Journal of Ambient Intelligence and Humanized Computing
https://doi.org/10.1007/s12652-020-02621-8

ORIGINAL RESEARCH

Multi-server searchable data crypt: searchable data encryption scheme for secure distributed cloud storage

Toka Shahien[1,2] · **Amany M. Sarhan**[1] · **Mahmoud A. M. Alshewimy**[1]

Received: 21 April 2020 / Accepted: 15 October 2020

Abstract
This paper introduces a multi-server searchable symmetric encryption (SSE) scheme called the Multi-Server Searchable Data Crypt "MS-SDC" that works on achieving a trade-off between efficiency/functionality and security. The proposed scheme has the merits of dividing the uploaded file in an encrypted form into blocks and distributing them across several storage providers, which is more acceptable than uploading the whole file directly to a single server where each server only holds a subset of file/block, to ensure more security for the file. Besides that, it extracts keywords for each uploaded file to be used later by the search engine giving the user the ability to browse across his own files. This means that the user has the ability to query/search for his encrypted files on the server-side without decrypting them. Furthermore, there are various features proposed different from those presented by previous works as the scheme is developed as a multithreaded-application to speed up the uploading time, and a unique master key is generated randomly for each uploaded file unlike the previous techniques where a single master key is created randomly for all the uploaded documents leading to easily hacking the system with master key leakage. Finally, the MS-SDC system is distinctive in its smooth usage and its robustness where it can run on any browser and can be applied to any file type. The experimental results demonstrate the effectiveness of our proposed system in comparison to previous works in terms of uploading and searching time, in addition to providing many new features, applying many layers of security, and keeping high-speed performance in an efficient manner. The proposed system has reduced the file upload time up to 64% of the current research upload time via multithreading implementation of the block distribution function.

Keywords Searchable encryption · Searchable symmetric encryption · Multithread-application · And encrypted data

1 Introduction

High technology development in hardware and software is producing greater growth of data which is inevitable for almost all organizations. So, many organizations prefer to outsource their data storage to third-party storage providers, to reduce the increasing storage costs, and access to their data easily anywhere and anytime. But, this allows the providers to learn the content of the files which is the main problem; these providers can be untrusted and insecure. There are a lot of organizations and companies that promise a storage service reliability and performance, but they can not take risks with their client sensitive data (Bösch et al. 2014; Poh et al. 2017).

Traditional access controls, which are used to provide confidentiality, are mostly designed for in-house services and depend greatly on the system itself to enforce authentication. An intuitive solution would be for the user to make all documents encrypted before storing on the cloud. However, the user will lack the search ability on these uploaded documents. The next solution for the user becomes either to give the cloud provider the encryption key or download all documents and locally decrypted them. The previous approach lacks privacy and efficiency (Bösch et al. 2014; Poh et al. 2017).

These issues have motivated much research on advanced searchable encryption schemes that enable searching on

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s12652-020-02621-8) contains supplementary material, which is available to authorized users.

✉ Toka Shahien
toka.shahien@f-eng.tanta.edu.eg

[1] Computer and Control Department, Faculty of Engineering, Tanta University, Tanta, Egypt

[2] Higher Institute of Engineering and Technology, Kafr El-Sheikh, Egypt

Published online: 05 November 2020

the encrypted data while protecting the confidentiality of data and queries. Searchable symmetric encryption (SSE) was introduced as a practical solution to this issue (Bösch et al. 2014; Kim et al. 2018; Poh et al. 2017, 2018; Wang et al. 2016). An SSE scheme encrypts documents in such a way that the output contains both the encrypted data and their associated metadata, which is stored together on the server. Through querying the metadata, the scheme returns references to the matching encrypted data, this is so-called (token or trapdoor) is generated and sent to the server, to allow the server to query this data. With the help of the trapdoor, the server is able to perform a search on the still encrypted data, which can then be retrieved and decrypted locally.

Existing (dynamic) SSE schemes mainly concentrate on settings for single-user single server (Cash et al. 2013; Chang and Mitzenmacher 2005; Chase and Kamara 2010) multi-user single server (Curtmola et al. 2011; Faber et al. 2015), and two servers (Ishai et al. 2016; Peter et al. 2014) where one of the servers serves as a query proxy. So, it becomes perfect to propose a scheme involving more than two providers where every provider stores only a portion of the files and the associated metadata. Such a scheme is of interest since many storage providers are nowadays available, i.e. Google Drive, Dropbox, and Microsoft OneDrive.

There are some schemes (Poh et al. 2017, 2018) that provide storage on two or more servers, but with some limitations such as there is no query search provided for the user and the number of servers is fixed. Other schemes (Hahn and Kerschbaum 2014; Naveed et al. 2014) supports file upload and download but they are implemented in bounded environments at which requiring a cloud server providing no client computing capacity. Schemes in (Hoang et al. 2018, 2019; Stefanov et al. 2013) have completely overcame the storage/search problem on untrusted servers and hid any data access on the server-side, this is to reach a high level of security but in exchange for that still have low efficiency in terms of depending on RAM protocol usage between the server and the client.

Recently proposed schemes were single-server schemes; only a few have targeted the problem of partitioning the encrypted files into several storage areas. Furthermore, these studies have many weakness points that open a wide door for the work in this field. For example, the number of servers was fixed, the block size was fixed, a single key was used for all the files, there is no query search and they worked only on the text files. According to this, the primary motivations for this work are:

The need to develop secure searchable encryption schemes for distributed environments.

- The need to enable the authenticated users with a more secure search facility to the files.
- The need to speed up the upload/download processes for distributed encrypted files.
- The need to keep a balance between the efficiency and the security.

Considering these needs when developing the proposed system has helped to produce more secure and faster SSE system.

To this end, this paper introduces a scheme called multi-server searchable data crypt (MS-SDC) which extends the work presented in (Poh et al. 2017, 2018). The proposed scheme is introduced with new features keeping the balance between security and performance unlike schemes presented in (Hoang et al. 2018, 2019; Stefanov et al. 2013). The proposed work also reduces the limitations found in the SDV scheme (Hahn and Kerschbaum 2014; Poh et al. 2017, 2018; Naveed et al. 2014) with more than two servers unlike schemes presented in (Cash et al. 2013; Chang and Mitzenmacher 2005; Chase and Kamara 2010; Curtmola et al. 2011; Faber et al. 2015; Ishai et al. 2016; Peter et al. 2014). With the proposed MS-SDC, in an efficient and private manner, the user can upload encrypted documents to be stored as split blocks on multiple storage providers with the ability of retrieval and search for certain text across these encrypted documents using our developed search engine.

The main strength points provided by the proposed MS-SDC system can be summarized as follows:

- The system allows documents to be split into encrypted blocks and distributed across multiple servers which ensures that any single storage provider holds only a part of the file/block which provides file security.
- The number of used servers and block size are configurable to reduce the information leakage.
- The system gives the user the ability to search on server-side among all these encrypted files without having to decrypt them by extracting keywords for each uploaded document (in cryptographically hashed form) to hide any data access patterns on the server-side.
- The designed search engine of the MS-SDC system, which is developed to search for certain file using file/query keywords and to display all the files related to these keywords ranked by matching.
- A master key is randomly generated for each uploaded file to increase the security of the system.
- The system is developed as multithreaded-application for faster execution.
- The system can handle all types of files not only the text ones.
- The system can run on any operating systems.

The rest of the paper is organized as follows: Section II presents the Searchable Encryption Scenario. Section III

summarizes the previous efforts in SSE scheme development. Section IV introduces in details the proposed MS-SDC scheme. Section V focuses on the MS-SDC scheme implementation, while section VI focuses on performance evaluation results. Section VII presents the security analysis of the MS-SDC scheme. Finally, section VIII presents our conclusions.

2 Searchable encryption

SE (Searchable Encryption) is a perfect way to keep users' sensitive data safe besides keyword-search ability on the server-side. SE allows searching on encrypted data without leaking information in the original data. The two main branches of SE are SSE and PEKS (Bösch et al. 2014; Kim et al. 2018; Wang et al. 2016). SSE (Searchable Symmetric Encryption) is the symmetric type of encryption that uses a private key for each user. In this manner, the user having the private key is able to produce encrypted documents and to create trapdoors for search. On the other hand, PEKS (Public key Encryption with Keyword Search) is the asymmetric type of encryption that uses a public key for all the users, but only the user with the private key can create trapdoors for search.

Searchable Symmetric Encryption (SSE) allows the user to store the data in an encrypted form on a semi-honest server/storage provider (e.g., a cloud provider), such that this encrypted data remains searchable without decrypting and without leaking the original data content or the words being searched for to the server. In most cases, this is achieved by introducing a searchable encrypted index, which is stored together with the encrypted data (e. g., documents) on a server. To enable the server to query the data, the client creates a trapdoor that allows the server to do the search on behalf of the client.

This current paper is interested in a searchable symmetric encryption scheme scenario shown in Fig. 1. The scenario includes mainly four entities: data owner, data user, cloud service provider, and key generator (Bösch et al. 2014; Mohamad et al. 2019; Wang et al. 2016). In this scenario, a data owner wants to store a set of documents on a cloud server because of the limitation of storage resources (or any other reasons). As any server has a problem with privacy, the owner has to encrypt the documents before uploading them. The data and metadata are encrypted with a cryptographic scheme that enables searching capability. If either the data owner or a licensed user needs to get specific documents, the user will need to submit some information in terms of query keywords to the cloud service provider. Upon receiving the encrypted search queries from the data user, the cloud service provider tests the encrypted queries in the cloud storage.

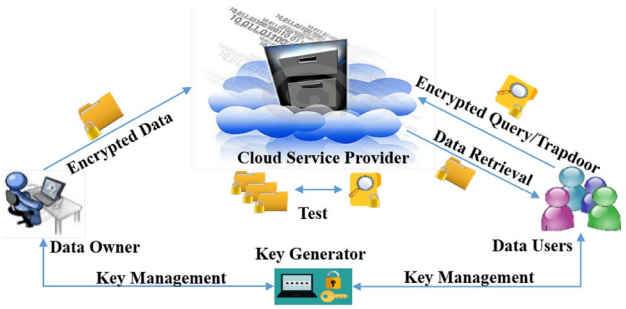

Fig. 1 Model of SSE scenario

The encrypted data that satisfies the search criteria is retrieved and sent back to the user upon completion of the test. The cloud service provider should not learn any information from this operation.

There are two types of SSE, static and dynamic. A static SSE is where the data is prepared and uploaded to the storage server once and after that only search queries are made. A dynamic SSE allows for data adding, removing or modification after the first uploading (Mohamad et al. 2019). Generally, a static SSE scenario is composed of six polynomial-time algorithms (Mohamad et al. 2019; Wang et al. 2016):

(1) A key generation algorithm: *Keygen* (1^k) which is executed by the client. Its input is a security parameter k and it outputs a secret key K.
(2) A keyword index generation algorithm: *BuildIndex (K, D)* which is executed by the client. It takes a secret key K and a set of documents D (the documents-keywords mapping DB) as inputs, and outputs a keyword index I.
(3) A symmetric encryption algorithm: *Encryption (D, K)* which is run by the client and is usually probabilistic. It takes a set of document D and keys and outputs a set of ciphertexts C of the documents.
(4) A keyword trapdoor generation algorithm: *Trapdoor (K, w)* which is executed by the client and is usually deterministic. Its inputs are both a secret key K and a query keyword w producing a trapdoor Tw as output for the keyword w.
(5) A keyword search algorithm: *Search (I, Tw)* which is executed by the server and is a deterministic interactive algorithm. It takes both a keyword index I and a trapdoor Tw as inputs and outputs a set of documents D (w) that contain the query keyword w.
(6) A symmetric decryption algorithm: *Decryption (C, K)* which is run on the client. It takes a set of ciphertexts C and keys K as inputs and outputs a set of documents D.

On the other side, there are dynamic SSE schemes that include the Update Algorithm which takes as input a document, the list of keywords and an operation name such as add, remove and modify. The Update algorithm outputs a new index and ciphertext.

Let D (w) denote the set of documents containing keyword w and id (d) denote the identifier for document d∈D. An SSE scheme is correct if the symmetric encryption scheme deployed is correct and for all keywords w, Search (Trapdoor $_{K}$ (w), I) = {id (d) |d∈D (w)} (Mohamad et al. 2019).

3 Review of the previous work

In this section, a brief summary of previous works dealing with the searchable encryption schemes is presented. SSE algorithms have been gradually developed through continuous work. At the beginning, single keyword search scheme was presented by Song et al. (2000) who proposed the first SE practical scheme for searching in encrypted data by using a special two-layered encryption construct that allows searching the ciphertext with a sequential scan. To improve the search efficiency, Goh et al. (2003) proposed a secure index construction, the idea is to use a pre-built dictionary of search keywords to build an index per document where the searching perform over the search indexes instead of scanning the entire ciphertext.

Curtmola et al. (2011) proposed a keyword-based secure index, where introduced single-keyword searched SSE with formal security definition, followed by refinements with extended capabilities such as a ranked query (Wang et al. 2010), this is more efficient in searching a query than the scheme proposed by Goh (2003). Van Liesdonk et al. (2010) proposed a dynamic SSE scheme by introducing index per keyword with an efficient search and update in contrast to the schemes in (Curtmola et al. 2011; Goh 2003).

Various researchers (Cash et al. 2014; Kamara et al. 2012; Stefanov et al. 2014) have also focused on the dynamic property extended from the model proposed by Curtmola et al. (2011) to support updates. The first DSSE (Dynamic Searchable Symmetric Encryption) scheme (Kamara et al. 2012) satisfied the conditions of sub-linear search time, security against adaptive chosen-keyword attacks, and dynamic. The first actual DSSE implementation on a very large dataset was (Cash et al. 2014), the most security DSSE scheme was proposed in (Stefanov et al. 2014). Some schemes can only run in limited environments. For example, the DSSE scheme in (Naveed et al. 2014) requires a cloud server providing no client computing capacity, but supports file upload and download; and the scheme in (Hahn and Kerschbaum 2014) can be implemented even if the client has no storage capacity. Wu et al. (2019) implemented several DSSE systems to compare their efficiency and security and identify the various disadvantages with a view to developing an improved system by Hiding Search Pattern. While others such as Cash et al. (2015) and Zhang et al. (2016) focused on attacks on the SSE scheme by presenting attack models based on an adversarial server's prior knowledge. Some others handled the issue from another perspective such as Ishai et al. (2016) who focused on improving SSE scheme security.

Unlike the above mentioned works based on a single server, the schemes in (Ishai et al. 2016; Poh et al. 2017, 2018; Mohamad et al. 2019) are proposed to work on two or more servers. At first, Bösch et al. (2014) introduced query proxy working side by side with the storage server but this leads to the scheme may not be scalable at which the query proxy re-encrypt the query and re-process the index every time a query is submitted. Ishai et al. (2016) introduced a helper server as an additional server that is to minimize leakage and for a wide variety of search functionalities, that also to scale to large databases. In contrast to these proposals, Poh et al. (2018) introduced an SSE scheme that divides the message into blocks and randomly distributes them among many different servers. This reduces leakage compared to schemes that use one storage server. The scheme in (Poh et al. 2017) is an extension to the scheme in (Poh et al. 2018) which focuses on improving the computation overhead compared to other schemes that submit documents directly to a cloud storage service.

SSE scheme which is based on a conjunctive keyword search is more efficient and suitable for real-time applications where it allows a user to obtain documents containing several keywords during a single query. However, it is inefficient and leaks some information to the server (Golle et al., 2004) proposed the first two conjunctive keyword search schemes. To improve efficiency, the schemes (Cash et al. 2013; Faber et al. 2015) proposed also studies conducted on conjunctive search. Cash et al. (2013) extended conjunctive query to Boolean query. Faber et al. (2015) extended Cash et al. (2013) to support range, substring, and phrase queries.

SSE scheme based on a ranked keyword search can optimize search results by returning the most relevant documents to reduce network traffic and enhance system usability. Cao et al. (2013) were the first to present a multi-keyword ranked search scheme with "coordinate matching" measurement. However, their search results are ranked based on the number of matching keywords without considering the importance of different keywords.

There are some schemes that overcome the storage/search problem on untrusted servers by assuming that the server is untrusted, and the client is trusted, including the client's processor, memory (RAM), and disk. Stefanov et al. (2013) presented Path ORAM, an extremely simple Oblivious RAM protocol for the encrypted index data on the server with a small amount of client storage to hide the data access

pattern. Hoang, et al. (2018, 2019) developed a series of Oblivious DSSE schemes called ODSE, which enabled oblivious access on the encrypted index with high security and improved efficiency over the use of generic ORAM. However, these types of schemes still have low efficiency with high security.

In order to achieve sublinear search complexity, SSE scheme designs deploy inverted index where the index key is encoded keywords and the index values are document identifiers. Examples of such designs include Curtmola et al. (2011), Chase and Kamara (2010), Naveed et al. (2014), Cash et al. (2014).

The trade-off in enabling search on ciphertext is the disclosure of some information regarding the documents, called leakage. Despite the leakage, an SSE scheme aims to protect the confidentiality of the stored documents and the queried keyword using symmetric cryptographic schemes like Chase and Kamara (2010), is the definition accepted for SSE schemes security.

However, attacks by Kuzu et al. (2012), Zhang et al. (2016) and Cash et al. (2015) were successful in recovering query keywords in published index-based SSE schemes. Wright and Pouloit (2017) generalized the attacks as a statistic inference attack and produced a statistical method framework to detect such vulnerabilities in an SSE scheme. Cash et al. (2015) studied the practical attacks and defined the attack goals and adversary capabilities. In addition, they categorized SSE leakage profiles to identify the extent of vulnerability of an SSE design to the different attacks. In (Mohamad et al. 2019), a new security definition against the practical attacks was proposed and proven to imply current security definitions. But, we must be sure that even a small increase in the amount of leaked information allows for constructing schemes breaking the lower bounds on the tradeoffs between the efficiency and the security for searchable encryption schemes (Bost and Fouque 2019).

The conclusion of those previous works is that some of them suffer the limitations of using a single server and others deal with only two servers and introduce limited features with some security problems. This was the motivation for us to develop a system that can enhance most of the aforementioned problems in an attempt to both obtain more features handling the limitations of these schemes besides preserving a simple user interface with high security against hacking.

4 The proposed multi-server searchablr data crypt (MS-SDC) system

Many challenges and issues are facing the distributed SSE such as working with different storage areas, storing various file types, securing the search queries and search keywords, and more. Focusing on some of these challenges, the main objective of the proposed system is to enable a secure distributed allocation of user's files on multi-storage areas. Through this system, the user can upload his own files in an encrypted form to remote hosting sites such as the cloud. The system splits the user files into blocks and distributes them randomly across multi-servers to ensure that more than one server hold the whole file. At the same time, the system enables the user to search for his encrypted files in the remote storage area without compromising the security of his files.

4.1 Plan of solution

In order to solve the above-mentioned problems, found in the previous works and discussed in section III, we introduce the proposed system that adds several novel functions and improvements to the existing systems (such as schemes (Hoang et al. 2018, 2019; Poh et al. 2017; Stefanov et al. 2013). A pointed comparison between the characteristics of our proposed system and of the system presented in (Poh et al. 2017) is given in Table 1. The comparison shows the various characteristics and how our system has improved them. The details of these improvements will be discussed in the rest of this section.

The work in (Hoang et al. 2018, 2019; Stefanov et al. 2013) has a major problem is that over time the server can identify some of the document content through the search process and sending the query/document keywords to it. Therefore, these schemes improve security by hiding the data access pattern from the server, as the server does not deal with the query/document keywords directly, only through the RAM protocol between the server and client. But, it still causes low efficiency, low speed, depending on the client's RAM and this is despite that many improvements have been applied to solve this problem. There are no features presented by those schemes other than improving system security.

To solve such a problem in our system, we try to make a balance between security and efficiency; by presenting different layers of safety while maintaining the possibility of providing and adding a lot of features (For example; a search engine and others, and running on multithread for maintaining efficiency in terms of speed. As for the point that the server can identify some of the file content through the search process, we prevent the server to deal directly with the query/file keywords. We allow the server to deal only with the searchable/trapdoor keywords (which are cryptographically hashed form of file keywords), only through the MS-SDC Manager between the server and client.

The proposed system is dedicated to distributing the user's files into multiple server storage areas in an encrypted form with searchability facility of these files to the authenticated users. As mentioned before, it extends the work

Table 1 Comparison between the characteristics of our system and system presented in (Poh et al. 2017)

Characteristics	Previous scheme (Poh et al. 2017)	Problem identified in (Poh et al. 2017)	Our system solution
Master key (mk)	Generate a single random master key for all user's uploaded files	Security preach; files can be easily hacked if master key is known	Generate a random master key for each uploaded file
Block size	Fixed/constant size blocks regardless of document size	Impractical in case of large sized files	Block size is configurable, can change according to file size
Searchability	There is no searchability	Files cannot be retrieved	A designed search engine is built
No. of servers	Fixed number of servers to deploy the file	Server load imbalance, does not support dynamic environments	Configurable/changeable number of servers is used according to the current system status
Uploading file with same name	Uploading multiple file with same name is allowed	Uploaded files with same name causes file overwrite (i.e. name collision)	Name collision problem is solved by renaming the file in an indexed way (i.e. given a copy number as in current OS)
No. of threads	Implemented as a single-thread application	Slow uploading of files	Implemented as a multithreaded application to speed up the upload process specially for large files
File type	Applied on text documents only	Only few files can be uploaded	Applied to images and videos and other types (like pdf) without any distortion or reduction of their quality
Type of O.S	A virtual machine that runs on Ubuntu 14.04	Limited usage for certain types of environments	Runs on both Windows/Linux and also Android operating systems

presented in (Poh et al. 2017) by adding advanced features to it in order to increase its security and flexibility. In order to attain such goals, the system architecture is decomposed into three main components as shown in Fig. 2. These components are named: MS-SDC Interface, MS-SDC Manager, and MS-SDC API, which provide the techniques contributing to our main results. The proposed MS-SDC Manager design is kept simple and generic so that any of its parts can be modified easily for appending any future features. The role of each component is described in the next subsections.

4.2 MS-SDC interface

This component is responsible for providing the proper interface for the MS-SDC system that enables the system user to upload and download files, after passing the authentication process. It also provides the users with the search facility for files using keyword(s) across the previously uploaded encrypted files without having to decrypt them, through our designed search engine.

4.3 MS-SDC manager

This component is the most crucial component in the system. It is responsible for managing the server databases, and providing the core functionalities of the system. It provides the following main functions:

(1) Preparing the file for upload and download.
(2) Extracting document keywords for each uploaded document.
(3) Creating and maintaining both the index entry and the query keyword tables.
(4) Linking the two previous tables to constitute the query index table.
(5) Managing query keywords from the search engine to retrieve the corresponding documents for these keywords ranked by matching.
(6) Hashing the data before being inserted into the database for an extra level of security.

The above functionalities are implemented through the interaction with the MS-SDC API.

4.4 MS-SDC API

This component is responsible for implementing our MS-SDC scheme and the required algorithms. Several APIs are designed to handle the required functions needed by MS-SDC Interface and MS-SDC Manager. It also implements the interactions with the databases and many other processes as will be shown next.

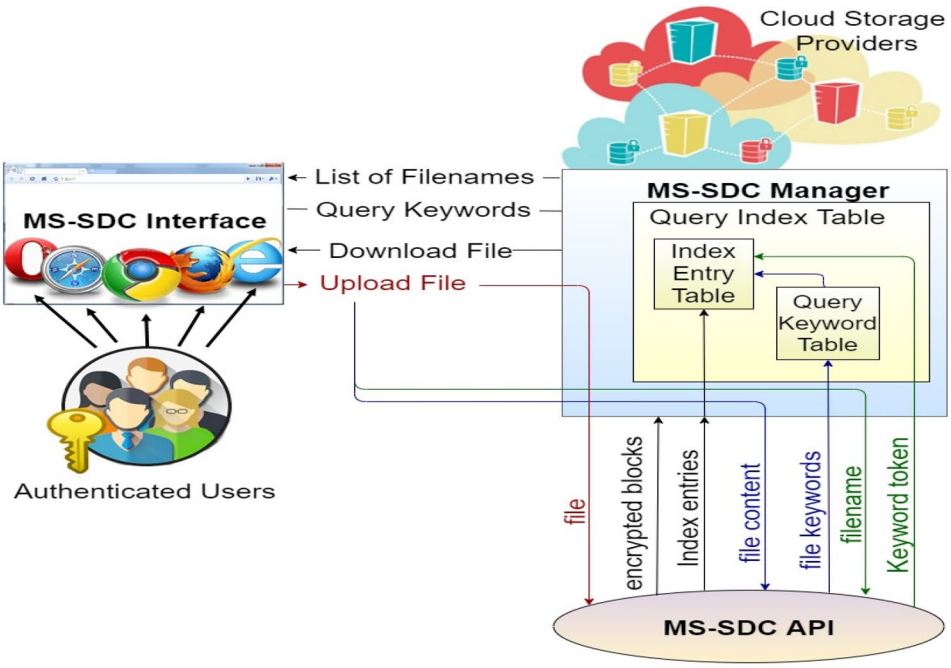

Fig. 2 Searchable data crypt (SDC) architecture

4.4.1 System components interaction

This section aims to describe the interconnection between the three components of the MS-SDC system (Interface, Manager, and API), and how they accomplish the work together. There are specifically three functional layers provided by our system for its users: upload, download, and search for the files using keyword(s).

4.4.1.1 The upload layer function Using this function, the data owner can upload the files he/she wants to store on the remote storage area. The data owner submits a file through the MS-SDC Interface, which passes it to the MS-SDC Manager. The data owner can choose the number of servers to store the file on, otherwise the MS-SDC Manager will determine the number according to the protocol used by the MS-SDC Manager for this manner. The MS-SDC Manager calls the proper MS-SDC APIs to execute the underlying distributed SSE scheme.

The MS-SDC API takes one of the following actions to complete the upload function according to the nature of the uploaded file. If the user uploaded a plain file, the system will perform the necessary steps to extract the file keywords from the file content. This step is only required if the user has uploaded an unencrypted file; this can rarely happen and is not recommended for security. Nevertheless, what normally happens is that the user appends the file (whether it is a document, image, or video) with its keywords to be extracted as file keywords. It is intuitive to consider the FileName as one of the file keywords. The workflow of the upload file process is summarized in Fig. 3.

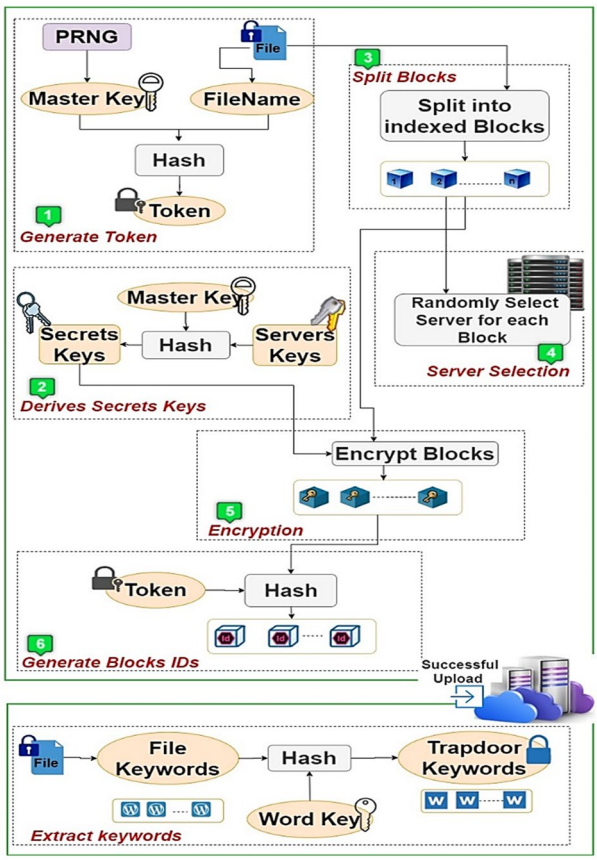

Fig. 3 The workflow of the upload file function

This function is responsible for splitting the file into indexed blocks, selecting a server for each block randomly, encrypting each block using the server key to finally get a group of encrypted blocks that are distributed to multiple storage areas. It is also responsible for maintaining the index entry table and query keywords table for subsequent layers.

4.4.1.2 The search layer function Our system provides the user with search functionalities that are performed as follows:

(1) The user submits one or more keywords through the MS-SDC Interface (across search engine) which in turn forwards these query keywords to the MS-SDC Manager.
(2) The MS-SDC Manager searches the query index table to obtain a list of filenames matching those keywords (ranked by matching degree) with the MS-SDC API assistance
(3) The MS-SDC Interface presents the list of filenames to the user.

From the basic searchable encryption scheme concept (Mohamad et al. 2019; Wang et al. 2016), only document index entries can be obtained. The idea presented here is to create a query index table by MS-SDC Manager instead of creating the index table by the SSE scheme to give our system flexibility to search.

4.4.1.3 The download layer function This layer provides the authenticated data users to download a file stored in the cloud storage area. The user submits/chooses the filename to download to the MS-SDC Manager (whether it was from the list of filenames generated by a previous search process, or from the list of previously uploaded filenames). As the file is distributed to multiple servers, this function is responsible for collecting file parts from these servers. The workflow of the download file process is summarized in Fig. 4.

To reconstruct the file from different storage servers, the MS-SDC Manager takes the following actions:

(1) Searches across the query index table using the filename to get the token and blocks ids.
(2) Collects the blocks distributed across the servers using the token and blocks ids to finally get each block and its server key.
(3) Decrypts each block according to its servers' key.
(4) Merges/Sorts the blocks according to blocks index to build up the original file.

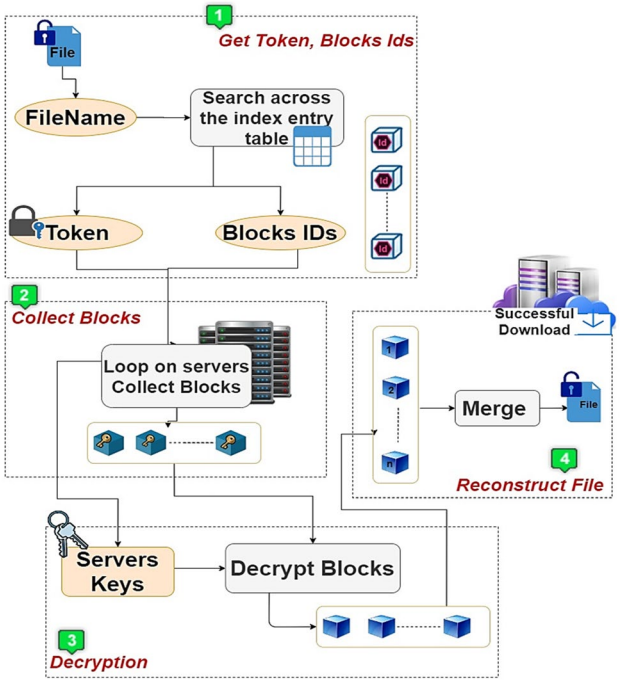

Fig. 4 The workflow of the download file function

4.4.2 The proposed system details

The implementation of the proposed system has been divided into four algorithms namely; FileUpload_Setup, KeywordExtraction, FileDownload, and SearchEngine_Keyword. The details of these algorithms are given in the next subsections. Our algorithms are based on the common SSE scheme algorithms found in (Mohamad et al. 2019; Wang et al. 2016) with proper modifications to cope with the changes we have made to the system at hand as follows:

- FileUpload_Setup: implemented using Keygen, Encrypt, Trapdoor, and BuildIndex algorithms (Mohamad et al. 2019; Wang et al. 2016).
- KeywordExtraction: implemented using Trapdoor and BuildIndex algorithms (Mohamad et al. 2019; Wang et al. 2016).
- SearchEngine_Keyword: implemented using Trapdoor and Search algorithms (Mohamad et al. 2019; Wang et al. 2016).
- FileDownload: implemented using Trapdoor, Search, and Decrypt algorithms (Mohamad et al. 2019; Wang et al. 2016).

It is worth mentioning that all the abbreviations/notations used in the definitions, and the MS-SDC scheme algorithms are listed in Table 2.

Table 2 List of abbreviations (notations)

Notation	Description
Pr	A probability function
C_i	An encrypted block i
D_i	A document i
D (w)	A set of documents containing keyword w
F_i, f_i	A file i, a block of file i
Fw	A file keyword
I	An index entry table
id (x)	An identifier of an item x
K	A secret key, a set of cryptographic keys
$L.^{\downarrow}$	A leakage profile that states information leakage of the L.phase in a SSE scheme
mk	Master key
negl (λ)	A negligible function with security parameter λ
Qw	A query keyword
S_i	A server i
Sw	A stop word
tok_i	A file keyword token
Tw	A trapdoor keyword
T	A query index table
W	A query keyword table
wk	A word key

4.4.2.1 FileUpload_Setup algorithm The uploaded files are processed according to the steps presented in the FileUpload_Setup algorithm shown in Fig. 5. The processing includes the secure distribution of the file into multiple servers and the extraction of the keyword index table. The FileUpload_Setup algorithm performs the following steps:

(1) A random master key (mk) is generated for each uploaded file using a pseudorandom function.
(2) The Keygen algorithm generates a set of secrets keys K using a hash function (master key, servers' keys).
(3) The file (F_i) is split into a set of blocks $F_i = \{f_{i1}, f_{i2}, \ldots, f_{ix}\}$ according to a predefined block size.
(4) The Trapdoor algorithm generates a keyword token (tok_i) for the file using a hash function (master key, filename).
(5) The file blocks are distributed across a number of servers chosen randomly.
(6) The Encrypt algorithm encrypts the file blocks into a set of encrypted blocks $C_i = \{c_{i1}, c_{i2}, \ldots, c_{ix}\}$ using the encryption function (each block with its secret/server' key).
(7) For each block, a block id is generated using a hash function (tok_i, encrypted block) to be stored on the server.
(8) The BuildIndex algorithm creates the index entry table I to store the links between the filename token (toki), the file blocks, and the servers storing these blocks.

From steps (5, 6 and 7), we notice that, a set of process is performed on each file block; randomly selecting a server, encrypting the block with its server key, and getting its id.

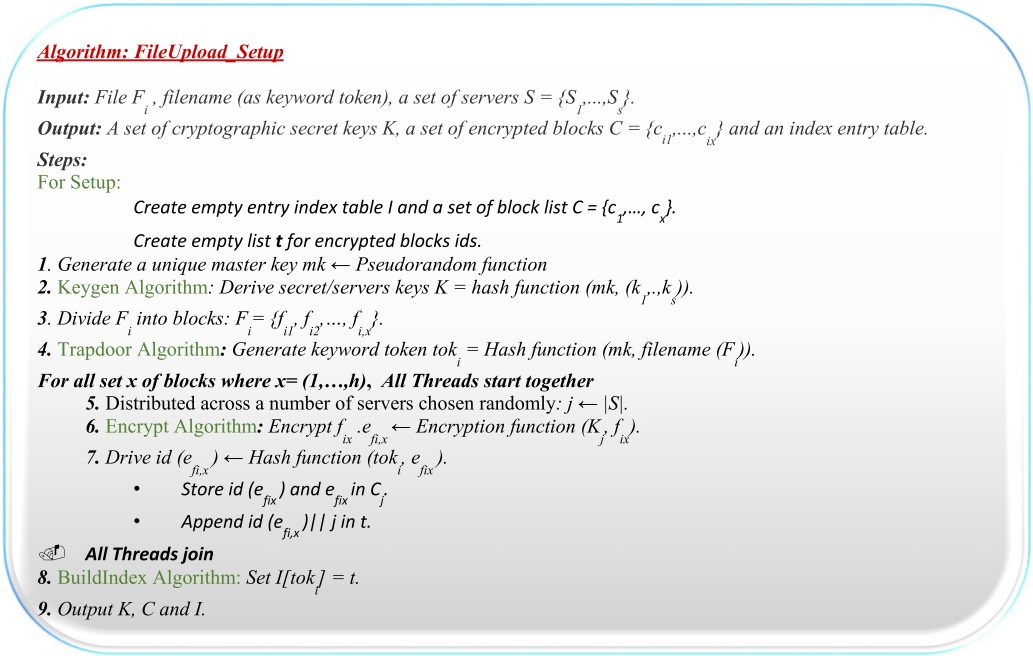

Fig. 5 FileUpload_Setup algorithm

One of the contributions of our work, which is different from the work presented in (Poh et al. 2017), is the usage of the multi-threading concept in the execution of blocks' processes where all the blocks' processes are performed concurrently to avoid wasting time especially in case of the existence of a large number of blocks to be stored. This guarantees that all the threads/blocks start and end their operations at the same time. The statements "all Threads start together" and "all Threads join" are utilized for this purpose as shown in Fig. 5.

4.4.2.2 KeywordExtraction algorithm This algorithm is responsible for extracting the document keywords as shown in Fig. 6. When the data owner uploads a file to the system, there are two options: (A) an unencrypted file is uploaded, or (B) an encrypted file is uploaded and the user appends the file's keywords along with the file. Both options are handled by the KeywordExtraction algorithm as follows:

4.4.2.3 Option (A): uploading unencrypted file The KeywordExtraction algorithm extracts the file keywords as follows:

The file content is split into a set of words.
These words are filtered by removing a list of stopwords $Sw = \{Sw_1,\ldots,Sw_x\}$, which is a list containing the common words in the language (as used in Google search engine, e.g. pronouns).
The frequency of the words occurance is computed to remove the words with frequencies under a certain threshold (for example, ≥ 2), as they are considered unimportant keywords.
Finally, the file keywords $Fw = \{Fw_1,\ldots,Fw_v\}$ are extracted.

4.4.2.4 Option (B): uploading encrypted file and a list of keywords The KeywordExtraction algorithm extracts/separates file keywords $Fw = \{Fw_1,\ldots,Fw_v\}$ directly from the file.

After preforming one of those options, the rest of the algorithm steps are followed:

(1) The Trapdoor algorithm is used to generate a set of trapdoor keywords using a hash function (word key, file keywords extracted). These trapdoors are used afterward for search.
(2) The BuildIndex algorithm creates the query keyword table (W) to store the links between the file (tok_i) and these trapdoor keywords. Then, the query index table T is created by linking the index entry table with a query keyword table W.

Figure 7 shows an example of the index entry table and query keyword table. The index entry table links every block with its: filename, token, id, index No., Server name, and Server Key, while the query keyword table links every File (token) with its query/trapdoor keywords.

Algorithm: KeywordExtraction

Input: *File (tok_i, content), an index entry table I, a set of stopwords $Sw=\{Sw_1,\ldots,Sw_x\}$*
Output: *a query keyword table W, lookup query index table T with two dictionary level.*
Steps:
For Setup
- *Get Word key wk → constant key.*
- *Create empty query keyword table W.*

Option (A) **Uploading unencrypted file**
a. *Split file content (F_i) into a set of file words.*
b. *Filter $Sw_{1\rightarrow x}$ from file words.*
c. *Extract a set of file keywords $Fw = \{Fw_1,\ldots,Fw_y\}$ ← compute iteration of words.*

Option (B) **Uploading encrypted file and a list of keywords**
- *Extract a set of file keywords $Fw = \{Fw_1,\ldots,Fw_y\}$.*

1. Trapdoor Algorithm***:*** *Generate a set of trapdoor keywords $Tw = \{Tw_1,\ldots,Tw_y\}$ ← Hash function (wk, $Fw_{1\rightarrow y}$).*
2. BuildIndex Algorithm
- *Append ($Tw_{1\rightarrow y}$)||File(tok_i) in W.*
- *Output query keyword table W.*
- *MS-SDC controller: Create query index table T → Updated/Linked index entry table I with query keyword table W.*

3. *Output query index table T.*

Fig. 6 KeywordExtraction algorithm

Fig. 7 An example for the index entry and query keyword tables

FileName	Token	Block Id	Block Index	Server Name	Server Key
Algorithm	Asdfg463	Uaet4354c	3	S2	Sgfhjkl
Introduction	Qwe564i	Rfv3454n	6	S1	Xcvnju

Index Entry table

Token	Keywords
Asdfg463	W1, W3, W4
Qwe564i	W2, W3

Query Keyword table

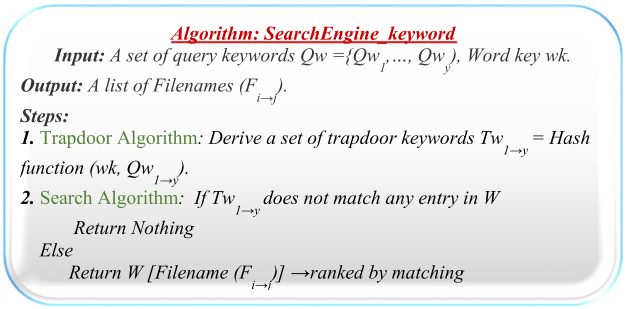

Fig. 8 SearchEngine_Keyword algorithm

Fig. 9 FileDownload algorithm

4.4.2.5 SearchEngine_Keyword algorithm The search for a certain document is performed as presented in the SearchEngine_Keyword algorithm given in Fig. 8. When the authenticated user submits query keyword(s), the SearchEngine_Keyword algorithm takes following steps:

(1) The Trapdoor algorithm generates a set of trapdoor keywords from the query keywords $Qw = \{Qw_1,\ldots, Qw_v\}$ using the same hash function (word key, query keywords). The query keywords should similar to the file keywords to get the same trapdoor for search
(2) The Search algorithm checks the query keyword table W to find any of the table entries/data that match these trapdoor keywords.
(3) If found, a list of filename $F_{i\rightarrow j}$ ranked by matching is sent back to the user. Each of these matched entries contains file token, blocks ids, blocks index, and blocks servers used for document retrieval.
(4) If no match is found, nothing is sent back.

In our implementation, these entries and the query index table T are used by the MS-SDC Manager to retrieve the corresponding filenames. The retrieved filenames are ranked by matching degree when shown to the user.

4.4.2.6 FileDownload algorithm The file download function is performed as presented in the FileDownload algorithm shown in Fig. 9. In order to collect the file for download, we need to retrieve its blocks from the servers containing them, then concatenate these blocks after being decrypted following the next steps:

1 The Trapdoor algorithm is used to obtain the trapdoor token (tok_i) using the hash function (mk, filename of the desired file).
2 The Search algorithm checks the index entry table I to find the index entries/data that match the trapdoor token (tok_i). This entry/data contains the links used for blocks download from storage servers.
3 The Decrypt algorithm, in our implementation, uses this entry/data by the MS-SDC Manager to retrieve the encrypted blocks, decrypt them using the decryption function (each block with its secret/server' key).
4 The blocks are concatenated according to blocks index to finally build the file ready for download.

5 MS-SDC system implementation

This section presents the details of the MS-SDC system implementation along with the performance criteria evaluation. To prove the validity and superiority of the proposed system, we have built a prototype of the system simulating it as multiple servers on a single local machine. The system

prototype was written as a multi-threaded application. The implementation environment runs on an Intel® Core™ i7-8550U CPU @ 1.80 GHz–1.99 GHz with 8 GB of RAM memory. The proposed MS-SDC Manager is implemented using Java Servlet on Apache Tomcat version 8.0.27. The MS-SDC API with the implementation of the multi-server $SSE_{MS\text{-}SDC}$ scheme is developed using Java NetBeans IDE 8.2. The web interface for the MS-SDC Interface is based on JSP and MySQL is deployed as the underlying database.

In this section, we will describe the details of each of the implemented functions and algorithms used in building the proposed system. The details will be organized as we have organized the discussion of the system components in section IV after adding the technical specification of the system with illustrative examples.

5.1 File uploading function

The MS-SDC system is implemented to be used as two different scenarios to increase its flexibility. The first one is to upload the encrypted file accompanied with its keywords, while the second one is to upload the plain file where the encryption and keyword extraction is performed on the server side. The second scenario is dedicated for cases where the user cannot perform the encryption and keyword extraction processes (for example, the user is using limited capabilities or resources devices). Of course, this is done after the proper authentication steps of the user.

Whether the file was plain or encrypted, the system encrypts the file and also hashes the file keywords for a second layer of security guaranteed by the system. For more usability of the system, if the uploaded file is not a searchable file (i.e. not textual file), the user can still supply some optional keywords to reach the file. For example, if it is a PDF file, the user can choose keywords representing the text or images inside the file to be able to retrieve it afterwards.

The two scenarios are implemented in the file upload function with the aid of two correlated algorithms: FileUpload_Setup and KeywordExtraction as discussed below. Figure 10 illustrates the sequence of the operations performed by the MS-SDC API for the file upload (FileUpload_Setup and KeywordExtraction) in detail.

5.1.1 FileUpload_Setup algorithm implementation

The first part of the file submission "FileUpload_Setup" is to prepare the file for distributed storage on the cloud as shown in Fig. 10. Firstly, we must set up the master key and the server/secret keys. A master key is randomly generated by PRNG "pseudorandom number generator" for each uploaded file to ensure high level of security. When a user uploads the file via the file upload interface, the MS-SDC

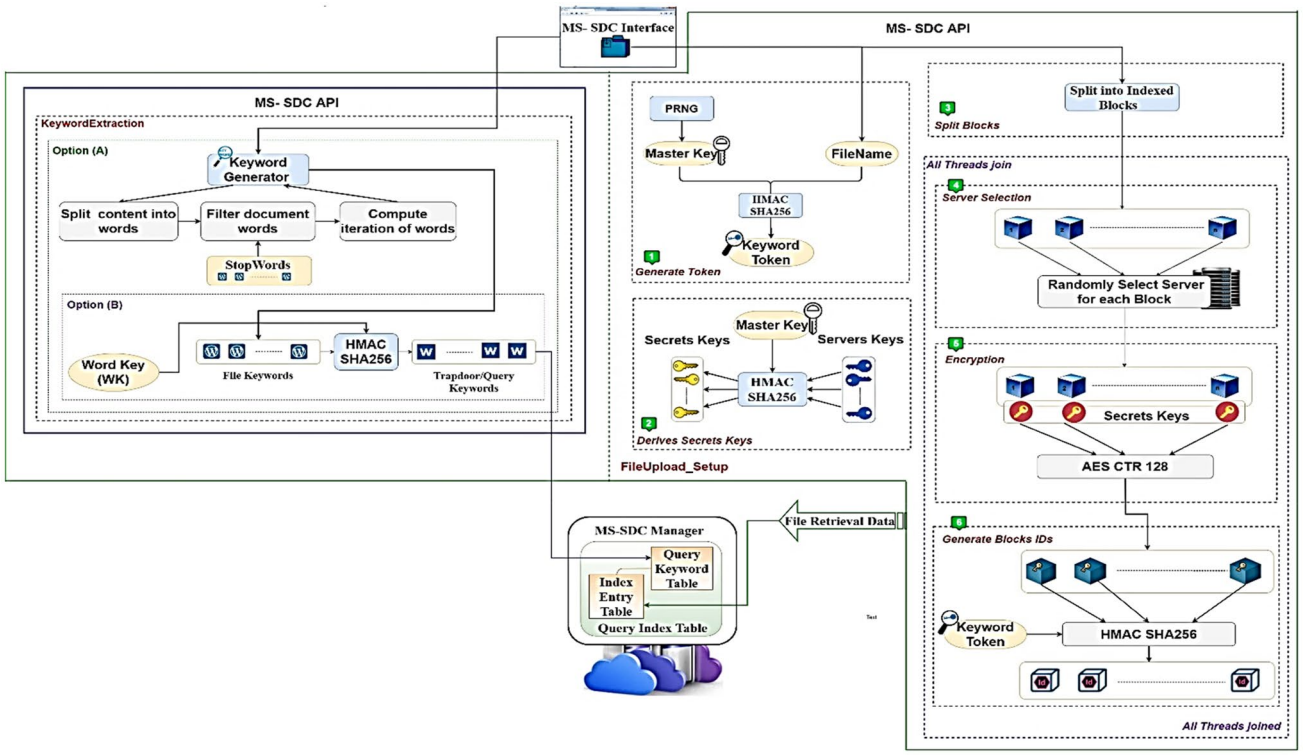

Fig. 10 MS-SDC: file submission "FileUpload_Setup"

API uses the filename as the keyword token. The keyword token is derived from both the master key and the filename using HMAC-SHA256 hashing function (Poh et al. 2017; Knopf 2007).

To avoid file overwriting, we must check the database to know if the filename is already found on the server by the same data owner or not. If it found, we make an index for the filename like windows OS (e.g. filename1, filename2, and so on), and then the file can be store on the database smoothly.

Two main steps are required to store the file on the cloud multiple servers:

(1) Prepare the file for distribution: FileUpload_Setup
(2) Prepare the file for subsequent search: KeywordExtraction

The first step requires the MS-SDC API to split this file into indexed blocks (the block size is a configurable parameter by the system owner, and is not fixed as implemented in previous work (Poh et al. 2017). After performing the first step, the following operations are simultaneously performed in parallel (via multi-threading) for all the blocks; each thread randomly selects one of the available servers (the number of servers is a configurable parameter by the system owner and is not fixed as implemented in previous work (Poh et al. 2017)). Each block is encrypted with the corresponding server key using AES-128-CTR encryption algorithm (Lipmaa et al. 2000), then is hashed with the keyword token using HMAC-SHA256 hashing function to produce the block identifier.

The required data for each block/thread (the file token, block identifier, block index, server name, and server key) is inserted as a part of the index table entry to recollect/reconstruct the file during the download process. Finally, the index entry and the encrypted blocks are sent to the MS-SDC Manager for distributing the encrypted blocks to the corresponding storage servers.

5.1.2 KeywordExtraction algorithm implementation

The first step of the file submission is "KeywordExtraction" which prepares the file for the search facility, as shown in Fig. 10. The KeywordExtraction algorithm extracts the file keywords from both scenarios: scenario (A) the textual plain file is uploaded, and scenario (B) the encrypted file is uploaded.

(1) Scenario (A) the textual plain file is uploaded: the MS-SDC API splits the file content into a set of words and then filters these words using a list of words called stopwords (which is a configurable parameter by the system owner, in our implementation we used the list of words that used in Google search engine). From the words of the previous step, the file keywords are extracted by computing their frequency. The words with frequency (≤ 2) are omitted and not considered keywords.
(2) Scenario (B) the encrypted file is uploaded: the user appends the file's keywords along with the file. Then the MS-SDC API uses these keywords directly and add the file name as a keyword to be used for search by file name.

The keywords obtained from any of the two scenarios for file upload are then hashed with the word key using HMAC-SHA256 hashing function, to produce the trapdoor/query keywords. All the previous data are inserted as a part of the query keyword table to be next used during the search process. Finally, the MS-SDC Manager creates the linked table called the query index table by updating the index entry table with the query keyword table.

As an example of the complete file upload process (FileUpload_Setup and KeywordExtraction), Figs. 11 and 12 show the submission for an 8 KB file called "Intoduction.txt" at which will be distributed to available servers with a block size of 2 KB. The file is split into 4 blocks (2 KB each) due to the predefined file and block sizes. Figure 11 shows the user list of uploaded files while Fig. 12 presents a snapshot of the full submission log for such operation showing Filename, Block Count, BlockIds (server name, and block identifiers), and the total time taken for file uploading "FileUpload_Setup and KeywordExtraction". Whereas the system developed as a multithreading application, we note that the time of the slowest thread (90 ms) is the time taken to distribute all the blocks across servers.

5.2 Document search function

When a user inserts query keywords, the MS-SDC Interface directs these keywords to the MS-SDC Manager, then the SDC Manager forwards these keywords back to the MS-SDC API. These keywords are hashed with the word key using the HMAC-SHA256 hashing function to produce

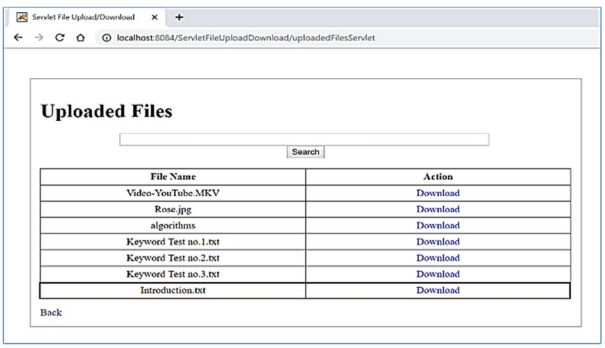

Fig. 11 List of all uploaded files

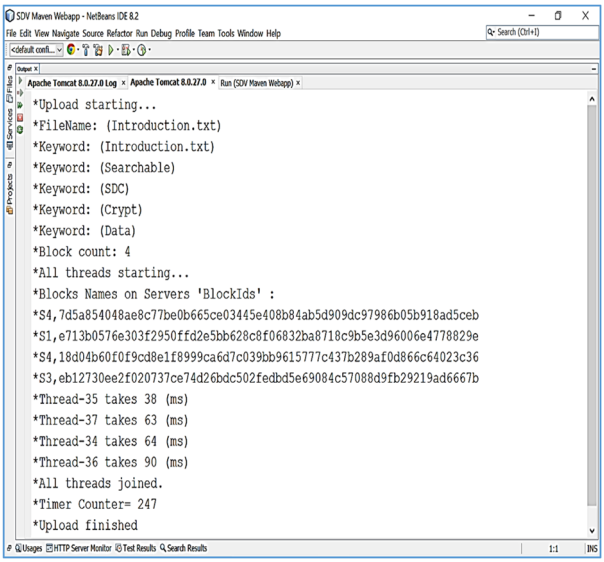

Fig. 12 Full submission log "FileUpload_Setup and KeywordExtraction"

trapdoor/query keywords. The MS-SDC Manager searches the query index table using these trapdoor/query keywords

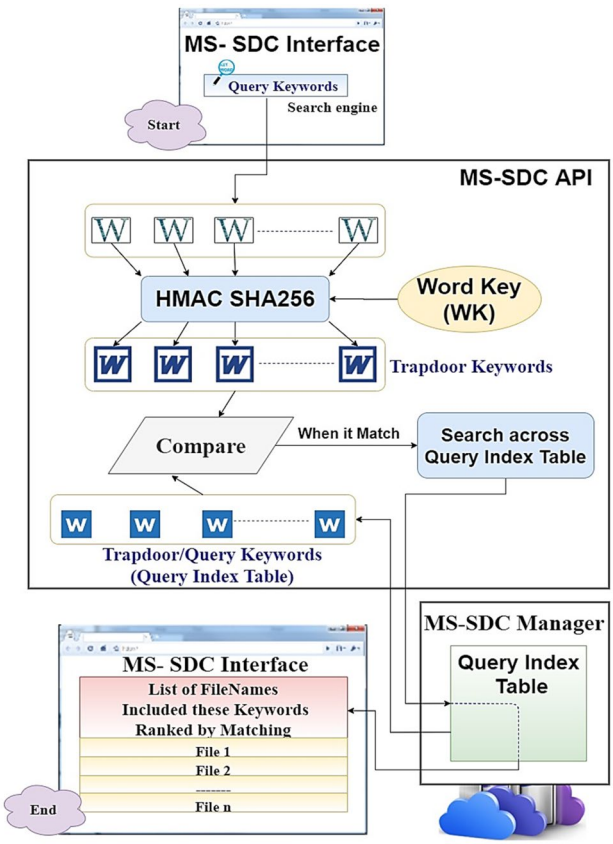

Fig. 13 MS-SDC file search details

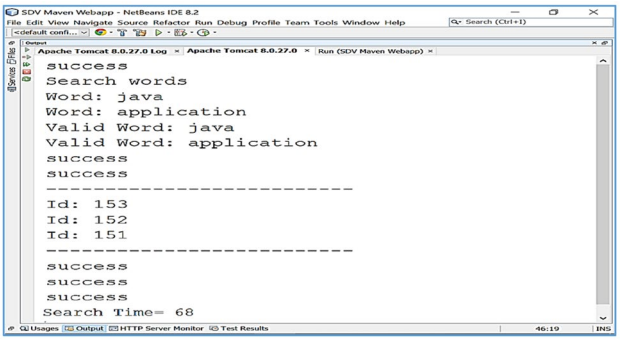

Fig. 14 Search by keyword log for files

to check if there is any matching. If there is a match, the MS-SDC Manager uses the query index table to obtain all the corresponding files to the entered keywords and rank them by matching degree. The MS-SDC Manager forwards these ranked files to the MS-SDC Interface as shown in Fig. 13.

As an example for keyword search function, Fig. 14 shows the search log while scanning the files for the keywords "java application" that generated three documents (Keyword Test no.1, Keyword Test no.2, and Keyword Test no.3) containing these keywords. The three files are ranked by matching degree. The log also shows the return of the identifiers for the corresponding files and the total search time.

5.3 File download function

Figure 15 illustrates the sequence of the operations performed by the MS-SDC API for document retrieval in detail. File download works as follows: the user selects the desired file to be downloaded via the MS-SDC Interface (whether it was from the list of searched filenames that appear to the user ranked by matching, or from the shown list of uploaded files). The MS-SDC Interface submits the filename to the MS-SDC Manager which then searches the query index table using the filename token to check if there is any matching. If there is a match, the block's identifiers and corresponding servers are retrieved. The MS-SDC Manager accesses the servers to collect the encrypted blocks and sort them according to the block number. The MS-SDC API decrypts each encrypted block with its corresponding server key using the AES-128-CTR decryption algorithm and merges these blocks into a complete file for download.

As an example of the file download process, assume we want to download an 8 KB file "Intoduction.txt" (this is the same file in Fig. 11). Figure 16 presents a snapshot of the retrieval log for such operation showing the Filename, Block Count, and BlockIds (server name, block identifiers).

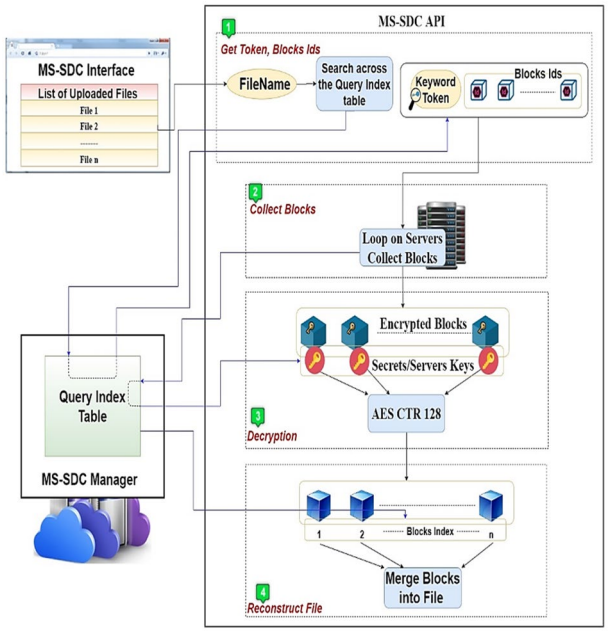

Fig. 15 MS-SDC file download in detail

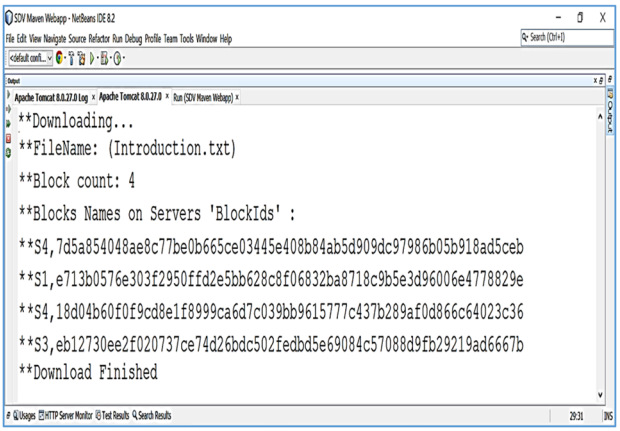

Fig. 16 Retrieval log

6 Security analysis

The entries of the index, the search token, and keywords queries must be protected to prevent any access to the stored data. Nevertheless, some information leak is inevitable for the search to be efficient but this is done under secure control. This characteristic of the scheme is called the leakage profile as was first formalized in (Curtmola et al. 2011).

A leakage function L describes what a secure protocol may leak. Security against a non-adaptive adversary A means that the adversary is not allowed to see the encrypted index or the trapdoors of any keywords before it had generated the index I and the queries (Q1,.., Qq) it wants to look for. On the contrary, the adversary A in the adaptive game is allowed to choose the queries adaptively; meaning that the choice of a query can be based on the output of the previous queries.

The leakage profile normally consists of the following three functions (Poh et al. 2017; Meharwade and Patil 2016).

- L^{setup} is information that the server got on by dealing with the encrypted documents and index, such as the block size (i.e., the block size is configurable in our scheme).
- L^{query} is the information compiled by the server in supervising messages across the search protocol, such as the number of blocks associated with the keyword search token.
- L^{update} is the information detected to the server by the changes to index and set of encrypted documents on the server such as add or removes servers and a number of keywords of each added or removed document.

Definition 1 A scheme SSE is said indistinguishable against non-adaptive adversaries according to a leakage L if for all polynomial-size A and polynomial q, there exists a simulator S such that (Poh et al. 2017; Meharwade and Patil 2016):

$$\left|\mathbf{Pr}\left[NonAdapReal_{A}^{SSE}(\lambda, q(\lambda)=1\right]\right.$$
$$\left.-\mathbf{Pr}[NonAdapIdeal_{A}^{S,I}(\lambda, q(\lambda)=1]\right| < negl(\lambda)$$

Definition 2 SSE is indistinguishable against adaptive adversaries according to a leakage L if for all polynomial-size adversaries A and all polynomial q, there exists a simulator S such that (Poh et al. 2017; Meharwade and Patil 2016).

$$\left|\mathbf{Pr}\left[AdapReal_{A}^{SSE}(\lambda, q(\lambda)=1\right]\right.$$
$$\left.-\mathbf{Pr}[AdapIdeal_{A}^{S,I}(\lambda, q(\lambda)=1]\right| < negl(\lambda)$$

By referring to definitions 1 and 2, and the description of our scheme with the underlying pseudorandom function and the symmetric encryption scheme is IND-CPA secure, we say that MS-SDC is L-secure against keyword attack by colluding servers. We define the leakage profiles L. based on an adversary A who controls all storage servers. This means the adversary sees leakages from all storage servers involved in the system.

- L^{setup}**:** For FileUpload_Setup, the information leaked to the adversary is the number of blocks on a server and the block size of the encrypted blocks. Therefore, in our current implementation, we use different block sizes.

This means that the storage servers learn different sizes of encrypted blocks, in addition to usage of a configurable number of servers which makes it more difficult for the adversary. To avoid easily hacking the system with master key leakage, a unique master key is generated randomly for each uploaded file instead of using a single master key for all uploaded files.

- L^{query}**:** For Search and KeywordExtraction, the adversary learns the number of encrypted blocks associated with the keyword token during the file download. Besides that, the adversary can learn the content of the document by the repetition of knowing the query keywords used for search on the server-side. So, we used query keywords in a hashed form, not original ones to avoid this issue.
- L^{update}**:** the leakage is similar to $L.^{query}$ at which the adversary can learn the number of encrypted blocks associated with the file. This happens during adding and removing a file as this causes submitting the encrypted blocks or removing the encrypted blocks of a file from the cloud storage servers.

7 Performance evaluation

Experiments were performed to analyze the performance of the proposed system. The evaluation of the proposed MS-SDC system is based on the upload time and the search time for files using keywords. When computing the file upload time, we used simulation of multiple servers on a single local machine to introduce an accurate time comparison. Real uploading on the cloud will suffer from instability of the Internet connection and server's current workload, thus will not be abstracted. However, we still have the system services priority on the workstation environment that may cause little inaccuracy in the uploading time computation. To overcome such problem, 10 experiments were made and their average was used as the result to get an accurate average uploading time for each file. The results are divided into two sets of experiments; the first is concerned with the file upload time while the other is concerned with the search time.

7.1 Experiment 1: effect of multithreading on file upload time

The first set of experiments were dedicated to evaluate the effect of using the multithreading approach presented in our work on the file upload time. The system was tested at different file sizes and different block sizes. The proposed MS-SDC system was tested without applying threading, as presented in (Poh et al. 2017), and with threading as proposed by our work.

We have performed several experiments on different types of files (videos, images, txt, and pdf) of different sizes at variable block sizes to determine the effect of file type, block size and using multithreading on the file upload time. Each experiment was performed 10 time on 10 files of the same type and of the same size, and the average of these times is obtained as the average upload time. Table 3 displays the results concluded from these experiments.

It is evident from the results that using multithreading approach has decreased dramatically the file upload time up to almost 64% for some files, with an average of about 56% that will give better server response and better user experience in the process. Furthermore, the file type did not affect the uploading time as the server handles the file as blocks regardless of its contents. Regarding the effect of the block size on the upload time, as the block size goes smaller, the number of file parts are generated and thus the upload time increases. However, using the multithreading approach covers this problem and thus the difference between the upload time at large block size (e.g. 4 MB) is very close to the upload time at small block size (e.g. 0.5 MB) which will in return encourage splitting the file for smaller parts to increase its security.

7.2 Experiment 2: search time

The search time for files using keywords is also evaluated, assuming that the total number of uploaded files equals to the number of files containing the search keywords. This is to avoid the effect of searching unwanted files which can deviate the results greatly. Figure 17 shows the average search time of searching the uploaded files using multiple keywords. As shown in the Fig., the search time depends on the total number of uploaded files. The search time is linearly proportional to the number of files containing the search keywords. It is clear that the system is able to produce the correct results in a short time.

7.3 Experiment 3: comparative study

In this subsection, we compare MS-SDC scheme with scheme presented in (Poh et al. 2017) in terms of their time performance. We perform the experiment on text files (.txt) only as shown in the scheme in (Poh et al. 2017) at file sizes: 1 MB and 5 MB. We divided each file into 5, 10, 15, 20, and 25 blocks. Table 4 shows the test results for 5 blocks and 25 blocks while the detailed results are displayed in Figs. 18 and 19.

From Table 4, we notice that the indexing time and uploading full document time is the same in both schemes where the effect of using the multi-threads proposed in MS-SDC reflects on the block handling time after the blocks are split.

Table 3 Average file upload time with/without threading for different file sizes

File size	Block size	Average upload time range (ms) without thread	Average upload time range (ms) with thread	Improvement %
10 KB	4 KB	68	38	44.80
	2 KB	85	47	45.10
	1 KB	138	64	53.90
	500 B	223	101	54.60
25 KB	4 KB	96	50	47.70
	2 KB	161	78	51.70
	1 KB	288	125	56.60
	500 B	525	226	56.90
50 KB	4 KB	142	62	56.50
	2 KB	416	175	57.90
	1 KB	642	267	58.30
	500 B	1123	451	59.80
100 KB	4 KB	260	110	57.80
	2 KB	629	244	61.20
	1 KB	1308	489	62.60
	500 B	2196	785	64.30
10 MB	4 MB	431	233	45.90
	2 MB	444	247	44.00
	1 MB	478	262	45.20
	500 KB	565	266	52.80
25 MB	4 MB	983	529	46.20
	2 MB	1132	562	50.30
	1 MB	1167	585	49.90
	500 KB	1900	835	56.00
50 MB	4 MB	2037	938	54.00
	2 MB	2425	1075	55.70
	1 MB	2865	1307	54.40
	500 KB	4132	1672	59.60
100 MB	4 MB	5032	2190	56.50
	2 MB	5771	2505	56.60
	1 MB	6972	2782	60.10
	500 KB	7953	2804	64.70

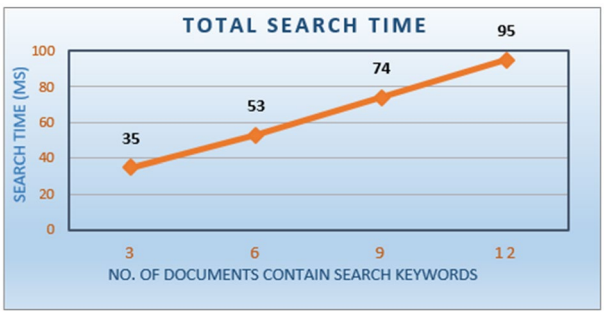

Fig. 17 The average search time

The results of scheme (Poh et al. 2017) illustrated in Fig. 18 were taken from their paper. From Fig. 18, it is axiomatic that the upload time is increasing linearly with the number of blocks, but we notice that there was a drop in the result presented in (Poh et al. 2017). The result may be reported after applying the scheme (Poh et al. 2017) as a single run. Therefore, we applied their scheme (Poh et al. 2017) 10 times and obtained the average uploading time to get an accurate comparative study between scheme (Poh et al. 2017) and our MS-SDC scheme as illustrated in Table 5. It is evident from the results that the MS-SDC scheme decreased the uploading time up to almost 64% less than the scheme (Poh et al. 2017).

Table 4 Performance data for 5 blocks and (25 blocks) in ms

	1 MB (1mb.txt)		5 MB (5mb.txt)	
	Scheme (Poh et al. 2017)	MS-SDC	Scheme (Poh et al. 2017)	MS-SDC
Indexing	26 (45)		99 (117)	
Uploading blocks	234 (465)	90 (182)	264 (679)	144 (254)
Improvement %	60.90–61.50%		45.50–62.60%	
Uploading full document	168		240	

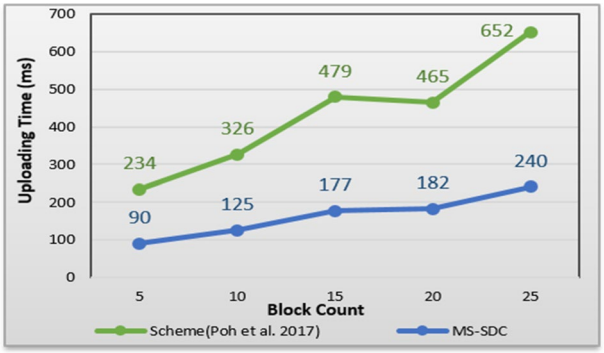

Fig. 18 Uploading time at different block count for 1 MB text file

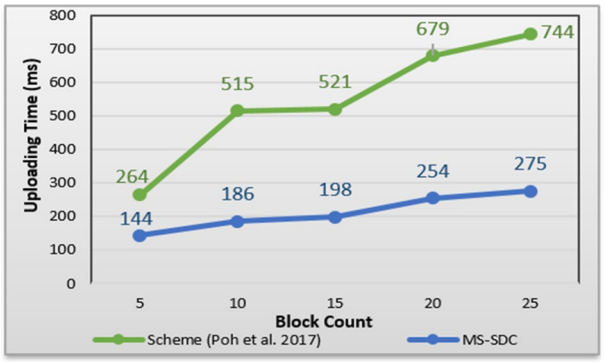

Fig. 19 Uploading time at different block count for 5 MB text file

8 Conclusion and future work

In this paper, a multi-server SSE system prototype, called MS-SDC, has been developed and implemented. The scheme presents solutions for privacy and searchability problems while maintaining minimal leakage at which there is no single storage provider will have the whole document. Besides the usage of multi-server storage providers, the master keys are created randomly for each uploaded document. Furthermore, the block size and number of servers are configurable parameters and not static to assure system scalability. The MS-SDC prototype is unique in its smooth usage while it is robust where it can run on any browser and can be applied to any type of files. By using the search engine and in a secure manner, the system provides keyword(s) searchability for the users to browse across their uploaded encrypted documents without decryption. Furthermore, it prevents servers from dealing directly with the query keywords to hide any data access patterns.

Table 5 Average file upload time for scheme (Poh et al. 2017) and MS-SDC scheme that applied on different file sizes

File size	Block count	Scheme (Poh et al. 2017)	MS-SDC	Improvement %
1 MB (1mb.txt)	5	222	90	59.50
	10	311	125	59.80
	15	441	177	59.90
	20	458	182	60.30
	25	644	240	62.70
5 MB (5mb.txt)	5	254	144	43.50
	10	413	186	54.90
	15	471	198	58.00
	20	625	254	59.40
	25	762	275	63.90

The MS-SDC system can be developed to have authorization levels for users at which each user has a limitation to access/modify into specific documents that have been previously uploaded by this user, not all uploaded documents. Besides the ability for adding or revoking any users by data owner easily. The MS-SDC system can perform all its operations on all types of documents except the keywords extraction. It has problems with some types of documents. So, our system can be developed to read all documents content types to extract its keywords such as pdf files and others that already encrypted which prevents read or edit operations. The MS-SDC search engine performs a search by ranked matching coordinate only, it can be developed to expand its scope by adding a search by meaning, similarity, and so on.

References

Bost R, Fouque PA (2019) Security-efficiency tradeoffs in searchable encryption. Proc Int Conf Priv Enhanc Technol 2019(4):132–151

Bösch C, Hartel P, Jonker W, Peter A (2014) A survey of provably secure searchable encryption. ACM Comput Surv (CSUR) 47(2):1–51

Cao N, Wang C, Li M, Ren K, Lou W (2013) Privacy-preserving multi-keyword ranked search over encrypted cloud data. IEEE Trans Parallel Distrib Syst 25(1):222–233

Cash D, Jaeger J, Jarecki S, Jutla CS, Krawczyk H, Rosu MC, Steiner M (2014) Dynamic searchable encryption in very-large databases: data structures and implementation. NDSS 14:23–26

Cash D, Grubbs P, Perry J, Ristenpart T (2015) Leakage-abuse attacks against searchable encryption. In: Proceedings of the 22nd ACM SIGSAC conference on Computer and Communications Security, pp. 668–679

Cash D, Jarecki S, Jutla C, Krawczyk H, Roşu MC, Steiner M (2013) Highly-scalable searchable symmetric encryption with support for boolean queries. In: Proceedings of the international Annual Cryptology, Springer, Berlin, Heidelberg, pp. 353–373

Chang YC, Mitzenmacher M (2005) Privacy preserving keyword searches on remote encrypted data. In: Proceedings of the international conference on Applied Cryptography and Network Security, Springer, Berlin, Heidelberg, pp. 442–455

Chase M, Kamara S (2010) Structured encryption and controlled disclosure. In: Proceedings of the international conference on the Theory and Application of Cryptology and Information Security, Springer, Berlin, Heidelberg, pp. 577–594

Curtmola R, Garay J, Kamara S, Ostrovsky R (2011) Searchable symmetric encryption: improved definitions and efficient constructions. J Comput Secur 19(5):895–934

Faber S, Jarecki S, Krawczyk H, Nguyen Q, Rosu M, Steiner M (2015) Rich queries on encrypted data: Beyond exact matches. In: Proceedings of the international European symposium on Research in Computer Security, Springer, Cham, pp. 123–145

Goh EJ (2003) Secure indexes. IACR J Cryptol ePrint Arch 2003:216

Golle P, Staddon J, Waters B (2004) Secure conjunctive keyword search over encrypted data. In: Proceedings of the international conference on Applied Cryptography and Network Security, Springer, Berlin, Heidelberg, pp. 31–45

Hahn F, Kerschbaum F (2014) Searchable encryption with secure and efficient updates. In: Proceedings of the ACM international conference on Conference on Computer and Communications Security, pp. 310–320

Hoang T, Yavuz AA, Durak FB, Guajardo J (2019) A multi-server oblivious dynamic searchable encryption framework. Journal of Computer Security 27(6):649–676

Hoang T, Yavuz AA, Durak FB, Guajardo J (2018) Oblivious dynamic searchable encryption on distributed cloud systems. In: Proceedings of the FIP annual conference on Data and Applications Security and Privacy, Springer, Cham, pp. 113–130

Ishai Y, Kushilevitz E, Lu S, Ostrovsky R (2016) Private large-scale databases with distributed searchable symmetric encryption. In: Proceedings of the Cryptographers' Track at the RSA conference, Springer, Cham, pp. 90–107

Kamara S, Papamanthou C, Roeder T (2012) Dynamic searchable symmetric encryption. In: Proceedings of the ACM international conference on Computer and Communications Security, pp. 965–976

Kim IT, Quan TH, Duc LV, Nguyen TK, Hwang SO (2018) An efficient searchable encryption scheme in the multi-user environment. In: Proceedings of the international conference on Green and Human Information Technology, Springer, Singapore, pp. 188–192

Knopf C (2007) Cryptographic Hash Functions. Leibniz Universität Hannover

Kuzu M, Islam MS, Kantarcioglu M (2012) Efficient similarity search over encrypted data. In: Proceedings of the IEEE 28th international conference on Data Engineering, pp. 1156–1167

Lipmaa H, Wagner D, Rogaway P (2000) Comments to NIST concerning AES modes of operation: CTR-mode encryption.

Meharwade A, Patil GA (2016) Efficient keyword search over encrypted cloud data. Procedia Comput Sci 78(3):139–144

Mohamad MS, Tan SY, Chin JJ (2019) Searchable symmetric encryption security definitions. Malays J Math Sci 13:31–47

Naveed M, Prabhakaran M, Gunter CA (2014) Dynamic searchable encryption via blind storage. In: Proceedings of the IEEE international symposium on Security and Privacy, pp. 639–654

Peter A, Leenders B, Lim HW, Tang Q, Wang H, Hartel P, Jonker W (2014) Distributed searchable symmetric encryption. In: Proceedings of the international conference on Privacy, Security and Trust

Poh GS, Baskaran VM, Chin JJ, Mohamad MS, Lee KW, Maniam D, Z'aba MR (2017) Searchable data vault: encrypted queries in secure distributed cloud storage. Algorithms 10(2):52

Poh GS, Mohamad MS, Chin JJ (2018) Searchable symmetric encryption over multiple servers. Cryptogr Commun 10(1):139–158

Song DX, Wagner D, Perrig A (2000) Practical techniques for searches on encrypted data. In: Proceedings of the IEEE international symposium on Security and Privacy, pp. 44–55

Stefanov E, Papamanthou C, Shi E (2014) Practical dynamic searchable encryption with small leakage. In: Proceedings of the international symposium on 21st Annual Network and Distributed System Security – NDSS, The Internet Society, 71:72–75

Stefanov E, Van Dijk M, Shi E, Fletcher C, Ren L, Yu X, Devadas S (2013) Path ORAM: an extremely simple oblivious RAM protocol. In: Proceedings of the ACM international conference on Computer and Communications Security, pp. 299–310

Van Liesdonk P, Sedghi S, Doumen J, Hartel P, Jonker W (2010) Computationally efficient searchable symmetric encryption. In: Proceedings of the international workshop on Secure Data Management, Springer, Berlin, Heidelberg, pp. 87–100

Wang Y, Wang J, Chen X (2016) Secure searchable encryption: a survey. J Commun Inf Netw 1(4):52–65

Wang C, Cao N, Li J, Ren K, Lou W (2010) Secure ranked keyword search over encrypted cloud data. In: Proceedings of the IEEE 30th international conference on Distributed Computing Systems, pp. 253–262.

Wright CV, Pouliot D (2017) Early detection and analysis of leakage abuse vulnerabilities. IACR J Cryptol ePrint Arch 2017:1052

Wu CF, Ti YW, Kuo SY, Yu CM (2019) Benchmarking dynamic searchable symmetric encryption with search pattern hiding. In: Proceedings of the international conference on Intelligent Computing and its Emerging Applications, pp. 65–69

Zhang Y, Katz J, Papamanthou C (2016) All your queries are belong to us: the power of file-injection attacks on searchable encryption. In: Proceedings of the 25th {USENIX} Security Symposium, pp. 707–720