities. Liu, Chu and Chen [21], proposed the notion of Multi-Data-Source SSE, which allows the data to come from different sources. Though the scheme is efficient in terms of index size and search time, it offers less support for dynamic updates. Another work by Hirano et al. [22] proposed two approaches for searching through multiple encrypted indexes. The first approach needs to generate N trapdoors equal to the number of encrypted indexes. The second approach utilizes hash chains that allows server to generate all trapdoors from a core element given by the user. But the scheme has a security issue that it may leak the searched keyword, as the hash chain uses the public information, the unique index identifier as the salt.

Recently, distributed query processing capabilities are considered in the SSE schemes. Kuzu, Islam, and Kantarcioglu [23] proposed a scheme that vertically partitioned the data into multiple servers. The inverted index construction depends upon the frequency distribution of the keywords over the document and organized as a list or block vector. In addition, the update operation requires additional data structures that may leak information and has communication overhead. The schemes [24,25,26], attempt to minimize the leakage by distributing sets of encrypted blocks randomly to different servers so that no server has information on the entire datasets. However, the schemes have communication overhead or require the user to store additional information about the blocks of data stored in each server. In addition, the schemes do not have options for dynamic updates.

Horvath and Vajda [28] revisited the role of sequential scan in SSE and claim that their construction offers efficient search under the assumptions, that only partial database needs to be scanned for many real-world scenarios.

The problem we focus in this work is to provide secure and efficient search over very largescale, highly dynamic datasets that require frequent updates. The proposed system enables us to search securely through multiple index files distributed across several servers. The parallel execution at Setup and Search phases produces efficient results. Also, the scheme does not require storage at the user, additional data structures for updates, and it minimizes the communication overhead between the user and the server.

3. Preliminaries

This section intends to provide information on the working principle of SSE scheme and contains the notations used in the security algorithms of SSE. The SSE scheme allows the data owner to store an encrypted document collection $D = (D_1, D_2,...,D_n)$ on an honest-but-curious server *S*, while preserving the ability to search through them through tokens. The search token represents an encrypted query that can be generated only by users with the appropriate secret key. Searchable Symmetric Encryption Scheme (SSE) is a collection of five polynomial-time algorithms (KeyGen, Enc, TokenGen, Search and update) such that:

***K*←*KeyGen*(1^k)**: a probabilistic key generation algorithm that is run by the data owner to setup the scheme. It takes a security parameter $1^{k,}$ and outputs secret keys *K* for the scheme.

(I_E, D_E) ← *Enc* (*K*, *D*, *I*): run by the owner that takes as input a secret key *K*, Index *I*, and a document collection *D*. It outputs an encrypted index I_E, and a sequence of ciphertexts D_E.

T_w ← *TokenGen* (*K*, *w*): run by the owner / user to generate a token for a given word. It takes a secret key *K* and a keyword *w* as inputs and returns a token T_w.

***Id*(*w*) ← *Search* (I_E, T_w)**: run by the server *S* to search for the documents in *D* that contain word *w*. It takes an encrypted index I_E, encrypted collection D_E and a token T_w and returns *Id*(*w*), the set of identifiers of documents containing *w*.

(I'_E, D'_E)←*Update*(I_E, *upd*), D_E): run by the data owner to perform an update operation *upd:= (addDoc, Id, W) or upd:= (delDoc, Id)* where *Id* is the document identifier to be added or removed, and $W := (w_1, w_2,...w_k)$ is the list of unique keywords related to the document to be added to the index. The Update algorithm adds (or deletes) the document to (or from) D_E, and results in an updated index I'_E and updated data collection D'_E.

The *KeyGen* algorithm generates keys for the encryption phase. The user generates associated keywords index for data files, and encrypts index *I* and document collection $D = (D_1, D_2,...,D_n)$ that have unique identifiers $ID = (Id_1, Id_2,...,Id_n)$ and a set of keywords $W = (w_1, w_2,...,w_m)$. Then user uploads the encrypted documents D_E and secure index I_E to the server. The index I_E efficiently maps a keyword $w \subset W$ to a set of identifiers to that correspond to a set of files. The receiver picks a target keyword and generate token for this T_w keyword by running *TokenGen* algorithm, then sends it to the server. As the server receives the search query, it runs the *Search* algorithm with the encrypted index and token and return the documents that contains the token.

In this work, a dynamic SSE scheme, which can search securely over very large-scale cloud data, is presented. The section that follows contains the detailed description of our scheme.

4. Construction of the Present Work

In this section, we propose a dynamic SSE scheme suitable to work with distributed colossal data storage that arises from multiple sources. The scheme makes use of parallel processing capabilities to achieve better performance. At *Setup* phase, the document collection, consists of several text and image files are encrypted in parallel and send to the cloud server along with the search index, which contains the essential keywords required to search through the documents. The documents and index files can be uploaded to distributed locations. During the *search* phase, the scheme uses MapReduce concept to enable efficient keyword search through the distributed collection. The Fig.1, shows the schematic diagram of the present work.

We assume that each document D_i is associated with a unique document identifier Id_i, generated by the owner. The metadata or keywords W_i

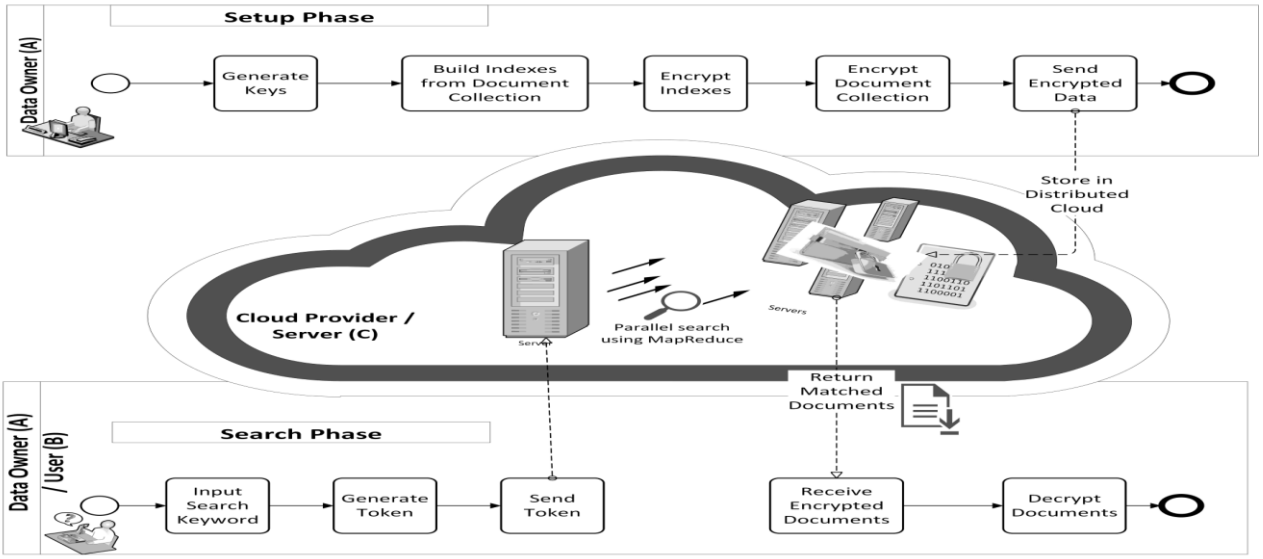


Fig. 1. Schematic diagram of proposed scheme (The actors are Data owner (A), Data user (B) and Cloud provider (C))

associated with the documents are used to build the index file *I*. The main idea of the SSE scheme

is to transform each unique keyword *w* to a searchable representation C_w, in a way that the user can search for documents in which the keyword occurs $\{W_i \mid w \in W_i\}$ by issuing a token T_w.

Our basic scheme comprises of the setup, search and update phases and the associated algorithms as described below.

4.1 Setup Phase

During the setup phase protocol, the data owner encrypts the data files and builds searchable indexes. It makes use of two algorithms namely:

- ***Key Generation:*** The key generation algorithm (Alg. 4.1.1) uses a random λ-bit string of length 128 bits and generates two secret keys, K_m, K_w (Keysize 128 bits) for encrypting data files and index. In our scheme, the key was generated randomly by using Java Crypto API and stored in a text file for further use by the data owner. The key can be shared using any secure key distribution protocols to the user with whom the owner wishes to share the data.
- ***Pre-processing / Index Building:*** The data owner uses BuildIndex algorithm (Alg. 4.1.2) to build a local index for his datasets. For example, in healthcare system, the patient's medical images, reports, and related healthcare data from diagnostic centers or smart devices or sensors constitute the dataset/s. The metadata that includes information such as patientID, date of investigation, along with the document identifier can constitute the index file. In our scheme, we use multiple index files of varying sizes to test the efficacy of the scheme. The index files are organized as a collection of CSV files. The index files as well as the document collections are encrypted using Advanced Encryption Standard (AES) algorithm in CBC mode (Cipher Chaining Block Mode) and are uploaded to the server. To speed up the execution time, the input data is split into multiple chunks and are encrypted in parallel using Java Executor Service.

Algorithm 4.1.1. KeyGen

Input: λ: A security parameter

Output: K_m, K_w**:** Secret keys for encrypting documents and index $K=(K_m, K_w) \leftarrow \{0, 1\}^{\lambda}$ (Randomly generates K_m, K_w from $\{0, 1\}^{\lambda}$)

Fig. 2, provides the work-flow of the setup phase that computes the encryption of document collection and index.

Algorithm 4.1.2. BuildIndex

Input: Secret Keys $K= (K_m, K_w)$, Document Collections: $D=D_1, \ldots, D_n$,

List of document Identifiers Id_i, and their metadata stored as CSV files I

Output: Encrypted document collection D_E and secure Index files I_E

a) Initialization:

Parse D and build W, the set of distinct words w that describes the content in D. For each document D_i, add a tuple $(Id_i, \{w, w \in W\})$ in the index file I, organized as a CSV file.

b) Encryption:

For each entry in D, do
$D_E \leftarrow E(K_m, D)$
For each entry in I, do
split I into n
for each child I_i of I, do
read $(Id_i, \{w, w \in W\})$
$I_{iE}(w) \leftarrow E(K_w, I(w))$
merge I_{iE} to I_E
return D_E, I_E
Upload D_E and I_E to the cloud server.

4.2 Search Phase

Whenever the user wants to retrieve the set of encrypted data items from the server, it generates search tokens using algorithm **TokenGen**. Upon receiving an encrypted search token T_w from the user, the storage provider uses the **searchIndex** algorithm to search through the encrypted index and returns the set of identifiers of documents that contains the search keyword w.

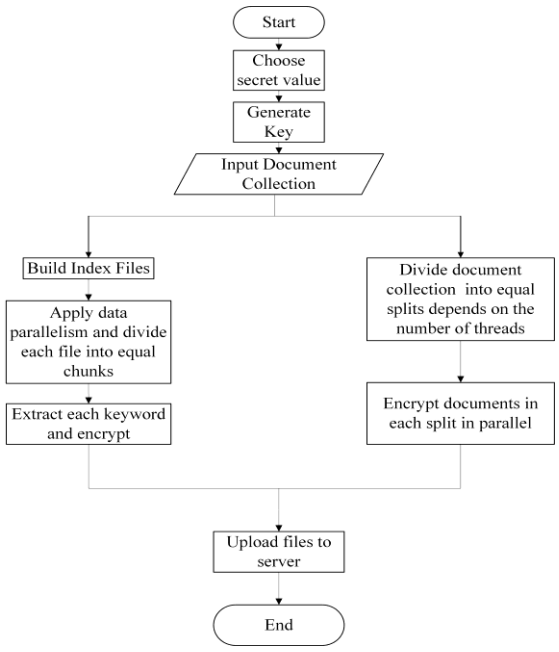


Fig.2. Diagrammatic representation of Set-up Phase

- ***TokenGen:*** The algorithm (Alg. 4.2.1) is run by the data owner /user to generate a search token for a given word. It takes a secret key K_w and a keyword w as inputs, and returns a token T_w. The token is used to locate the correct entry in the index file.
- ***searchIndex:*** Our scheme (Alg. 4.2.2) uses multiple index files that can be stored in different nodes or clusters. To cope up with large scale data, the MapReduce computing framework is used as an efficient tool to search through the encrypted data. The multiple encrypted index files are fed to different mappers. At each mapper, the files to be processed are divided into equal splits and are assigned to a map task. The map task scans the CSV file, and if the token is found, sends the corresponding document identifier to the output list. The reducer combines the results generated by the mappers and fetches the corresponding encrypted documents from HDFS (Hadoop Distributed File System) and sent it to the user.

Algorithm 4.2.2. searchIndex

Input: Search Token T_w, Encrypted Index files I_E, Encrypted Document collection D_E
Output: *resId*, the List of document identifiers Id_i that contains the search keywords *w*
The MapReduce Framework of Hadoop is used to search through the encrypted indexes.
Map Task: Map (I_E, T_w)
for each tuple $t \in I_E$ do
Id← *t[0];*
for each word C_w in tuple *t* do
if T_w == C_w then
return (C_w, Id)

Reduce Task: Reduce (C_w, *[Id_1, Id_2,…]* pairs from map task*)*
resId = []
for each C_w in map results
resId ← resId.append(*I*
for each *Id* in res*Id*:
Fetchdocuments(*Id*);
sendToUser(d_E);
done

The detailed process flow of the search phase is depicted in Fig. 3.

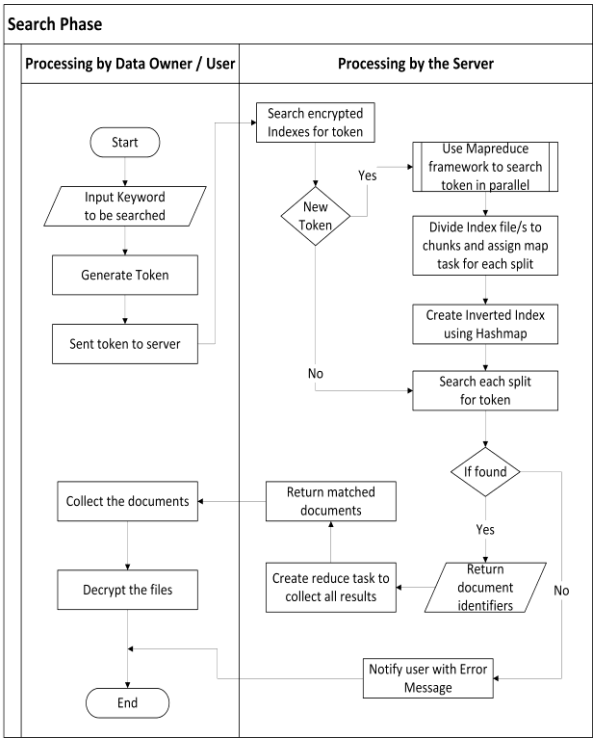


Fig.3. Diagrammatic representation of Search Phase

4.3 Update Phase

Our construction allows dynamic update of document collection as well as Index file as described in (Alg. 4.3.1). When a new file is uploaded to a document collection, the metadata corresponding to document is appended to the existing index file. The scheme is easily scalable to support adding new document collection as well. The forward indexing allows the scheme to perform dynamic updates without additional overhead.

The ***addDoc*** algorithm, allows straightforward addition of a new document to the existing collection. The document is encrypted using the key *K* and added to the document collection D_E. The keywords *W* are encrypted and appended to I_E.

Algorithm 4.3.1. *Update*

Input: Document D_{Id}, Encrypted Index files I_E, Keys K, a set of keywords *W*, Encrypted Document collection D_E.
Output: results in an updated index I'_E and an Document collection D'_E.

***(I'_E , D'_E)←Update(I_E , upd), D_E)*:** run by the data owner to perform an update operation *upd:= (addDoc, Id, W) or upd:= (delDoc, Id)* as described below:

a) *addDoc(D_{Id}, Id, W)*
$D_{E\,(Id)} \leftarrow E(K_m, D_{Id})$ // Encrypt document
tuple *t*← *Id*
For each keyword *w* in *W,*
$C_w \leftarrow E(K_w, w)$
t← *t.append*(C_w)
$D'_E \leftarrow D_E + D_{E\,(Id)}$ // Upload $D_{E\,(Id)}$ to D_E
$I'_E \leftarrow I_E + t$ // Append I_E with *t*

b) del*Doc(D_{Id}, Id)*
Send *Id* to Document Collection
$D'_E \leftarrow D_E - D_{E\,(Id)}$
Search I_E for *Id* using MapReduce
If Found
$I'_E \leftarrow I_E - I(Id)$ // *Delete tuple*

5. Implementation and Results

The proposed scheme has been successfully implemented using Java as a tool. The details of

the implementation and the results are discussed in the following section.

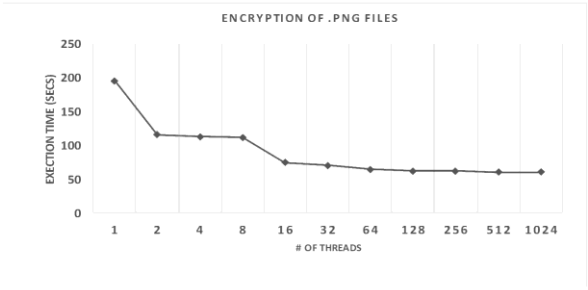
- ***Our Prototype Summary:*** Our model includes processing at three levels: pre-processing / setup at owner, at the user and the server during search. The setup phase generates the encrypted documents and index files. The user generates a token to search through encrypted documents and decrypts server responses. The server uses the encrypted index to answer user requests and provide mechanism for updating the index and document collection.
- ***Experimental Platform:*** The experiments at setup phase has been implemented in Java and run on DELL system, equipped with Intel Core i5-6402P @ 2.80 GHz x 4 computer with 8GB RAM running Ubuntu 14.04. Each data point in Fig. 4 and Fig. 5 is an average of five executions. The server-side experiments were performed using Hadoop MapReduce framework on Amazon EMR cluster on a r4.2xlarge instance running Linux containing eight High Frequency Intel Xeon E5-2686 v4 (Broadwell) Processors and 61GB of RAM with EBS volume of 32MB.
- ***Dataset:*** We chose two types of real-world datasets, a collection of 1000 text documents of 25 MB and a set of 1000 png images of total size 100 MB. To check the performance of the present scheme, different sizes of index files were used as shown in Table 1. The index files are CSV files that contain metadata that forms the set of keywords that can be used for searching through the document collection. The first field of the index file contains the document identifier. Each row contains the set of keywords that describes the corresponding document.

5.1 Setup Phase

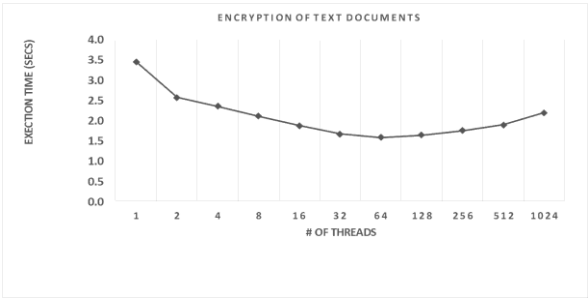
The document collections are encrypted using Advanced Encryption Standard (AES) algorithm in Cipher Block Chaining mode. The key is generated randomly and stored in a secure location at the data owner for later use to decrypt the files. To speed up the process, the files in the document collection are encrypted in parallel using Java Executor Service. The experiments were performed in varying the number of threads from 2, 4, 8, 16, … , 1024 in powers of 2. From the Fig.4(a) and Fig.4(b), it can be seen that the execution time considerably decreases with the parallel execution.

Table 1. Encrypted Index Files: document identifier (Column A), size of the index file (Column B, number of keywords words in the document Column C)

Document Id (A)	File Size (in GB) (B)	No. of keywords (C)
sseIndx0A	0.02	741,870
sseIndx1A	0.04	1,285,499
sseIndx2A	0.06	2,628,837
sseIndx3A	0.125	4,777,216
sseIndx4A	0.25	7,875,647
sseIndx5A	0.5	21,925,296
sseIndx6A	1.00	40,905,048
sseIndx7A	2.50	93,722,000
sseIndx8A	4.00	507,444,000



(a)



(b)

Fig. 4. Parallel Execution of Document Collection - Varying # of Threads (Execution time for .png files (a), text collection (b))

The data parallelism produces better results with increase in size of the document collection; the execution time of parallel encryption of image collection is reduced to 1/3 of the non-parallel implementation. This approach is suitable for massive datasets.

Our scheme uses large-scale index files as it requires by many real-time applications. A search index file, organized as a CSV file, is divided into equal chunks of lines depends on the number of threads and executed in parallel. For each line, the individual keywords are extracted, encrypted using AES algorithm. The results are collected, and the encrypted index file is uploaded to the server. An index file of size 123 MB with 1,048,578 lines and 16,777,216 keywords is selected for testing and the results are depicted in Fig. 5. The parallel execution reduces the execution time by 50%.

5.2 Search Phase

The search phase consists of two algorithms. The scheme allows the user to retrieve the documents through the keywords. The secure tokens are generated by the user for the given keyword, and sent to the remote server. To realize the distributed computing environment, the server application is implemented using Hadoop MapReduce framework. The index files are distributed to different nodes using HDFS. The index files are searched in parallel and the encrypted documents linked to the token are returned to the user.

The server-side experiments are conducted at amazon cloud by varying the number of nodes. To perform empirical analysis, we use multiple index

Fig. 5. Parallel Execution of Index Varying # of Threads

files of varying sizes ranging from 0.25 GB to 4 GB. The experiments are repeated for clusters with 1 master node and varying the number of core nodes to 1, 2, 4 and 8. In contrast to the existing schemes, our scheme uses very large index files that consist of keywords ranging from few thousands to 50 million and still produces efficient results.

Initially, with single and two nodes clusters, the search time is linear with the index file size. There is a significant reduction in the search time with increase in the number of nodes as shown in Fig. 6. As the number of nodes increases, it produces consistent search time irrespective of different index file sizes. Since the MapReduce framework can handle split size of 128 MB at a time, the scheme produces efficient results with just four nodes. Thus, the scheme is suitable for handling very large-scale datasets.

Our scheme allows searching through multiple index files distributed across multiple nodes. The experiments are repeated by varying the number of index files at each node. As shown in Fig. 7 and Fig. 8, the usage of multiple files does not affect

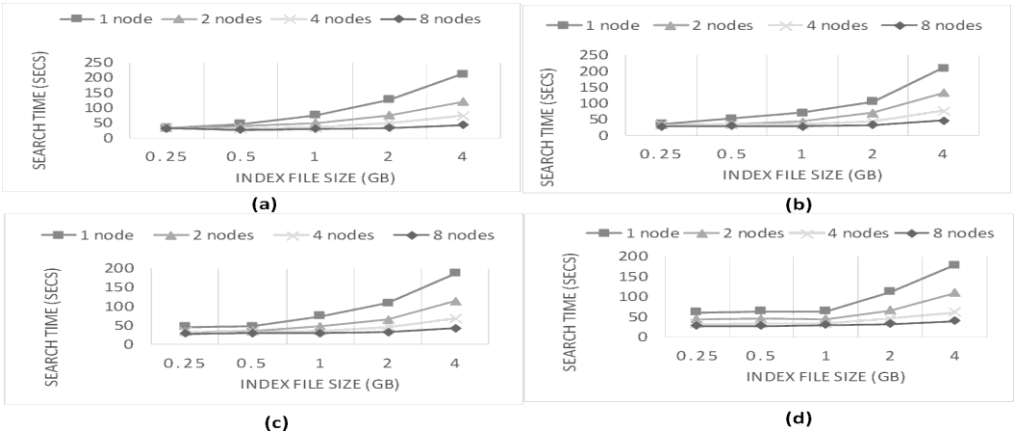


Fig. 6. Search time Evaluation: Varying # of Index files – Single Index (a), 4 Files (b), 8 Files (c) and 16 Files (d)

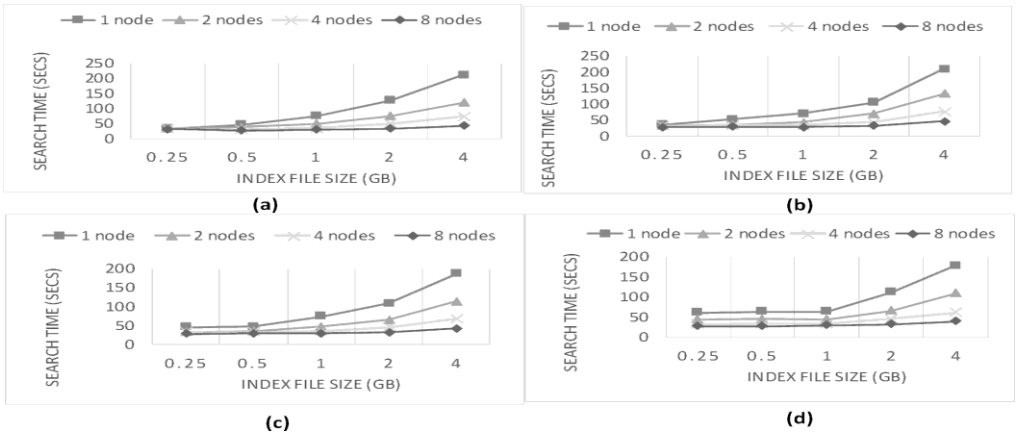


Fig. 7. Search time Evaluation: Varying # File Size and # of index files – 0.5 GB (a), 1 GB (b), 2 GB (c) and 4 GB (d)

the performance. Strikingly, as the number of index files increases, the scheme produces better results for larger file sizes. The search time is consistent for files with size greater than 0.5 GB irrespective of number of index files. Thus, the scheme is suitable for handling multiple index files located at distributed locations.

The scheme allows multiple searches during a session. The inverted index created at runtime during the search phase allows subsequent searches would be performed with constant search time.

6. Performance Evaluation

In this section, we evaluate the performance of our present work compared with the other SSE schemes proposed for large-scale cloud strorage. The schemes are evaluated respectively, in terms of index structure, index size, search time, update complexity, leakage, and query processing and the results are tabulated in Table 2.

The present work deviates from the existing methods in the following aspects; the ability to work with larger and distributed indexes through MapReduce framework, and creation of inverted index at search phase to facilitate multiple keyword search. As mentioned in section 5, the parallel implementation of the Setup and Search phases offer significant reduction in the execution time.

In most of the real-world cloud-based applications, the data collection has seen rapid expansion with frequent updates [28]. Inverted index based approaches may not be practical for such applications. Our scheme based on forward indexing offers straightforward update of dynamic storage. The linear search complexity inherent to forward indexing, has been addressed. The distributed processing of search algorithm with MapReduce framework offers sub-linear search.

Unlike the other schemes [21,23,24,28], the search operation could be done in a few seconds even for the index with more than 500,000,000 keywords and thus eliminates the need to have smaller indexes as described in [15,21,24]. An inverted index, created at runtime of search phase,

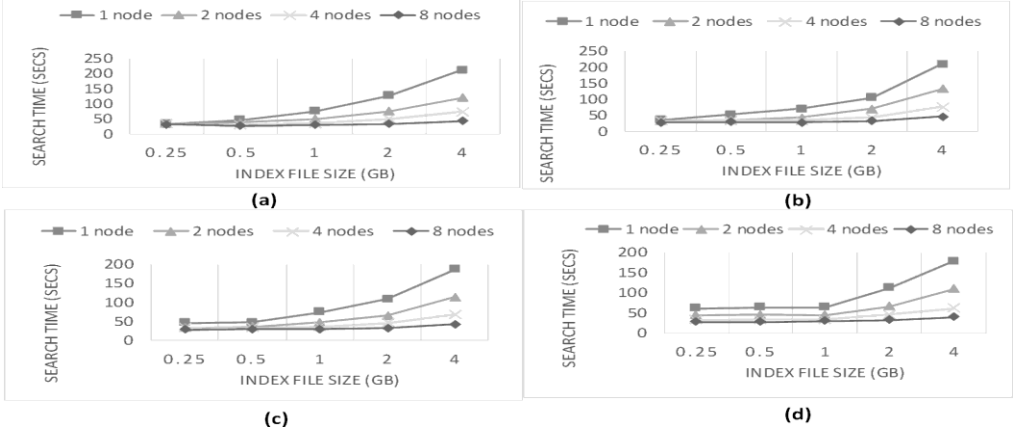


Fig. 8. Search time Evaluation: Varying # File Size and # of nodes – 0.5 GB (a), 1 GB (b), 2 GB (c) and 4 GB (d)

Table 2. Comparisons with a few SSE schemes

Scheme	Index Size	Search Time	Update Complexity	Index Structure	Leakage	Distributed processing
[14]	$O(W *2^d)$	O(n)	Static SSE	Inverted Matrix	N, m, r	No
[15]	O(mn)	O(log m+r)	Static SSE	Inverted & Bloom filters	N, m	No
[20]	O(N)	O(r/p)	O(N)	Inverted	N, n	No
[21]	O(n)	O(n)	O(N)	Inverted	N, n	No
[23]	O(N)	O(r)	Static SSE	Inverted Tree	N, n	Yes
[24]	O(Wn) + O(N)/s	O(n)/s	O(m)	Inverted	N/s, n/s, r/s	Yes
[26]	O((N + n)l/s)	avg(r/s)	O((N + n)l/s)	Inverted	W + n, ((N + n)l)/s	Yes
[28]	O(mn)	O(nm)	O(m)	Forward	N, n, m	No
Ours	O((mn)l/s)	O(mn/c)/s O(r)/s *	O(m)	Forward & Inverted*	N/s, n/s, m/s, r/s	Yes

n is the number of documents; m is the number of keywords per document;
N is the number of document-keyword pairs; p denotes the number of processors
r is the number of documents matching the query keyword; s denotes the number of servers
l represents the number of blocks ; c denotes the number of splits / map tasks
W is the totalnumber of keywords; d is the atmost length of the keyword
** for subsequent keyword search*

allows the user to perform multiple searches during a session. The HashMap structure of the index, further reduces the search complexity to O(r/s) for subsequent query processing. Unlike [20,21,24,26], updating the data collection and index, with a new entry is straightforward in our approach and do not require any additional overhead or data structures.

6.1 Security Analysis

In this present construction, the cloud servers are considered as honest-but-curious Adversaries. The scheme uses block cipher AES, which has the underlying PRP, PRF defined as $F : \{0, 1\}^{\lambda} \times \{0, 1\}^{*} \rightarrow \{0, 1\}^{n}$, for $\lambda \in N$, which is computationally indistinguishable from a random function when observed by any probabilistic polynomial-time adversary $A = (A_1, A_2..., A_k)$. To analyse the proposed scheme, the well known model described in [4] is used.

The SSE security model is parametrized by the leakage functions $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_3$ indicating the information leakage during Setup, Search and Update protocols. We assume that each adversary have control over only a single server except the master node. The leakage functions for a single server design are described as follows:

$\mathcal{L}_1(I_E, D_E)$: Given an encrypted Index collection I_E and document collection D_E, L_1 outputs the size of each document $|D_i|$, the identifiers of each document Id_i, length of the index |I| and the keyword distributions for a single server.

$\mathcal{L}_2(I_E, T_w, D_E, t)$: Given a Token T_w, at time t, L_2 outputs the document identifiers Id_i containing keyword w, the number of matched documents (i.e the access pattern), and if the query is repeated, the

set of tokens used by previous searches. (i.e the search pattern).

L_3 (I_E , T_w , W, Id) : Given a document *Id*, and related keywords *W*, L_3 outputs the the number of keywords added or removed, the document identifier, updated index size.

Let $\lambda \in N$ be the security parameter and *E*, the secure encryption scheme. For an adversary *A* and a simulator *S*, we conduct two probabilistic games $Real_A(\lambda)$ and $Sim_{A,S}(\lambda)$, of which, $Real_A(\lambda)$ conducted by adversary *A* and challenger *C* and while $Sim_{A,S}(\lambda)$, simulated by *A and S* using only the available information that an adversary can get from leakage functions.

$Real_A(\lambda)$: The challenger *C* runs $KeyGen(1^\lambda)$ to generate a random secret key K for *E*. *A* sends *D* to *C*. Then *C* returns $I_E \leftarrow BuildIndex(K, D)$ and $D_E \leftarrow E(K, D)$ to *E*. Then *A* submits q polynomial queries, $Q = (w_1, w_2,...,w_q)$. For each keyword, *A* receives a Token $T_w \leftarrow TokenGen(K, w)$ from *C* and picks the next keyword as a function of previously obtained tokens and search outcomes. Finally, *A* outputs a bit $b \in \{0, 1\}$.

$Sim_{A,S}(\lambda)$: *A* chooses a set of documents D and sends it to *S*. Given $L_1(I_E , D_E)$, *S* computes (I^*_E , D^*_E) and sends them to *A* . In the search phase, *A* makes q queries expressed as Q. For each keyword, *S* learns from $L_2(I_E , T_w, D_E, t)$ and returns T_w to *A*. Finally, *A* outputs a bit $b' \in \{0, 1\}$.

We say that our scheme satisfies adaptive semantic security if for all probabilistic polynomial-time (PPT) adversaries *A*, there exists a PPT simulator S such that: $|Pr[Real_A(\lambda) = 1]| - |Pr[Sim_{A,S}(\lambda) = 1]| \leq negl(\lambda)$, where $negl(\lambda)$ is a negligible function.

The SSE security model can be extended to multi-server setting. If an adversary *A* who controls all servers $S_j \in S$, then *A* has access to the whole of (D_{sj}, I_{Sj}) $S_j \in S$. and can form a single index table I from all I_{sj}. Similarly, *A* can consolidate all the previous query responses to perform statistical attacks. In this case, *A* has the same information as an adversary of the single server SSE scheme, and hence gain equal advantage.

We assume that our servers are non-colluding and hence each adversary *A* has control over only partial documents and partial indexes stored on a single server except for the one that can attack the master node. For a single server SSE, the scheme can be proved semantically secure against *A*. The simulator *S* can adaptively generates (I'_E , T'_w , D'_E) as follows.

➢ As the size of each D_E is known to *S* through L_1, it can simulate the encrypted documents $D'_j \leftarrow \{0, 1\}^{|Dj|}$ *for (j=1,...,n). Let* $D' = (D'_1, ...,D'_n)$. *S* creates I'_E and randomly chooses $S_i\{0, 1\}^n$ and stores it in each entry of I'_E. Then *S* sends (D'_E , I'_E) to *A* .

➢ For q queries, *S* checks whether it has seen w_i earlier, from the search pattern revealed by L_2. If yes, then *S* retrieves the token previously generated for w_i and uses it as T'_{wi}. However, if w_i has appeared for the first time, then *S* generates a token T'_{wi} returns T'_{wi} to *A*.

Since the underlying symmetric functions PRF, PRP, *E* are secure cryptographic primitives, our scheme will guarantee that each encrypted document D_i and a real ciphertext D'_i is indistinguishable. Similar discussion can be applied for index construction and token generation, and thus makes *A* unable to distinguish between I'_E and real index I_E, T_{wi} in $Real_A(\lambda)$ and T'_{wi} in $Sim_{A,S}(\lambda)$ without the knowledge of the key K.

A scheme offers better security, if it can control the leak. Our scheme can minimize the access pattern leakage without using complex structures as required by [14,15,16,24,26]. Since the documents and indexes are partitioned horizontally and sent to multiple servers S, it is now hard for an Adversary which controls only a subset of S, to know the actual size of the documents, indexes, and the distribution of keyword-document pairs. Thus, our scheme controls L_1.

Multiserver block-based SSE design provides higher security for SSE [25]. The MapReduce framework used in the search phase, split the indexes into blocks and randomly assign them to multiple tasks which makes it even more harder for the adversary to know the keyword and document distribution as in [25, 26].

The update operation is straightforward in our scheme, which can leak only the number of

keywords added, in contrary to the other implementations [16,21,26]. The schemes may leak the documents that have the same keywords. Since the data and indexes are distributed, the other servers may not learn when and where the update had happened. In addition, the frequent updates to the storage and index, offer inconsistent search results over time, and thus prevents the adversary from learning from the previous search patterns. Thus, the present constuction controls L_2, and L_3. Moreover, the creation of inverted index during search phase and its construction using HashMap provides location transparency and prevents the leakage of keyword-document pair distribution. Thus, the present work offers efficient, secure and practical solution and is more relevant for highly dynamic large-scale cloud storage.

7. Conclusion

In this work, we have implemented a Dynamic Searchable Symmetric Encryption scheme that can perform secure searches on the encrypted data stored on the real cloud architecture. As the data generated in the cloud-based applications are usually very large and is often uploaded to distributed locations, we have designed the scheme that is suitable for handling multiple index files of larger sizes. The data distributed across multiple locations are handled in parallel without compromising on the security. The security of the scheme can be further improved by using a probabilistic algorithm. To speed up the pre-processing phase, the document collection and index files have been executed in parallel. The scheme supports dynamic update of the encrypted collection with minimal updates and communication overhead. The search phase has been implemented at Amazon Web Services using Hadoop MapReduce concept. The scheme performs efficient search through very large index files with millions of keywords. The parallel execution of distributed files at the server, produces minimal search time even for larger file sizes. Remarkably, the search time remains consistent with increase in the number of files. Thus, the scheme is suitable for any real-time application that involves large-scale datasets stored across multiple clouds.

References

[1]. Z. Zheng, J. Zhu, M. R. Lyu, "Service-generated Big data and Big data-as-a-service: An overview", In Proceedings of IEEE International Congress on Big Data, pp. 403-410, 27 June -2 July 2013. DOI=10.1109/BigData.Congress.2013.60.

[2]. D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searching on encrypted data. In proceedings of IEEE Symposium on Security and Privacy", SP'00, pages 44–55, 14 -17 May 2000. DOI= 10.1109/SECPRI.2000.848445

[3]. E. J. Goh, "Secure Indexes", Cryptology ePrint Archive, Report2003/216, 2003. http://eprint.iacr.org/2003/216.pdf

[4]. O. Goldreich and R. Ostrovsky. "Software Protection and Simulation on Oblivious RAMs", Journal of the ACM (JACM), Vol. 43, No. 3, pp. 431–473, 1996. DOI=10.1145/233551.233553

[5]. C. Gentry, "Fully homomorphic encryption using ideal lattices", In Proceedings of the ACM symposium on Theory of computing (STOC '09), pp.169-178, ACM, 31 May – 02 June, 2009. DOI=10.1145/1536414.1536440

[6]. S. Kamara, K. Lauter, "Cryptographic cloud storage", In Financial Cryptography and Data Security, FC2010", Vol. 6054 of LNCS, Springer-Verlag, pp. 136–149, 2010. DOI = 10.1007/978-3-642-14992-4_13

[7]. R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions", In Proceedings of the 2006 ACM Conference on Computer and Communications Security, CCS'06, pp. 79–88, 30 October – 03 November, 2006. https://eprint.iacr.org/2006/210.pdf

[8]. C. Bösch, P. Hartel, W. Jonker, A. Peter, "A survey of provably secure searchable encryption", ACM Computing Survey (CSUR), Vol. 47, No. 2, pp. 1–18, 2015. DOI=10.1145/2636328

[9]. S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption", In Proceedings of the ACM Conference on Computer and Communications Security, CCS'12, pp.965–976, 16 – 18 October, 2012. DOI=10.1145/2382196.2382298

[10]. S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption", In Financial Cryptography and Data Security, Vol. 7859 of LNCS, pp. 258–274, 2013. www.ifca.ai/pub/fc13/78590253.pdf

[11]. E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage", In proceedings of Network and Distributed System Security Symposium,NDSS'14,2014. https://eprint.iacr.org/2013/832.pdf

[12]. F. Hahn, F. Kerschbaum, "Searchable Encryption with Secure and Efficient Updates", In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, CCS'14, pp. 310-320, 03 – 07 November, 2014. DOI=10.1145/2660267.2660297

[13]. M. Naveed, M. Prabhakaran, and C. Gunter, "Dynamic searchable encryption via blind storage", In Proceedings of the IEEE Symposium on Security and Privacy, S&P, 18 – 21 May 2014. DOI=10.1109/SP.2014.47

[14]. X. Jiang, X. Ge, J. Yu, F. Kong, X. Cheng and R. Hao, "An Efficient Symmetric Searchable Encryption Scheme for Cloud Storage", Journal of Internet Services and Information Security (JISIS), Vol. 7, No. 2, pp. 1-18, May 2017. https://pdfs.semanticscholar.org/2a0e/df98524c92931478cd3fbc32afb6caa7a57d.pdf

[15]. R. Miyoshi, H. Yamamoto, H. Fujiwara, and T. Miyazaki, "Practical and Secure Searchable Symmetric Encryption with a Small Index", In Secure IT Systems, NordSec 2017, Vol. 10674 of LNCS, Springer, pp. 53-69, November 2017. DOI=10.1007/978-3-319-70290-2_4

[16]. H. Li, Y. Yang, Y. Dai, S. Yu, and Y. Xiang, "Achieving Secure and Efficient Dynamic Searchable Symmetric Encryption over Medical Cloud Data", IEEE Transactions on Cloud Computing, November 2017. DOI = 10.1109/TCC.2017.2769645

[17]. J. Yang, S. Li, X. Yan, B. Zhang, and B. Cui "Searchable Symmetric Encryption Based on the Inner Product for Cloud Storage", International Journal of Web and Grid Services, Vol. 14, No. 1, pp.70–87, January 2018. DOI=https://doi.org/10.1504/IJWGS.2018.088393

[18]. N. P. Smart, Future Directions in Computing on Encrypted Data, Technical Report,ECRYPT-CSA, http://www.ecrypt.eu.org/csa/documents/D2.2ComputingonEncryptedData.pdf, November 2015.

[19]. M. I. Salam, W. Yau, J. Chin, S. Heng, H. Ling, R. Phan, G. S. Poh, S. Tan and W. Yap, "Implementation of searchable symmetric encryption for privacy-preserving keyword search on cloud storage", Journal of Human-centric Computing and Information Sciences, Vol. 5, No. 19, 2015. DOI = 10.1186/s13673-015-0039-9

[20]. D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Dynamic searchable encryption in very large databases: Data structures and implementation", In Proceedings of Network and Distributed System Security Symposium NDSS'14. 2014. https://eprint.iacr.org/2014/853.pdf

[21]. C. Liu, L. Zhu, and J. Chen, "Efficient Searchable Symmetric Encryption for Storing Multiple Source Data on Cloud", In Proceedings of the IEEE Conference Trustcom/BigDataSE/ISPA, 20 – 22 August 2015. DOI=10.1109/Trustcom.2015.406

[22]. T. Hirano, M. Hattori,Y. Kawai, N. Matsuda, M. Iwamoto, K. Ohta, Y. Sakai, T. Munaka, "Simple, secure, and efficient searchable symmetric encryption with multiple encrypted indexes", Advances in Information and Computer Security, IWSEC 2016, Vol. 9836 of LNCS, pp. 91–110. Springer, Cham, 12-14, September 2016. DOI = 10.1007/978-3-319-44524-3_6

[23]. M. Kuzu, M.S. Islam, and M. Kantarcioglu, ", In proceedings of the 5th ACM Conference on Data and Application Security and Privacy, CODASPY '15, pp. 271-278,02–04 March 2015. DOI=10.1145/2699026.2699116

[24]. Y. Ishai, E. Kushilevitz, S. Lu, and R. Ostrovsky, "Private Large-Scale Databases with Distributed Searchable Symmetric Encryption", Topics in Cryptology - CT-RSA 2016, Vol. 9610 of LNCS, Springer, pp. 90-107, February 2016. DOI = 10.1007/978-3-319-29485-8_6

[25]. M. Mohamad, J. Chin, and G. Poh, "On the Security Advantages of Block-Based Multiserver Searchable Symmetric Encryption" , In the proceedings of IEEE 14th Annual Conference on Privacy, Security and Trust (PST), pp. 349 – 352, 12-14 December 2016. DOI = 10.1109/PST.2016.7906985

[26]. G.S. Poh, M.S. Mohamad, & J.J Chin, "Searchable symmetric encryption over multiple servers", Cryptography and Communications, Volume 10, Issue 1, pp.139–158,January2018. https://doi.org/10.1007/s12095-017-0232-y

[27]. F. Han, J. Qin, J. Hu, "Secure searches in the cloud: A survey", Future Generation Computer Systems, Vol. 62, pp.66-75, September 2016. https://doi.org/10.1016/j.future.2016.01.007

[28]. M. Horvath and I. Vajda, "Searchable Symmetric Encryption for Restricted Search", Journal of Communications Software and Systems, Vol.14, No. 1, pp.104-111, March 2018. DOI = https://doi.org/10.24138/jcomss.v14i1.419