Soft Computing (2020) 24:5609–5631
https://doi.org/10.1007/s00500-019-04033-z

FOCUS

A privacy-preserving multi-keyword search approach in cloud computing

Ahmed M. Manasrah[1,2] · Mahmoud Abu Nasir[2] · Maher Salem[1]

Published online: 8 May 2019

Abstract
Cloud computing provides the users with the ability to outsource their data to a third-party cloud storage for cost-effective management of resources and on-demand network access. However, outsourcing the data to a third-party location may raise concerns about data privacy. To maintain the user's privacy, users tend to encrypt their sensitive data before outsourcing it. Encrypting the data will preserve its privacy, but at the same time, it makes the searching process for a specific keyword a time-consuming and challenging process, mainly if the encryption key is not provided. On the other hand, the data owner should be able to perform multiple keyword searches to retrieve specific documents that are relevant to the search query. This paper proposes a new privacy-preserving multi-keyword search approach for the cloud outsourced data. The objective of the proposed approach is to allow the data owners and the authorized users to retrieve the most relevant data with minimum computation and communication overhead, and reduced false positives (irrelevant documents) and searching time. To evaluate the proposed approach, the NSF research dataset is used. Results demonstrate that the proposed method achieves better searching time and overall performance of the cloud environment regarding computation and communication overhead as well as false positives in comparison with other approaches.

Keywords Privacy preserving · Data privacy · Cloud privacy · Data encryption · Multi-keyword search · Searchable indexing schemes

1 Introduction

Cloud computing became the platform of choice for users looking for storage infrastructure with minimal cost and administration efforts. In such an environment, users (i.e., data owners) can easily increase their subscribed storage space without the need to get extra local storage devices (Plageras et al. 2018). Cloud environment, therefore, provides a cost-effective resource utilization with on-demand network access anytime anywhere. Despite the different advantageous benefits of the cloud, users still have serious concerns regarding their outsourced data security in the cloud (Stergiou et al. 2018). Such concerns include exposing outsourced data to malicious attacks, the data integrity of the outsourced data as well as the privacy of the data itself (Li et al. 2017; Ranjan et al. 2017).

Privacy-preserving concerns are one of the main issues that hinder cloud computing adoption by users and organizations because outsourcing the data to the cloud service provider (CSP) deprives the data owners (DO) of directly controlling their data and applications. Therefore, CSPs provide their customers with various infrastructure capabilities with a high degree of transparency into their operations to ensure the preservation of their data privacy and its security. However, CSPs usually provide different mechanisms to suite the customers' needs such as auditing and history-based access control (Nedjah et al. 2017). Likewise, some mechanisms are also offered by the CSPs to control and optimize the use of the resources through

Communicated by B. B. Gupta.

✉ Ahmed M. Manasrah
amanasrah@hct.ac.ae; ahmad.a@yu.edu.jo

Mahmoud Abu Nasir
mahmud.s@yu.edu.jo

Maher Salem
msalem1@hct.ac.ae

[1] Computer Information Sciences, Higher Colleges of Technology, Sharjah, United Arab Emirates

[2] Computer Science Department, Information Technology and Computer Sciences, Yarmouk University, Irbid, Jordan

resource allocation and load balancing mechanisms (Aljammal et al. 2017; Manasrah 2017). Nevertheless, balancing between these mechanisms and the preservation of the cloud data privacy is still a significant challenge where physical perimeters are virtual (Takabi 2014). However, researchers are also convinced that data's sensitivity and privacy can be preserved by using strong encryption mechanisms before outsourcing the data to the cloud. While data encryption offers protection against malicious attacks and illegal accesses, it increases the computation and the communication overhead at the DOs side, especially with users having large data.

Consequently, increasing the overhead and the computation time will also increase the cost of the service being used (Manasrah et al. 2016). On the other hand, the DO will lose the ability to search his encrypted data for specific keywords to retrieve the most related documents to the search query. Encrypting the data before outsourcing them to the cloud will raise a new key management concern, as this key has to be managed by the DO or the CSP. Nevertheless, if the key management is delegated to the CSP, the problem will become even more complicated.

This paper contributes a privacy-preserving multi-keyword search approach for the cloud outsourced data. The computation of the encryption and decryption of the data before outsourcing is reduced through adopting a probabilistic public key encryption. Moreover, the proposed approach also reduces the searching time and increases its accuracy while maintaining the privacy of the retrieved data through a document's topic clustering summary using a summarization technique. The multi-keyword search capability is another feature to increase the search accuracy and retrieve the most relevant files to the user that fulfills the search criteria. To the best of our knowledge, this is the first approach that attempts to address the problem of privacy preserving using a clustering technique based on document topics and summary.

The paper is organized as follows. Section 2 describes the related work. Section 3 gives the preliminaries and the main notations. Section 4 presents the privacy-preserving multi-keyword search approach. In Sect. 5, we analyze the security of the schemes. Section 6 carries on the experiment, compared with the existing schemes regarding the search results accuracy and efficiency. Section 7 concludes the work and presents future directions.

2 Related work

The research in the area of privacy preserving over cloud environment is mainly classified into encryption-based privacy-preserving approaches in which researchers try to achieve the security of the encrypted files using various encryption-based techniques. Other groups of researches focus on indexing and searching algorithms because encrypting the data before outsourcing is the only acceptable way to secure the files while residing outside the data owner's premises. However, the question of how to search these encrypted files at the cloud without decrypting them with minimum computations is challenging. The DO first encrypts his data before outsourcing it to CSP and later applies the keyword search or a ranked keyword search to retrieve them. These searchable encryption schemes can be grouped as symmetric key encryption (Chang and Mitzenmacher 2005; Chase and Kamara 2010; Curtmola et al. 2006), fuzzy-searchable encryption (Adjedj et al. 2009; Wang et al. 2012) and public key encryption (Bellare et al. 2007; Boneh et al. 2004; Manasrah and Al-Din 2016). However, these techniques require a considerable number of comparisons to retrieve the results. The required number of comparisons increases the time and computation complexity and hence the overall cost (Pasupuleti et al. 2016).

Clustering similar documents together based on standard features will speed up the searching process over encrypted documents set (Krishna and Handa 2016). In this regard, Zhu et al. (2015) proposed a hierarchical clustering algorithm using fuzzy logic and swarm intelligence (HCSF) to solve the ciphertext search complexity issues. The generated clusters in this schema are created using swarm intelligence to provide the proper choices for the initial cluster centers. The fuzzy logic is used for addressing ambiguous documents during the index establishment. The authors also define the search process as the following sequence: (1) index formation during which the clusters and the sub-cluster are generated using the k-means algorithm to provide proper initial cluster center values. However, to improve the search efficiency, the hierarchy algorithm is used with the clustering process. (2) Encryption of document and clusters center values. (3) Trapdoor generation allows users to generate a query vector. (4) And finally, the search process in which the cloud server receives the trapdoor and chooses the cluster center and the sub-cluster center and the documents based on the similarity with the search query. This schema is efficient because the cloud server does not need to scan all the documents. However, it does not support dynamic updates (i.e., insertion, modification and deletion of data). Similarly, Handa and Challa (2015) proposed a cluster-based searching schema. This schema reduces the number of comparisons and the time required to perform the searching and achieves the security requirements. This search schema includes the following steps: (1) cluster generation to generate multiple clusters depending on the similarity between the documents; (2) document index generation to generate the index for each document; (3) cluster index generation to generate the clustered index using the bitwise product of the indices of all the documents within the cluster; (4) document encryption encrypts all documents; (5) query index generation to calculate the HMAC for each search term in the search query; (6) document searching in which the CSP will select the appropriate cluster and select the document in this cluster; (7) document decryption in which the DO will send the secret key to the end user; then the user decrypts the document. This schema is efficient, but it doesn't support the dynamic updates.

To overcome and solve the dynamic updates issues, Krishna and Handa (2016) proposed a searching schema based on dynamic clustering. This schema enables dynamic updates by generating the index dynamically. This schema is efficient and reduces the cost of the index generation. Their work is an extension to work presented in Handa and Challa (2015). This search schema includes the following steps: (1) cluster generation to generate the clusters based on the similarity between keywords; (2) cluster index generation to generate the clustered index by calculating the hash-based message authentication code (HMAC) for each keyword in the cluster; (3) document index generation which is similar to the cluster index generation; (4) document encryption to encrypt the documents; (5) query index generation in which the authorized users (AU) calculate the HMAC value for all search query items and generate search index using the bitwise product for each term index and finally send this query index to the CSP; (6) document searching, when the CSP receives the searching query, compares it with the clustered index to select the matching cluster and then compares it with document index to select the matching documents and retrieve it; (7) document decryption, when the user retrieves the documents, he will request the credentials from DO to decrypt the documents. However, the proposed schema never evaluated over big data sets. Therefore, Chen et al. (2016) proposed a schema to search over encrypted data using hierarchical clustering for a big data environment. In this schema, the documents are grouped into sub-clusters (i.e., like a tree and subtrees), and each document is represented as a point in the vector space to reflect the relevance of the corresponding documents to the normalized length of vectors. The proposed schema delivers good efficiency in terms of searching time. Nevertheless, it needs to store the high-dimensional vectors at the cloud server to represent the clusters and the documents, which will require a substantial unnecessary storage volume in big data environments. Similarly, Xiangyang et al. (2017) presented an efficient multi-keyword text search over encrypted cloud data (MUSE) based on hierarchical clustering. In this schema, the authors introduce a novel index structure, a new tree data structure and a depth-first search algorithm to improve the efficiency of the text searching. This schema may leak the privacy of keywords because the CSP may find some high-frequency keyword usage and check whether a particular keyword exists in a document by comparing the searching requests with the searching results. Therefore, they further enhanced the MUSE by adding some phantom terms into document vectors and query vectors, and then the privacy will be protected. The performance of EMUSE schema is less than the basic MUSE because of adding some phantom terms, but they keep the privacy preserved and does not leak certain frequency on the used keywords neither the text itself.

Most of the proposed works in the field of clustering-based privacy preserving are focusing on clustering the data at the CSP. For instance, Yin et al. (2018) proposed a privacy-preserving k-means clustering algorithm for a multi-dimensional and encrypted data using the scalar-product-preserving encryption primitive at the cloud server. The primary goal of their proposed algorithm is to perform data mining over the encrypted cloud data while preserving the data privacy on behalf of users. Similarly, Jiang et al. (2018) proposed a cloud server-based privacy-preserving collaborative *k*-means clustering framework on mining encrypted cloud data for useful knowledge while keeping the privacy of both data and the mined results enacted. Despite the number of methods in privacy preserving, these methods are quite complex in terms of computation and memory usage, thus leading to limited usage of these methods especially in the case of paid cloud services. Table 1 provides a comparison and summary for some of the cluster-based privacy-preserving schemas.

3 Preliminaries and notations

In this paper, the following variables and notations are used: D is the documents collection denoted as a set of n data documents $D = \{d_1, d_2, \ldots, d_n\}$, where $d_i \in D$, $\forall i \in \{1, 2, \ldots, n\}$, W is all distinct words collection denoted as a set of m words $W = \{w_1, w_2, \ldots, w_m\}$, where $w_j \in W$, $\forall j \in \{1, 2, \ldots, m\}$, C is the total cluster collection denoted as a set of k-clusters $C = \{C_1, C_2, \ldots, C_k\}$, where $C_t \in C$, $\forall t \in \{1, 2, \ldots, k\}$, *and* V_t is the cluster center vector of C_t, $id(d_i)$ is the vector space model which represents the document identifier (id) that can help to identify the actual document d_i uniquely. I_t is the index tree, u is a node in the index I_t, and θ_t is a multinomial topic distribution. The multinomial distribution provides a formula for expanding an expression such as $(x_1 + x_2 + \cdots + x_t)^n$. *Word_Index* is the index for each word in all documents. *Cluster_Index* is the index for each cluster in C, and x_i is a pseudorandom binary which is a binary sequence that is generated with a deterministic algorithm. The distance measure between two vectors p *and* q, i.e., $E(p, q)$, is the Euclidean distance.

Table 1 Summary for some of the known cluster-based privacy-preserving schemas

Schema	Description	Dataset	Clusters #	Drawbacks
Zhu et al. (2015)	Proposed the HCSF to solve the efficiency issues in ciphertext search	7505 documents and 61,188 keywords	20	The schema doesn't support the dynamic updates
Handa and Challa (2015)	Proposed a cluster based searching schema reduces the number of comparisons and the time required	Up to 6000 documents	5	The schema doesn't support the dynamic updates
Krishna and Handa (2016)	Proposed a searching schema based on dynamic clustering	Up to 6000 documents	5	The number of clusters is static
Chen et al. (2016)	Proposed an approach to searching over encrypted data using hierarchical clustering to support a big data environment	51,000 documents and 22,000 keywords	NA	Need a huge unnecessary storage volume in big data environments
Xiangyang et al. (2017)	Proposed an efficient searching schema on cloud computing based on hierarchical clustering	The real dataset includes about 20,000 abstracts and extracts about 10,000 keywords	NA	MUSE less efficient than MRSE-HCI

Vector space model based on term frequency-inverse document frequency (TF-IDF) algorithm is very popular in information retrieval (IR) as well as insecure multi-keyword search (Ramos 2003; Salton et al. 1975). By using the vector space model, each document in D will be represented as an n-dimensional vector. All elements in $id(d_i)$ will be normalized in the final score to increase the retrieval performance.

Moreover, the following documents will be used for illustrative purposes:

Document-1 (d_1) has the following sentence {"*cloud computing means accessing resources over the Internet instead of your computer's hard drive*"}
and Document-2 (d_2) has the following sentence {"*Cloud computing is a model for enabling ubiquitous access to shared pools of configurable resources*"}.

4 The privacy-preserving multi-keyword search approach

This section presents the new privacy-preserving schema that is based on topic summary and clustering to reduce the number of comparisons required for the searching and retrieving of the outsourced encrypted data. The proposed schema is based on public key encryption along with a new ranked based multi-keyword searching approach in for cloud environment. In this approach, a model of cloud environment that consists of three entities as illustrated in Fig. 1 is considered.

The DO who will handle most of the computation before outsourcing the data to the cloud, such as creating Cluster_Index, documents index and encrypting the user's documents and their index. The CSP is an entity that provides different cloud services to the DO's and AU's such as searching over the outsourced data on their behalf. The CSP offers storage resources as a pay-per-use model. The way and the time in which a resource is occupied is used to determine the cost of the service (Manasrah and Gupta 2017). The DO shares typically some information with the AU that can be used for searching documents using a set of keywords. The documents usually are encrypted; hence, the AU is allowed to use the shared information to perform the searching process over the encrypted data before decrypting it to its original form. The interactions between the three entities are as follows:

1. The DO outsources a set of documents to the CSP in an encrypted format and still poses the ability to retrieve them back. To achieve this, the DO handles the document topics-based clustering process as well as extracting the document summary to be used in the index creation. The DO creates a Cluster_Index for each cluster by generating a summary for each cluster and sends the Cluster_Index to be used in a trapdoor generation based on the similarity with the search query. The DO also creates an index for each document and encrypts all user's documents and their indexes to be kept at the CSP.
2. The AU retrieves some documents from the CSP by communicating with the DO to request for specific information for the trapdoor generation. The similarity between the search query and the Cluster_Index identifies the matched cluster. Finally, the AU sends the search query to the CSP as a trapdoor. The CSP uses the trapdoor to apply the ranked keyword search

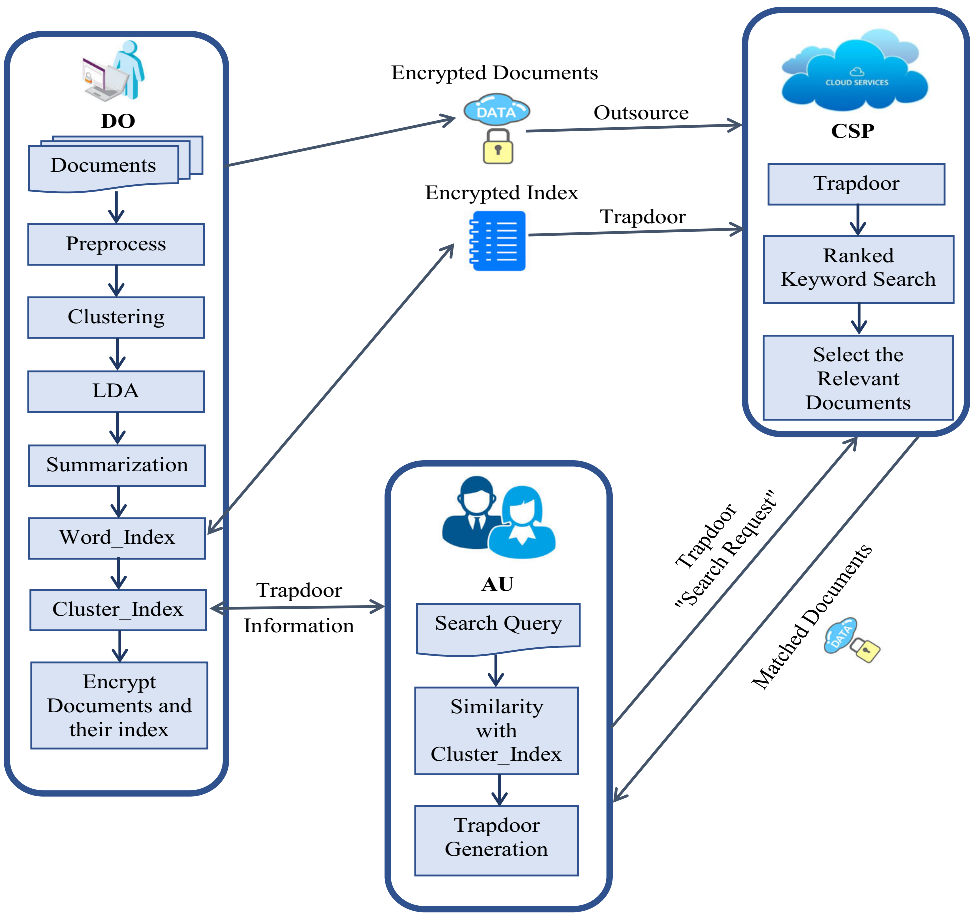


Fig. 1 The interactions between DO, AU and CSP

for the documents in the matched cluster and returns a set of encrypted and relevant documents to the AU.

3. Finally, the AU browses through the returned documents to verify their integrity before decrypting them using the private key.

The privacy-preserving multi-keyword search (PPMKS) approach uses a ranked keyword search over encrypted documents. To ensure efficiency, the CSP should return the top-k most relevant documents based on a relevance score rank to enhance the retrieval accuracy as well as minimizing the communication cost. To reduce the time and the number of comparisons that are required for the searching process, the proposed approach clusters the documents based on the document topic and summary. To ensure the preservation of the privacy, the CSP should learn neither the index nor the original plaintext. The index should not contain any information that might be used to guess the original keywords and break the encrypted data. The proposed approach consists mainly of two phases: (1) setup phase and the (2) retrieval phase.

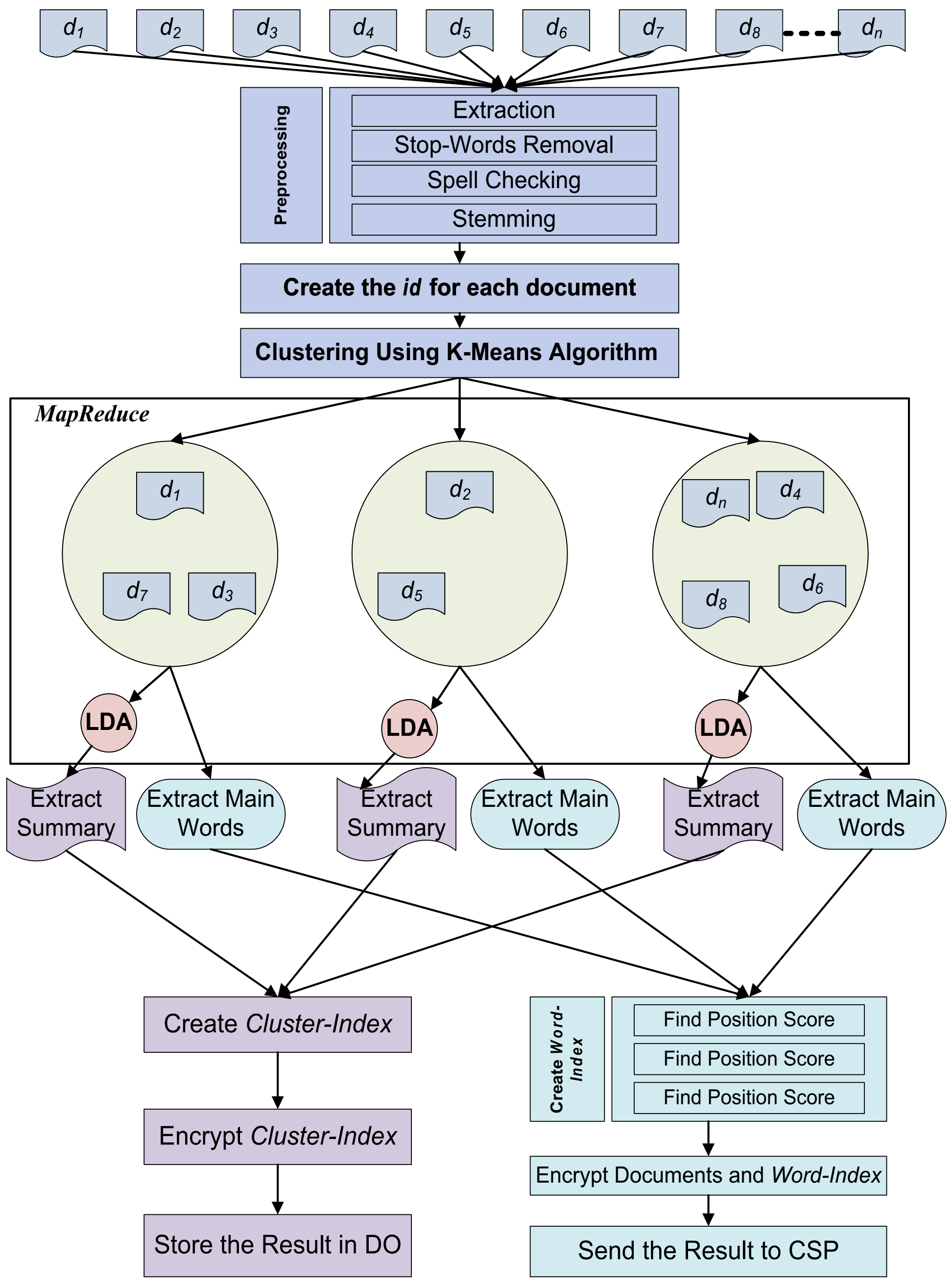
4.1 Setup phase

Upon the DO selection of the documents to be outsourced to the CSP, the DO generates a pair of public and private keys for the encryption and decryption processes. During this phase, a document index will be created from multiple keywords that are extracted from the document collections for each cluster. The relevance score is then calculated for each keyword to be kept in the document index. The index and the document collections are encrypted using the private key to ensure its privacy. Finally, the DO outsources the encrypted documents and the encrypted index to the CSP. The setup phase consists of five steps: (1) key generation, (2) data preprocessing, (3) clustering and summarization, (4) word index creation and (5) privacy preserving, as illustrated in Fig. 2.

4.1.1 Key generation

The key generation phase generates one public (PK) and one private (PR) keys using the key generation algorithm proposed in Pasupuleti et al. (2016). The DO chooses two large primes numbers (p, q) randomly of the same size to compute $N = pq$. Then it calculates r and s using the

Fig. 2 Setup phase framework



extended Euclidean algorithm (Katz et al. 2008) where $pr + qs = 1$. Thus, the public key (PK) can now be defined as $PK = \{N\}$, and the private key (PR) is $PR = \{p, q, r, s\}$. However, the documents may contain punctuations, numbers, stop-words and various suffixes. Processing such documents will increase the complexity and the computation in handling the different types of documents. Therefore, a preprocessing that aims to remove the unnecessary punctuations, numbers, stop-words, the different suffixes and creating the document words matrix to be used with the other setup steps is needed.

4.1.2 Data preprocessing

The data preprocessing phase aims to prepare or transform the raw format of the documents into an understandable format for outsourcing, using four steps; extraction, stop-words removal, spell checking and stemming.

A. Extraction

In this step, a list of individual words is extracted based on the (1) punctuation marks and (2) the white spaces. This list of words is then used to generate an index after the stop-words removal, spell checking and stemming. To further minimize the needed processing steps, the commonly used words such as *over*, *the*, *instead*, *of*, *your* should be removed to focus on the important words instead.

B. Stop-words removal

To speed up the searching process, the most common words in the English language that does not add any useful meaning to the text, such as a, an, the, with, etc., should be removed from the document. This means reducing the number of the keywords in the index where the stop-words may account for 20–30% of the total word count in a text document (Witten et al. 2016). In this work, a stop list that contains 571 words from Salton and Buckley (1988) is adopted.

C. Spell checking

The spellchecker Hunspell is used to recognize misspelled words in the documents and replaces them with the correctly spelled ones. Hunspell is a popular spellchecker tool that is widely used in popular software packages including Google Chrome, Mac operating system, Opera and InDesign (Xu et al. 2015). Hunspell has a dictionary of correct words which will be compared with all words in W.

D. Stemming

The words in W should, therefore, be stemmed to their roots. For example, the words "material, materially, materialize, materialization and materiality" can be stemmed to the word "material" (Ramasubramanian and Ramya 2013). By removing the various English suffixes, the efficiency of the searching process could be enhanced by minimizing the number of words for the matching stems. The stemmer of Porter et al. (2002) which is a set of rules to group words with similar roots will be adapted to remove the suffixes.

For example, after applying the adopted Stemming algorithm over W, the list of words W in ascending order will be:

$$W = \{access, cloud, comput, drive, hard, internet, mean, resourc\}.$$

After preprocessing all the documents, the identifier (id) for each document must be built to represent the vector space model for each document, using the following equation:

$$id(d_i) = \{w_{1,i}, w_{2,i}, \ldots w_{t,i}\} \tag{1}$$

where $w_{j,i}$ is how many times the jth word in W appears in d_i. t is the number of words in W.

For instance, if d_1 content is "*Cloud computing means accessing resources over the Internet instead of your computer's hard drive*", and d_2 *content is* "*Cloud computing is a model for enabling ubiquitous access to shared pools of configurable resources*", then the result of the preprocessing step will be as follows:

d_1 = {*cloud comput mean access resourc internet comput hard drive*}.
d_2 = {*cloud comput model enabl ubiquitou access share pool configur resourc*}.
W = {*access, cloud, comput, configur, drive, enabl, hard, internet, mean, model, pool, resourc, share, ubiquitou*}.
id(d_1) = {*1,1,2,0,1,0,1,1,1,0,0,1,0,0*}.
id(d_2) = {*1,1,1,1,0,1,0,0,0,1,1,1,1,1*}.

However, if the DO has a huge number of documents, then the searching process may require a longer time. Therefore, to speed up the searching time, this paper proposes to group the documents with similar topics or concepts (i.e., semantic clustering) together as the following section demonstrates.

4.1.3 Semantic clustering

Clustering is used to group the documents into different clusters before creating the index. Classifying the documents should reduce the number of documents for each cluster to increase the efficiency of the searching process. In this regard, the proposed approach adopts the clustering technique that is proposed by Nagwani (2015) due to its superior performance, semantic similarity and main topics extraction using latent Dirichlet allocation (LDA) for summarizing the huge documents collection (Blei et al. 2003). To perform the clustering, the DO collects all (n) documents into one data set $D = \{d_1, d_2 \ldots d_n\}$. The DO then apply the K-means clustering algorithm on D, to generate k-clusters, denoted by $C = \{C_1, C_2 \ldots C_k\}$ where $t = 1, 2 \ldots K$ and C_t is a set of similar documents with the same features based on documents similarities. That means all documents within the same cluster should be as similar as possible $C_t = \{\cup (D_i \in C_t)\}$, and the documents in different clusters should be as dissimilar as possible from another document in all other clusters.

Definition 1 Assume that $C_t = \{id(d_1), id(d_2), \ldots, id(d_i)\}$ is a cluster contains i documents where $id(d_i) \in C_t$ is the corresponding identifier vector of the document i, then if the cluster center of C_t is denoted as V_t, then we have:

$$V_t[j] = \frac{\sum_{l=1}^{i} V_l[j]}{i} \tag{2}$$

where j: 1, 2, …, m is the words index in W, and i is the number of documents in C_t. The steps for the K-means clustering are illustrated in Algorithm 1 as follows:

Algorithm (1): The K-means clustering

```
Input: k: number of clusters, D: a dataset containing n objects.
Output: A set of k clusters.
 Randomly choose k documents from D as the cluster centers.
    Repeat
      (re)assign the document vector to the closest cluster center as follows
      $E(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$
    Recalculate the cluster centers
    Until the centroids no longer change
 End
```

Unfortunately, the *k*-means algorithm distance computations require (*nk*) calculations in each iteration, where *n* is the number of all documents and *k* is the number of clusters. Therefore, to address the issue of the extensive computations, especially with huge datasets, the parallel *K*-means clustering (PKMeans) based on MapReduce is also used (Zhao et al. 2009). The Map function is shown in Algorithm 2.

The Combine function will perform the procedure of combining the intermediate data of the same Map function. The Combine algorithm is illustrated in Algorithm 3.

Algorithm (2): The Map Algorithm for PKMeans Clustering

```
Input: Clusters Centers, <key, value> pair
Output: <key', value'> pair, where the key' is the identifier of the closest cluster center and the
value' is the vector of the content of the closest center
 minDis = Double.MAX VALUE;
 index = -1;
        For i=0 to centers.length do
            dis= ComputeDist(value, centers(Adjedj et al.));
            If dis < minDis {
            minDis = dis;
            index = i;}
        End For
 Take the index as a key';
 Construct value' as a vector comprise of the values of different dimensions;
 output < key', value'> pair;
 End
```

Algorithm (3): The Combine Algorithm for PKMeans Clustering

```
Input: the key is the identifier of the cluster, V is the list of the documents assigned to the same cluster.
Output: <key', value'> pair, where the key' is the identifier of the cluster, and the value' is the vector comprised of the sum of the documents in the same cluster and the documents number.
  Initialize one array to record the sum of the value of each dimension of the documents contained in the same cluster.
  Initialize a counter num as 0 to record the sum of document number in the same cluster.
      For each document in V
          Construct the document instance from V.next()
          Add the values of different dimensions of the instance to the array
          num++
      End For
  Take key as a key'
  Construct value' as a vector comprised of the sum values of different dimensions and num
  output <key', value'> pairs
  End
```

The Reduce function, on the other hand, will perform the procedure of updating the new centers. The Reduce algorithm is illustrated in Algorithm 4.

Algorithm (4): The Reduce Algorithm for PKMeans Clustering

```
Input: the key is the identifier of the cluster, V is the list of the partial sums from the different core.
Output: <key', the value'> where key' is the cluster identifier and value' is the new center vector.
  Record the sum of the value of each dimension of the documents within the same cluster.
  Initialize a counter NUM as 0 to record the sum of documents number in the same cluster.
      For each document in V
          Construct the document instance from V.next()
          Add the values of different dimensions of the instance to the array
          NUM += num
      End For
  Divide the entries of the array by NUM to get the new center's coordinates
  Take key as a key'
  Construct value' as a vector comprise of the center's coordinates
  output <key', value'> pairs
  End
```

After the clustering phase is done, the clusters topics have to be extracted to represent the main information in the documents collection to organize and summarize the collection of documents in the same cluster.

4.1.4 Topic modeling

Topic modeling is the process of finding the main words (i.e., keywords) from a collection of documents. Many algorithms were used to create topic models such as the LDA modeling which was proposed in Blei et al. (2003) to extract the main words from a collection of documents. The LDA modeling is adopted in this work to generate topics and main terms for each cluster in $C = \{C_1, C_2 \ldots C_k\}$. These terms will be used for the summary extraction of each cluster. The LDA operates as follows:

1. Select a multinomial θ_t as the topic distribution for each topic t from a Dirichlet distribution with parameter β as the parameter of the Dirichlet prior on the per-topic word distribution.
2. For each document d, a multinomial document distribution θ_d is selected from a Dirichlet distribution with

parameter α as the parameter of the Dirichlet prior on the per-document topic distributions.

3. For each word w_i in document d_i, a topic t from θ_d is selected.
4. A word w_i from θ_t is selected to represent the topic for the document.

The probability of generating a corpus to be used is given by the following equation (Blei et al. 2003):

$$\iint \prod_{t=1}^{k} P(\theta_p|\beta) \prod_{b=1}^{N} P(\theta_p|\alpha) \left(\prod_{t=1}^{Nb} \sum_{b=1}^{K} P(t_i|\theta) P(w_i|t,\emptyset) \right) \mathrm{d}\theta \mathrm{d}\emptyset \tag{3}$$

where K is the number of topics, θ_p is the topic distribution for document p, i.e., multi-dimension vector, β is the parameter of the Dirichlet prior on the per-topic word distribution, N is the total number of words in all documents, α is the parameter of the Dirichlet prior on the per-document topic distributions, t_i is the topic for the ith word in a document, θ is the topic distribution for a document, w_i is the ith word in a document, and $\emptyset$ is the word distribution for a topic.

LDA is used to generate topics and to extract the main terms for each cluster. The extracted sentences from each document represent the summary for each cluster which will be used to generate the index for each cluster. However, to deal with the possible huge amount of document within each cluster, the MapReduce is used with the LDA creation to reduce the expensive computations across clusters and to ensure that all values associated with the same key are brought together in the reducer as illustrated in Algorithm 5.

Algorithm (5): The Map Algorithm for LDA

```
Input: List of Clusters C, List of Documents D
Output: the list of topics for cluster $C_t$
 For each cluster $C_t \in \{C_1, C_2, ..., C_k\}$.
    Extract the documents in $C_t$ as $\{d_{t1}, d_{t2}, ..., d_{tn}\}$
    For each document, merge the text from the text collection.
        | Text = Text $\cup$ {$Text_{tj} \in d_{tj}$}
        | Apply LDA topic modeling to these collections and get the list of topics for the
        | cluster $C_t$ as $T_t = \{T_{t1}, T_{t2}, ..., T_{tk}\}$
    End For
 End For
```

The Reduce function, on the other hand, updates the parameters that are associated with each topic. It requires an aggregation over all intermediate $\emptyset$ vectors. The Reduce algorithm is illustrated in Algorithm 6.

Algorithm (6): The Reduce Algorithm for LDA

```
Input: List of Clusters C, List of topics for the cluster $C_t$
Output: List of Topics for all Clusters
   Integrate the topics of all clusters
  For each cluster $C_t \in \{C_1, C_2, ..., C_k\}$.
     Extract the topics discovered by LDA in the documents in $C_t$ as $T_t = \{T_{t1}, T_{t2}, ..., T_{tk}\}$
       | For each document extract the text and compute the text collection.
       |        Topics = Topics $\cup$ {$T_{tj} \in T_t$}
       | End For
 End For
```

The result of the LDA process is the list of the main words for each cluster which should represent the main information in all documents within the same cluster.

Definition 2 Assumes that $W_t = \{w_1, w_2, \ldots, w_j\}$ is a distinct word in cluster C_t where $w_j \in W$, if w_j is a representative word of C_t, then w_j will be added to the *Main_Words* list of C_t which has the following structure:

$$Main\_Words\ (C_t) = \{w_j, w_j \in W\ and\ w_j\ is\ main\ word\} \quad (4)$$

Then the *Main_Words* (C_t) are used to create the *Cluster_Index* for the cluster (C_t). The process of creating the *Cluster_Index* is to find all sentences in the cluster (C_t) which contains any word in *Main_Words* (C_t).

Algorithm (7): Word Index Creation Algorithm

```
Input: List of the Clusters C, the list of Documents D, the list of Words W
Output: List of Index for all Words
  For each cluster in C = {$C_1$, $C_2$, ..., $C_k$}
      Scan $C_t$ (a cluster in C)
      Extracts its distinct words $W_t$ = {$w_1$, $w_2$,.... , $w_m$}
    For each word in W = {$w_1$, $w_2$, ...., $w_m$ } do
      For each Document in $C_t$ in D = {$d_1$, $d_2$, ..., $d_n$} do
          Computes $PS_i = \frac{1}{\sqrt{k}}$          // Position Score
          Computes $LS_i = \frac{|Ws_i|}{\sum_{i=1}^{N}|Ws_i|}$ // The Length Score
          Computes $RS_{wi} = \frac{1}{|f_d|}.(1 + Inf_{d.t})$ // The Relevance Score
          Computes $I(w_i) = PS_i + LS_i + RS_{wi}$ //Index of Word $w_i$
      End For
    End For
  End For
```

Definition 3 Assumes that S_t = {S_1, S_2, …, S_i} is all sentences in C_t, if S_i contains any $w_j \in$ *Main_Words* of C_t, then S_i will be added to the *Cluster_Index for* C_t in the following form:

$$Cluster\_Index\ (C_t) = \{S_q, S_q \in d_i\ and\ S_q \in w_j\} \quad (5)$$

where S_q is a sentence in d_i and w_j is a representative word of C_t.

The ranked keyword search proposed in Cao et al. (2014) is adopted to enable the CSP to return the most topmost relevant documents to reduce the network traffic and to support multiple keywords search.

4.1.5 Word index creation

After generating the clusters, the DO creates an index for each cluster using LDA and creates another index for the documents collection within each cluster. The clustered index acts as the summary for each cluster. As discussed in Sect. 2.4 various indexing techniques are existed in the literature. In this paper, the indexing technique proposed by Pasupuleti et al. (2016) is adopted and enhanced with different features such as the document length and the keyword location and frequency. The proposed indexing technique is executed using a ranking function to evaluate the relevance score. The *Word_Index* consists of three scores: position score, length score and the relevance score. The details of the word index creation steps are illustrated in Algorithm 7 and works as follows:

The DO scans cluster C_t and extracts its distinct words W_t = {w_1, w_2, …, w_m}. For each document d_i in cluster C_t over each Word w_j, the following scores are computed:

1. **Position Score (PS)** The location of the words indicates there importance, as the most important word appears first (i.e., the words in the title should have higher weights than those in the abstract and the text), the location score (Sarkar 2012) of word w_j is calculated using Eq. (6).

$$PS_i = \frac{1}{\sqrt{k}} \quad (6)$$

where PS_i is the location score of the word w_j in a document d_i. k is the location of the word from the top of the document.

2. **Length Score (LS)** Long documents usually contain more important information. The length score is calculated using Eq. (7).

$$LS_i = \frac{|Ws_i|}{\sum_{i=1}^{N}|Ws_i|} \quad (7)$$

where LS_i is the length score of the document. Ws_i is the set of words in the document. N is the number of documents in a cluster.

3. **The Relevance Score (RS)** the information retrieval (IR) community uses the term frequency (TF) × inverse document frequency (IDF) to compute the RS. TF is simply the number of times a given word appears within a document. IDF is obtained for each cluster by dividing the number of documents in this cluster by the number of documents containing the word within the same cluster. The relevance score (Wang et al. 2012) is calculated using Eq. (8).

$$RS_{wj} = \frac{1}{|f_d|}.(1 + \text{In} f_{d.t}) \tag{8}$$

where RS_{wi} is the relevance score for the word w_j in the document d_i. $|F_d|$ is the length of the document, and $f_{d,t}$ denotes the word frequency in document d_i.

Finally, to generate a *Word_Index* for w_i, all previous scores are calculated as follows:

$$I(w_j) = PS_i + LS_i + RS_{wi} \tag{9}$$

where $I(w_i)$ is an index for the word (w_i) in the document.

The result of the word index creation algorithm is an index for each word w_j in all documents (D). The structure of the *Word_Index* is illustrated in Fig. 3.

The data usually have to be encrypted before outsourcing to the cloud to ensure its privacy. Therefore, the *Word_Index* and all documents must be encrypted before outsourcing to the CSP, and the *Cluster_Index* must be encrypted before sharing with AUs.

Cluster #	*Document #*	*Word #*	*I(w_i)*

Fig. 3 *Word_Index* structure

4.1.6 Privacy preserving

The DO encrypts both the document index and the document collection to preserve the privacy using the probabilistic public key encryption technique that is based on the Rabin cryptosystem (Gupta et al. 2016) and proposed in Pasupuleti et al. (2016). This technique should support not only keyword search over the encrypted data but also offer high-security characteristics. The procedure of the encryption process is illustrated in Algorithm 8.

Algorithm (8): The Encryption Algorithm

```
Input: list of the Documents D, the Words_Index I
Output: list of the encrypted Documents D', the encrypted File Index I'
    The DO authentic public key N
    Let Document d = {m_i} 1 ≤ i ≤ n
    Select r as a seed
    $x = r^2$ mod N
        For each i from 1 to n do
            $x_i = x_{i-1}^2$ mod N        // The pseudorandom sequence bits
            $p_i = x$ mod 2              // The Least Significant Bits
            Compute $I'(W_i) = p_i \oplus I(W_i)$
            Compute $c_i = p_i \oplus d_i$
        End for
    Compute $x_{n+1} = x_n^2$ mod N
    End
```

1. Let the documents $D = \{d_1, \ldots, d_n\}$ with length n, where each m_i is a binary string of length h and index $I(w_i)$.
2. Select the random seed r and generate
$$x = r^2 \mod N \tag{10}$$
where r is the random seed and N is the public key.
3. Generate the pseudorandom binary bit x_i
$$x_i = x_{i-1}^2 \mod N \tag{11}$$
$$p_i = x \mod 2 \tag{12}$$
where p_i is the least significant bits (LSB) of x_i. LSB is the rightmost bit position in a binary that determines whether the number is even or odd.
4. The pseudorandom bit sequence p_i XORed with the binary representation of *Word_Index* to get $I'(W_i)$ and the plaintext of document d_i to get the ciphertext c_i;

$$I'(W_i) = p_i \oplus I(W_i) \quad (13)$$

and

$$c_i = p_i \oplus d_i \quad (14)$$

5. Finally, generate the next random

$$x_{n+1} = x_n^2 \bmod N \quad (15)$$

4.2.1 Trapdoor generation

If the DO or the AU wants to retrieve the documents with certain keywords, they should compute the trapdoor for keywords $w_i \in W$ using the bit sequence x_{n+1} and the *Cluster_Index* to be sent to the CSP as a search request as Algorithm 9 illustrates.

Algorithm (9): The Trapdoor Generation Algorithm

```
Input: Cluster Index, the Search Query
Output: trapdoor ($T_{w_i}$ , index-for-specific-cluster, k)
    For each term j in a search query I do
        Compute $W_{i,j} = tf_{i,j} * log\frac{n}{df_j}$   // The weight W of term j in a search query i.
        Compute $Sim_{j,k} = \frac{\sum_{i=1}^{t} w_{i,j} * w_{i,k}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} * \sqrt{\sum_{i=1}^{t} w_{i,k}^2}}$   // to determine index- for- the- specific- cluster
    End For
  User authenticate key r
  Compute Trapdoor: $T_{w_i} = \sum_{i=1}^{m} H(w_i)^r$  // is an encrypted form for each word
                                                  in the search query w
  Send trapdoor ($T_{w_i}$ , index-for-specific-cluster, k) to the CSP.
  End
```

The DO sends the encrypted documents collection and the encrypted *Word_Index* $D' = \{d_1', d_2', \ldots, d_n', I'(w_i)\}$ to the CSP. The resulting bit sequence x_{n+1} and the *Cluster_Index* will be kept at the DO who will share it with the AUs to retrieve the needed documents with certain keywords search.

4.2 Retrieval phase

In this phase, if the DO or the AU wants to retrieve certain documents with certain keywords, they should start by generating a trapdoor for the set of keywords using the bit sequence x_{n+1} and the *Cluster_Index*. The CSP searches for the matched documents in the same cluster based on their corresponding relevance scores using the trapdoor. If the keywords match with the *Word_Index*, the CSP ranks the matched documents based on the relevance score and sends the top-*k* most relevant documents back to the DO or the AU in a ranked descending ordered. Then DO or AU decrypts the document using their private key. This phase consists of three processes: (1) trapdoor generation (2) ranked keyword search and (3) data decryption.

1. The AU will get the trapdoor information that contains the bit sequence x_{n+1} and the *Cluster_Index* from the DO. The bit sequence x_{n+1} will be used to encrypt documents, where the *Cluster_Index* will be used to determine to which cluster the search query belongs.
2. Using the cosine similarity proposed by Salton and Buckley (1988), the AU computes the similarity between the search query and *Cluster_Index* to determine which cluster contains this search query (to determine an *index-for-the-specific-cluster*) as in Eq. (16).

$$Sim_{j,k} = \frac{\sum_{i=1}^{t} w_{i,j} * w_{i,k}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} * \sqrt{\sum_{i=1}^{t} w_{i,k}^2}} \quad (16)$$

where $Sim_{j,k}$ is the cosine similarity value between the two sentences j and k. $W_{i,j}$ is the weight of the term i in the sentence j. t is the number of terms in the sentence.

3. The weight W of the term j in a search query i (i represents a sentence in a single cluster) can be computed using Eq. (17):

$$W_{i,j} = tf_{i,j} * \log\frac{n}{df_j} \quad (17)$$

where $tf_{i,j}$ is the number of times that the term j appears in cluster i summary in *Cluster_Index*. n is the number of clusters in the collection. In a single cluster, n is the number of all sentences in the cluster. df_j is the cluster frequency of term *j*. In a single cluster, it is the sentence frequency of term *j* (Alguliev and Aliguliyev 2007).

4. For search query, the AU computes the trapdoor as follows (T_{w_i}, *index-for-specific-cluster*, *k*) where *k* is the number of the retrieved documents and T_{w_i} is an encrypted form for each word in the search query *w* as follows:

$$T_{w_i} = \sum_{i=1}^{m} H(w_i)^r \tag{18}$$

where *H* is a collision-resistant hash function (Secure Hash Algorithm 1 (SHA-1)). The SHA-1 is a cryptographic hash function; it produces a single output message digest of 160-bit (20-byte) from an input message. Multiple chunks of 512 bits each compose the input message. Afterward, each chunk is further divided into sixteen 32-bit words ($W_t = \{W_0, W_1, W_2, \ldots W_{15}\}$), one 32-bit word for each round of the SHA-1 processing (Chaves et al. 2006). The T_{w_i} generation is illustrated in Algorithm 10.

Algorithm (10): The SHA-1 Algorithm

```
Input: the search query w with appending a one-bit equal one, followed by enough zeros until
the message is 512 bits
Output: encrypted form for each word in the search query $T_{w_i}$
Set Initialize variables $H_0$, $H_1$, $H_2$, $H_3$, $H_4$
        For each data-chunk
            $W_t$ = expand(data block)
            $A = H_0$ ; $B = H_1$ ; $C = H_2$ ; $D = H_3$ ; $E = H_4$
            For t= 0, t≤79, t=t+1
                Temp = RotL5 (A) + $f_t$(B,C,D) + E + $K_t$ + $W_t$
                E = D
                D = E
                C = RotL30(B)
                B = A
                A = Temp
            End For
            $H_0 = A + H_0$ ; $H_1 = B + H_1$ ; $H_2 = C + H_2$ ; $H_3 = D + H_3$ ; $H_4 = E + H_4$
            $T_{w_i}$ = RotL128 ($H_0$) ∨ RotL96 ($H_1$) ∨ RotL64 ($H_2$) ∨ RotL32 ($H_3$) ∨ $H_4$
        End For
```

4.2.2 Ranked keyword search

The CSP searches for the matching documents in a specific cluster after receiving the trapdoor (T_{wi}, *index-for-specific-cluster*, *k*) from the AU or the DO. The CSP conducts a ranked keyword-based search as illustrated in Algorithm 11.

Algorithm (11): The Ranked Keyword Search Algorithm

```
Input: index-for-specific-cluster, the Words_Index
Output: top-k most relevant documents
  Selects the matching cluster $C_i$ using the index-for-specific-cluster and the Words_Index.
  Avg=0
  Count=0
    For each term j in ($T_{wi}$) do
        For each document $d_j$ in $C_i$ do
            compute the score of j in $d_j$ from the Words_Index
            add the score to Avg
            increase Count
        End For
      compute the Avg score for $T_{wi}$ in each document using:
      $Avg_j$ ($T_{wi}$)= $Avg_j$ / Count
    End For
  Sends top-k most relevant documents in descending order to the AU
  End
```

The CSP first compares the *index-for-specific-cluster* with the clusters number in *Words_Index* to select the matching cluster and then finds the matching entries of the corresponding encrypted document in this cluster using (T_{wi}). The CSP gets the matched document identifiers by checking whether the terms in the search query existed in these documents and then finding the ranks for each matched terms to determine the matched documents according to the relevance scores and send the top-k most relevant documents in descending order back to the DO or the AU. To obtain the rank of the corresponding documents, the B^+ tree-based and the dictionary (key/value) data structure is used per cluster. The B^+ tree-based is a balanced search tree with O ($\log(n)$) searching time for the worst-case scenario. The data that represent the score for each word are stored at the leaf nodes, where the other nodes only store the keys. The B^+ tree-based and the dictionary (key/value) data structure can be demonstrated as the following definition.

Definition 4 Assume u is a node of index tree I_t, then if a node u is an internal (non-leaf) node, then it has the following four fields: (1) $u.n$ is the router value currently stored in node u, (2) the router values stored in node u in increasing order ($u.router_1 < u.router_2 < \cdots < u.router_{u.n}$), (3) $u.leaf$ is a Boolean field where a zero value *indicates* that u is a non-leaf node and (4) $u.n + 1$ pointers $u.c_1$, $u.c_2$, $u.c_3$, …, $u.c_{u.n+1}$ to the children of u. If a node u is a leaf node, then it has the following three fields (1) $u.n$ is the number of key values currently stored in u. (2) The key values stored in node u in increasing order $u.key_1 < u.key_2 < \cdots < u.key_{u.n}$ and (3) $u.leaf$ are a Boolean field where one value means that u is a leaf node. The searching process using B^+ tree-based is illustrated in Algorithm 12.

Algorithm (12): The searching process using B+ tree-based Algorithm

```
Input: Key k, Tree T
Output: the key values of the node searched for
 x = T.root      // x is the node in the tree T
   while not x.leaf
       i = 1
         While i ≤ x.n and k > x.router_i  // n is the number of the keys stored in the tree.
                                   // x.router_i is the first router value which is
                                   // greater than or equal to the key k searched for.
           i = i+1
         End
       x = x.ci
       Read(x)
   End
 i = 1
   while i ≤ x.n and k > x.key_i   // all router values in node x are smaller than the
                                   // key k searched for.
       i = i+1
   End
 If i ≤ x.n and k = x.key_i
     Return ( x, i )
   Else return NULL
 End
```

The CSP first selects the matching *cluster* using the *index-for-specific-cluster* and then selects the corresponding tree for this cluster and starts comparing each term in the trapdoor (Tw_i) against the selected tree to determine which documents contain the search query. Then it computes the score for each term in (Tw_i) to produce the final score for each document. Finally, these documents must be sorted in descending order based on their scores, and the top-k most relevant documents will be retrieved and send back to the AU or the DO, as portrayed in Fig. 4.

After the ranked keyword search is completed, the top-k most relevant documents will be retrieved and send back to the AU or the DO in an encrypted format.

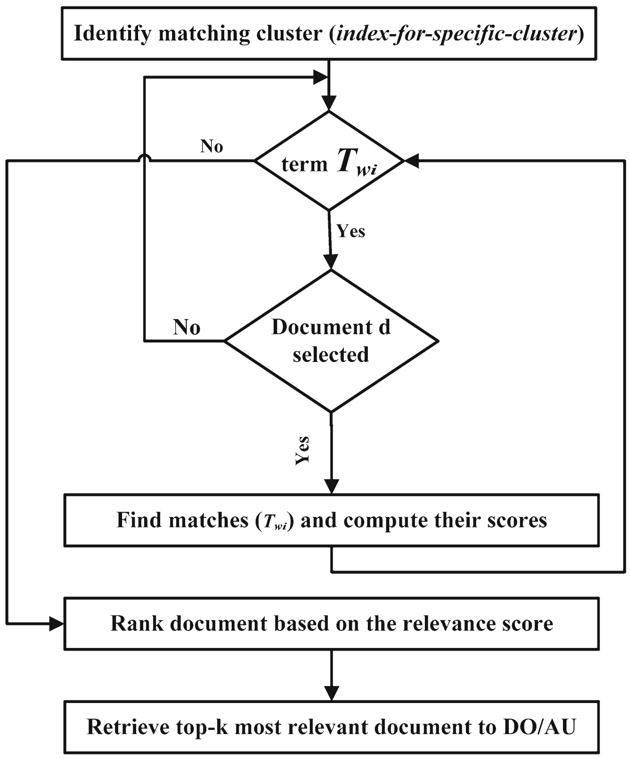

Identify matching cluster (*index-for-specific-cluster*)
No
term T_{wi}
Yes
No
Document d selected
Yes
Find matches (T_{wi}) and compute their scores
Rank document based on the relevance score
Retrieve top-k most relevant document to DO/AU

Fig. 4 Ranked keyword search

4.2.3 Data decryption

The AU and the DO need to decrypt the received documents from the CSP in response to his/her search query. The plain text could be obtained using the shared private key (Pasupuleti et al. 2016). The procedure of the decryption process is illustrated in Algorithm 13.

```
Algorithm (13): The Data Decryption Algorithm
Input: encrypted text of document $c_i$
Output: plaintext of document $d_i$
computes the four modular square roots (r,c,a,b)
  $r = ((p+1)/4)^{n+1} \, mod (p-1)$
  $c = ((q+1)/4)^{n+1} \, mod (q-1)$
  $a = x_{n+1}^{r} mod\, p$
  $b = x_{n+1}^{c} mod\, q$
  $Y = brp + asq \; mod\, N$
    For i to n do
        $X_i = Y_{i-1}^2 mod\, N$
        Let $p_i$ be the h least significant bits of $X_i$
    End For
  Compute $d_i = P_i \oplus c_i$
End
```

To decrypt the document, the AU computes the four modular square roots (r, c, a, b) and uses them to get the plaintext d_i (Pasupuleti et al. 2016).

To ensure the integrity of the received documents, the authorized user will check the documents by matching the hash value from CSP with the hash value from the data owner, so the authorized user can detect the modification of the encrypted data in a cloud environment if there are matches between these hash values. This could be achieved using any collision-resistant hash function H (e.g., *SHA-1*). As a result, if any hacker changes the document content, the index content or the encrypted documents, then h_1 ! = h_2 which means the data are changed or corrupted in the cloud, so our approach can achieve the data integrity of encrypted documents stored in the cloud. Therefore, privacy-preserving requirements are satisfied in our approach.

5 Security analysis

Privacy preserving of cloud computing data is considered as one of the most important issues that stop users from adopting cloud computing into their infrastructures. Therefore, in this section, we analyze the security of PPMKS against internal and external attacks. To analyze the security of PPMKS, we adopt the following definitions proposed by Pasupuleti et al. (2016).

Definition 1 (*Semantic security*) Semantic security indicates that if one is given a ciphertext, the internal and external adversary should learn nothing about the corresponding encrypted plaintext. Thus, we can say that it is semantically secure.

Definition 2 (*Data privacy*) If an adversary gets some of the retrieved encrypted data and the corresponding secret key, he can learn nothing about the plain data in polynomial time.

Definition 3 (*Index privacy*) If an adversary gets hold of a given I for a set of keywords, he should learn nothing from the corresponding keywords in polynomial time.

Definition 4 (*Data integrity*) If an internal or external attacker modified the data, the changes should be detected by the users.

Definition 5 If for any polynomial-time probabilistic algorithm A, i.e., $Pr[(x, y) = A(1^k, H) : x \neq y \wedge H(x) = H(y)]$ is negligible, then a hash function H is collision resistant.

Theorem 1 *The PPMKS is semantically secure against internal and external attacks according to* Definition 1.

Proof If we prove that internal and external adversary cannot access or learn nothing from the ciphertext and indexes then we can say PPMKS is semantically secure. Consider our key generation and encryption process, the

DO selects two large distinct and primes p, q then using an extended Euclidean algorithm to generate $PK = \{N\}$ and $PR = \{p, q, r, s\}$, then the documents and indexes are encrypted using PK, $x_i = x_{i-1}^2 \mod N$, $p_i = x \mod 2$ and $c_i = p_i \oplus d_i$. Observe that N is a large integer; an internal and external adversary can see only the ciphertext c_i. If factoring N is difficult, then p_i is LSB of the principal square root x_n of x_{n+1} modulo N is simultaneously secure. Thus, the internal and external adversary can do nothing better than guessing the pseudorandom bits p_i, $1 \leq i \leq t$. More formally, if the integer factorization problem is hard, then the PPMKS is semantically secure against internal and external adversary attacks. □

Theorem 2 *Based on* Definitions 2 and 3, *the PPMKS approach satisfies privacy preserving for documents and indexes.*

Proof We have to prove that PPMKS satisfies privacy preserving for documents and indexes against the internal and external attack. In the chosen-ciphertext attack (CCA), if the attacker has temporary access to ciphertext, then he may try to decrypt it to get the plaintext. To prove that, assume that the composite number N be the modules in the RSA module $N = pq$ where p and q are two large distinct and primes with the same size. Assume that the document of the DO contains such a composite number N that two factors p and q only known to the DO. Define p_i to be the bit vector whose value is the least significant bits of x_i, where $x_i = x_{i-1}^2 \mod N$. To encrypt the d_i pick a random p_i, then DO computes $c_i = p_i \oplus d_i$ and $x_{n+1} = x_n^2 \mod N$. To decrypt d_i, the AU computes $x_i = x_{i-1}^2 \mod N$ and $d_i = p_i \oplus c_i$. The given $x_i = x_{i-1}^2 \mod N$ is very hard for any attacker to compute such random seed x to access the document. Based on the above, this approach is secure against CCA. Based on our encryption approach and its security strength against internal and external adversary attacks as well as the CCA from Theorems 1 and 2, PPMKS satisfies privacy preserving for documents and indexes. □

Theorem 3 *Based on* Definition 4, *PPMKS approach satisfies the data integrity for the documents.*

Proof We have to prove that the PPMKS approach has data integrity for the documents against internal and external attacks. If internal or external attacker corrupts any retrieved data, the AU should check this attack by matching hash value h_2 from CSP with hash value h_1 from DO. Then the AU can detect if the data modification occurs in the cloud. So, if $h_1 = h_2$ says that encrypted documents and indexes are not modified in the cloud then the data integrity is satisfied, otherwise encrypted documents and indexes are modified in the cloud. That is, if an attacker tries to add some data to the encrypted documents or encrypted indexes, then the AU should be able to detect this change. So, PPMKS approach satisfies the integrity of documents stored in the cloud against internal and external attacks. □

Theorem 4 *Based on* Definition 5, *the H is collision resistant, and it is very hard for an attacker to find two distinct inputs $x \neq y$ that have same hash value $H(x) = H(y)$.*

Proof We have to prove that the attacker cannot guess the same hash value $H(x) = H(y)$ on two different inputs. The two distinct inputs x, y are called collision for a hash function H if $x \neq y$ but have same hash values $H(x) = H(y)$. The H is collision resistance that satisfies the collision resistance requirement, for a probabilistic polynomial-time algorithm A, if two distinct inputs x, y, are used to find a collision for the function H with minimum probability. Therefore, if H is collision resistant, it is impracticable to guess two distinct inputs x, y that produce the same hash value. Hence, it is very difficult for an attacker to guess a hash value which makes $h_1 = h_2$ due to the security strength of hash function. □

6 Performance evaluation

In this section, we present the performance analysis of the PPMKS, in which the search precision and search time are analyzed separately, and then compared with other approaches MRSE-HCI (Chen et al. 2016) and MUSE (Xiangyang et al. 2017). The search precision (Δ) and search time (Δ) on a real data set of NSF Research (Bache and Lichman 2013) is also evaluated.

6.1 Dataset and evaluating environment

The real dataset consists of 129,000 abstracts describing NSF awards for basic research from 1990 to 2003, and we randomly choose 20,000 abstracts for our experimental data and extract about 1000 distinct keywords. The experimental hardware environment is Intel Core i7-8550, 4 GHz, processor cache 8M, number of cores 4, 16 G memory and SSD hard disk; and software environment is Windows 10 Pro 64-bit operating system and Visual Studio 2017 development platform with C# Programming Language.

6.2 Search precision and time evaluation

The search precision (Δ) is used to estimate how well the search results satisfy the user's satisfaction. In order to evaluate Δ, we adopt the definition of precision from Xiangyang et al. (2017) as in Eq. (19).

$$\Delta p = \frac{k'}{k} \tag{19}$$

where k is the number of documents retrieved, and k' is the number of the real top- documents that are retrieved by CSP. The average value of Δ for PPMKS is shown in Fig. 5.

This figure illustrates that the Δ of PPMKS does not change when k changes from 50 to 200.

Further, we evaluated and analyzed the average search time (Δ) based on the number of queried keywords (q), number of documents (m) and the number of retrieved documents (k). The evaluation of Δ is based on q is illustrated in Fig. 6. This figure shows the time that is needed by the CSP to search the documents based on a trapdoor from the authorized user. The search time includes fetching the *Word_Index* and computing the score for each keyword in the queried keywords for each document in the same cluster that have the highest similarity score with the queried keywords.

The search time increases as q grows, because of the computation needed for each keyword in the query and this will consume more time. For example, PPMKS needs on average (22 ms) to search for a query of 20 keywords and (55 ms) to search for a query of 50 keywords. The Δ of PPMKS based on m should be affected by the number of documents for each cluster, so we will use PPMKS-100 to represent a cluster with a maximum number of documents to be 100 and PPMKS-300 to represent a cluster with a maximum number of documents to be 300. The evaluation of Δ based on m is illustrated in Fig. 7.

Figure 7 shows that the search time Δ of PPMKS-100 is lower than PPMKS-300, because the Δ is affected by the number of documents in each cluster, the Δ of PPMKS in the second scenario does not change significantly because the search time is based on the number of documents in each cluster and not the number of all documents (m). The search time of PPMKS does not change while grows because the searching process is carried out for all documents in the same cluster regardless of the number of the

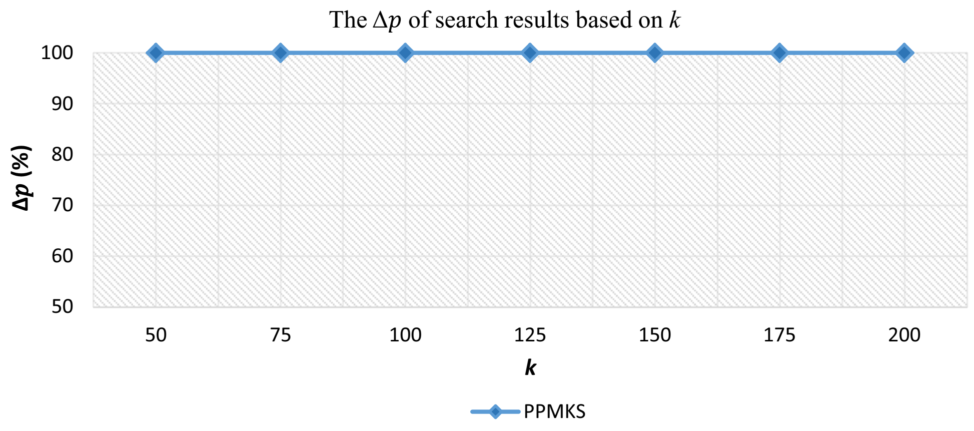


Fig. 5 The search precision of search results based on k

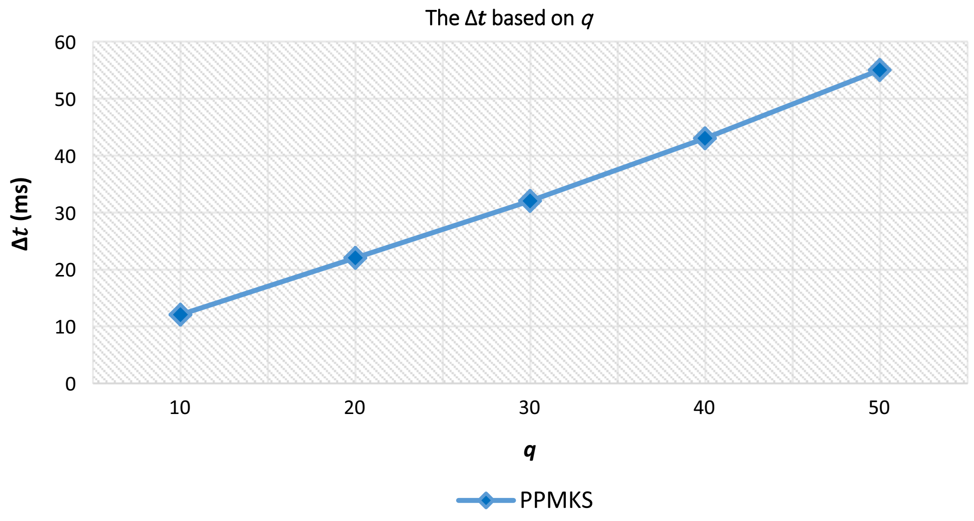


Fig. 6 The Δ of the CSP to search the documents based on q

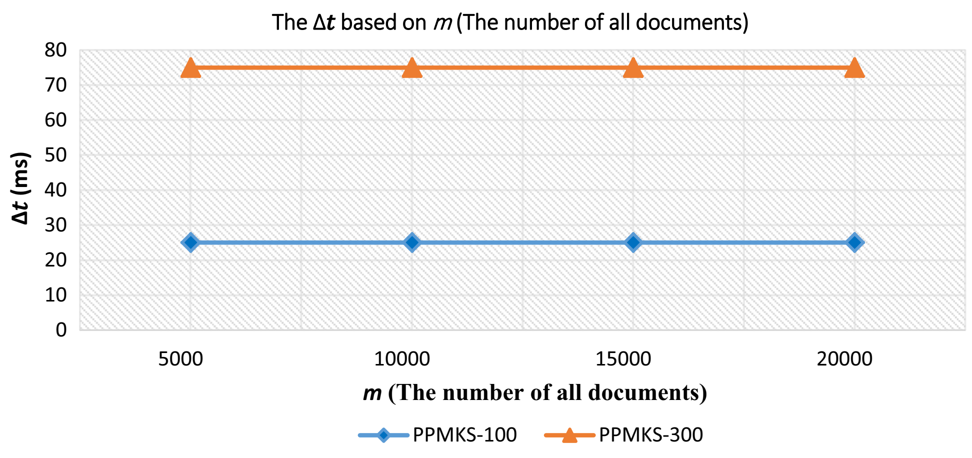


Fig. 7 The Δ of the CSP to search the documents based on *m*

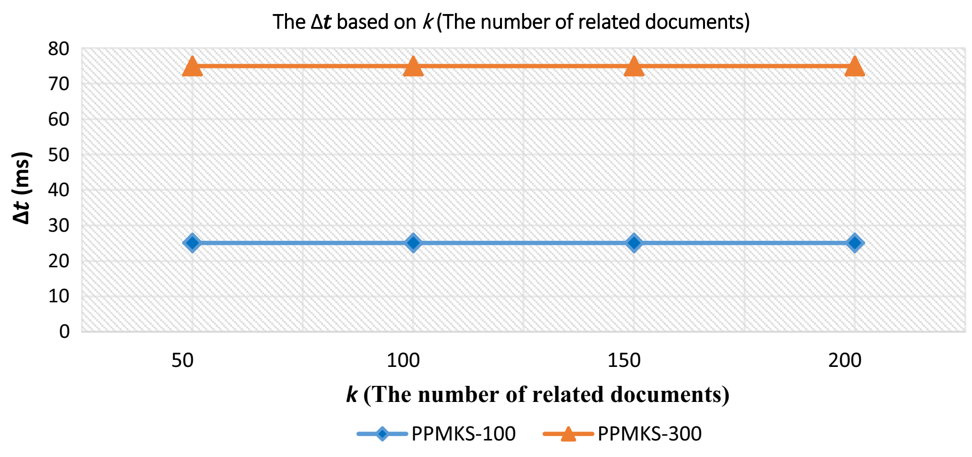


Fig. 8 The Δ of the CSP to search the documents based on *k*

retrieved documents (*k*). The evaluation of Δ based on *k* is illustrated in Fig. 8.

Figure 8 illustrates that the search time Δ of PPMKS-100 is lower than PPMKS-300 scenario and Δ does not change while grows. For example, PPMKS-100 needs on average 25 ms to search if the number of retrieved documents is 100 documents, and PPMKS-300 needs on average 75 ms to search if the number of retrieved documents is 150 documents.

The search time of PPMKS changes based on the maximum number of documents in the cluster; because the searching process is carried out for all documents in the same cluster, so the search time will increase while the number of documents in the cluster increases. The evaluation of Δ based on the maximum number of documents in the cluster is illustrated in Fig. 9.

Figure 9 shows that the search time Δ of PPMKS increased based on the maximum number of documents in the cluster. The search time includes fetching the words entry list from the *Words_Index* and computing the score for all keywords in the query, for all documents in the same cluster that have the highest similarity score with the query, and find the highest documents score. For example, PPMKS needs on average 0.1 s to search if the maximum number of documents in the cluster is 400 documents, and PPMKS needs on average 0.175 s to search if the maximum size of the cluster is 700 documents.

6.3 The search precision evaluation

In this section, we compare Δ with the results of other approaches presented in (Chen et al. 2016) and (Xiangyang et al. 2017) which are denoted as MRSE-HCI and MUSE, respectively. The average value of Δ for MRSE-HCI, MUSE and PPMKS is shown in Fig. 10.

This figure illustrates that the Δ for MRSE-HCI, MUSE and PPMKS does not change significantly when *k* change from 50 to 200, the PPMKS's and MUSE's Δ are significantly larger than MRSE-HCI. The PPMKS's and MUSE's Δ are approximately equivalent because the two approaches perform the same calculation of vectors space model and query vectors. The MRSE-HCI's Δ is lower than the other approaches because the MRSE-HCI clusters the very most relevant documents into the same cluster by dynamic

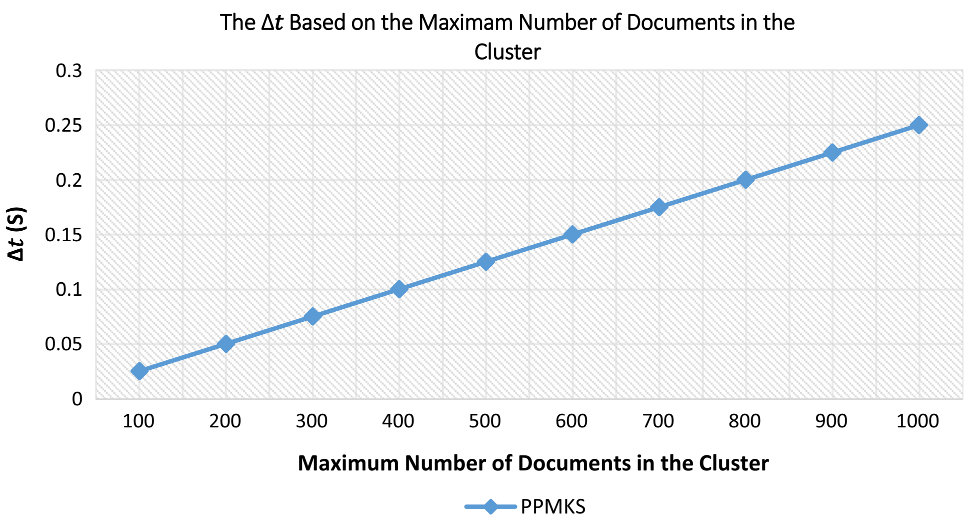


Fig. 9 The Δ of the CSP to search the documents based on the maximum size of cluster

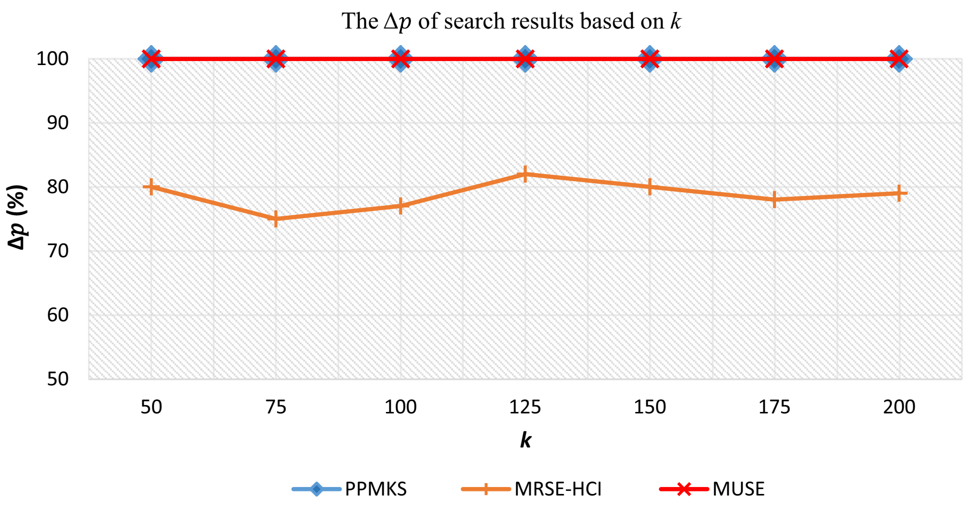


Fig. 10 The search precision of search results based on k

-means, which means the search result will contain only the very most relevant documents and some relevant documents might be ignored.

6.4 The search time evaluation

In this section, we compare the average search time (Δ) with the results of other approaches based on the number of queried keywords (q), number of documents (m) and the number of retrieved documents (k). In MRSE-HCI, the maximum number of documents for clusters should be initialized; thus, we will use MRSE-HCI-100 to represent the maximum numbers of documents per clusters to be 100 and MRSE-HCI-300 to represent the maximum numbers of documents per clusters to be 300 documents.

The evaluation of Δ based on q is illustrated in Fig. 11; this figure shows the time that is needed by the CSP to search the documents based on a trapdoor that sends from the authorized user. The search time includes fetching the *Word_Index* and computing the score for each keyword in the queried keywords for each document in the same cluster that have the highest similarity score with the queried keywords.

By comparing all approaches, our approach uses the advantages of clustering by separating data categories by similar features; that means the server may not search for all documents, but search only in a specific cluster based on the query. This approach takes less time to search the documents based on a trapdoor. For example, MRSE-HCI-100 needs 38 ms, MRSE-HCI-300 needs 100 ms and MUSE needs 85 ms to search if the number of queried keywords is 30 keywords, while our model needs on average 32 to search if the number of queried keywords is 30 keywords.

The evaluation of Δ based on m for PPMKS, MRSE and MUSE approaches is illustrated in Fig. 12. This figure shows that the search time Δ of PPMKS is lower than MRSE in both 100 and 300 documents. The Δ of PPMKS

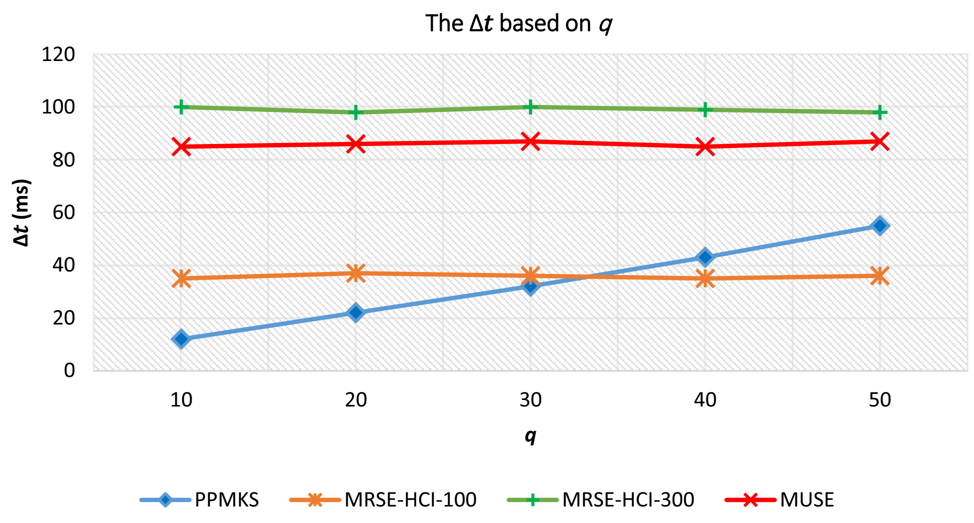


Fig. 11 The Δ of the CSP to search the documents based on *q*

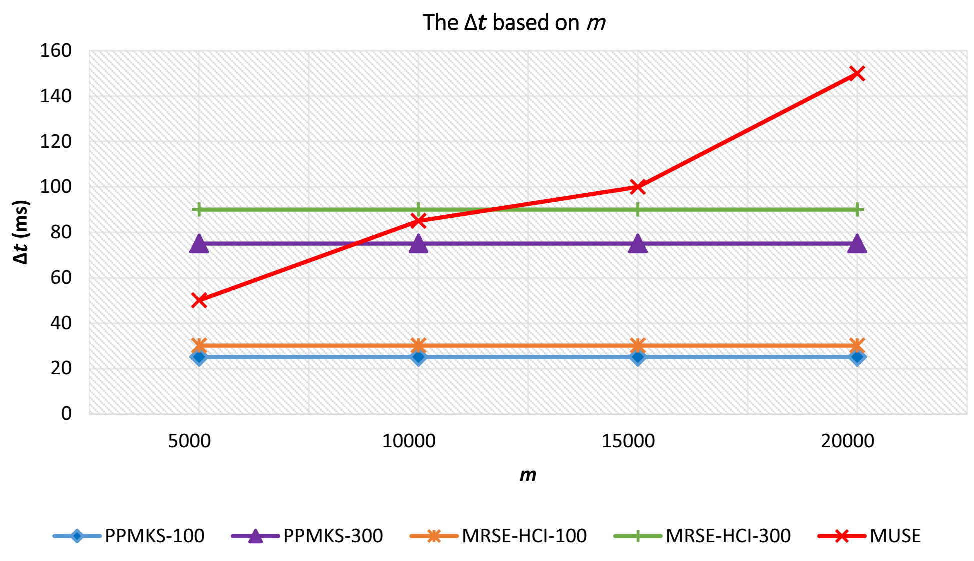


Fig. 12 The Δ of the CSP to search the documents based on *m*

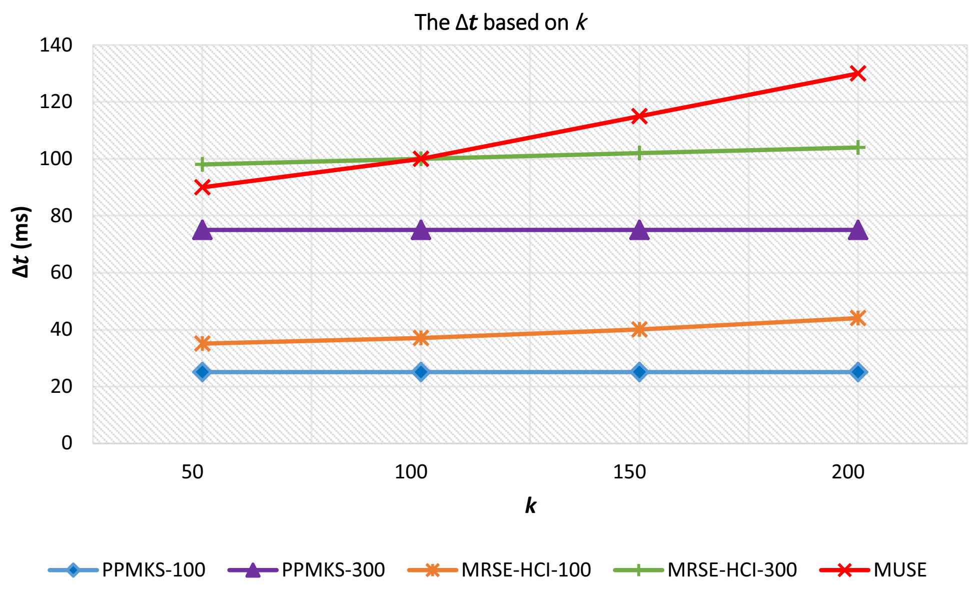


Fig. 13 The Δ of the CSP to search the documents based on *k*

and MRSE does not change significantly because the search time in these two approaches was based on the number of documents in each cluster and does not get affected by the number of all documents (m), while MUSE increase in increased.

The evaluation of Δ based on k is illustrated in Fig. 13, and the search time of all other approaches increases as grows, because if grows, this will consume more computation time, thus, increase the search time. The PPMKS does not change while grows because the searching process was done for all documents in the same cluster regardless of the number of the retrieved documents (k).

Figure 13 shows that the search time Δ of PPMKS is lower than MRSE and MUSE with 100 and 300 documents per cluster. The Δ of MUSE and MRSE increases as grows because these approaches need more computation time while the number of retrieved documents is increases. For example, MRSE-HCI-100 needs 40 ms, MRSE-HCI-300 needs 102 ms, and MUSE needs 116 ms to search if the number of retrieved documents is 150 documents, while PPMKS-100 needs on average 25 ms and PPMKS-300 needs on average 75 ms to search if the number of retrieved documents is 150 documents.

7 Conclusion and future work

In this paper, we proposed a new privacy-preserving approach based on topic summary and clustering to reduce the time and the number of comparisons required for searching and retrieving outsourced encrypted data. Although there are many benefits for cloud computing, privacy-preserving concerns are one of the main challenges that still considered a barrier for users to adopt cloud computing in their infrastructure, because outsourcing the data to a third-party deprives the DO of direct control to their data and applications. The best solution to preserve the privacy of any sensitive and important data is to encrypt data before outsourcing it. The encryption protects from malicious attacks and illegal accesses, but it significantly increases the computation and the communication overhead on the data owners especially for clients with large data size. It is very important to allow any DO or AU to send multiple keywords in the search request and retrieve the related documents in the order of their relevance to these keywords. The ranked search system allows data users to perform the searching process quickly by finding the most relevant documents. Developing such an approach will add a significant contribution to the domain of privacy-preserving multi-keyword search over encrypted data. However, the PPMKS approach yields good results in the evaluation process which shows an enhancement on the performance of the cloud environment.

The PPMKS approach was developed to retrieve the most relevant document with the fastest searching time. We achieved this objective by using the probabilistic public key encryption approach to reduce the computation overhead of the data owner device while encrypting the data. Also, using the clustering and summarization technique reduces the time of the searching process. Also, the ranked multi-keyword searching over the encrypted data reduces the communication overhead during the files retrieval phase, hence providing the most relevant files to the users with minimum false positives.

We are verifying and validating the proposed approach and comparing the results to those related to the filed from the literature. We achieved this objective by developing testing and evaluating environment to compare the search precision and the search time of the proposed approach against others. In comparison with these studies, it was found that the proposed approach made a significant enhancement on the search precision and the search time.

In the future, we will focus our efforts on achieving more enhancements on the overall performance of the cloud environment. As a future work in PPMKS approach, we will enhance PPMKS to support more efficient dynamic data operations and ranked multi-keyword search over the big encrypted data in the cloud environment. We will achieve this goal by applying a dynamic clustering algorithm that may use artificial intelligence concepts to discriminate the optimal number of clusters. We will also enhance PPMKS to support more integrity check of rank order in the multi-keyword search result and privacy-preserving guarantees in the stronger threat model.

Acknowledgements We are thankful for the people who gave technical assistance in getting the related IR information and concepts.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflicts of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

Adjedj M, Bringer J, Chabanne H, Kindarji B (2009) Biometric identification over encrypted data made feasible. In: Information systems security. Springer, Berlin, pp 86–100

Alguliev R, Aliguliyev R (2007) Experimental investigating the F-measure as similarity measure for automatic text summarization. Appl Comput Math 6:278–287

Aljammal AH, Manasrah AM, Abdallah AE, Tahat NM (2017) A new architecture of cloud computing to enhance the load balancing. Int J Bus Inf Syst 25:393–405

Bache K, Lichman M (2013) UCI machine learning repository. http://archive.ics.uci.edu/ml

Bellare M, Boldyreva A, O'Neill A (2007) Deterministic and efficiently searchable encryption. In: Advances in cryptology—CRYPTO 2007. Springer, Berlin, pp 535–552

Blei DM, Ng AY, Jordan MI (2003) Latent Dirichlet allocation. J Mach Learn Res 3:993–1022

Boneh D, Di Crescenzo G, Ostrovsky R, Persiano G (2004) Public key encryption with keyword search. In: Advances in cryptology—Eurocrypt 2004. Springer, Berlin, pp 506–522

Cao N, Wang C, Li M, Ren K, Lou W (2014) Privacy-preserving multi-keyword ranked search over encrypted cloud data. IEEE Trans Parallel Distrib Syst 25:222–233

Chang Y-C, Mitzenmacher M (2005) Privacy preserving keyword searches on remote encrypted data. In: Applied cryptography and network security. Springer, Berlin, pp 442–455

Chase M, Kamara S (2010) Structured encryption and controlled disclosure. In: Advances in cryptology—ASIACRYPT 2010. Springer, Berlin, pp 577–594

Chaves R, Kuzmanov G, Sousa L, Vassiliadis S (2006) Rescheduling for optimized SHA-1 calculation. In: International workshop on embedded computer systems. Springer, Berlin, pp 425–434

Chen C, Zhu X, Shen P, Hu J, Guo S, Tari Z, Zomaya AY (2016) An efficient privacy-preserving ranked keyword search method. IEEE Trans Parallel Distrib Syst 27:951–963

Curtmola R, Garay J, Kamara S, Ostrovsky R (2006) Searchable symmetric encryption: improved definitions and efficient constructions. In: Proceedings of the 13th ACM conference on computer and communications security. ACM, pp 79–88

Gupta B, Agrawal DP, Yamaguchi S (2016) Handbook of research on modern cryptographic solutions for computer and cyber security. IGI Global, Hershey

Handa R, Challa RK (2015) A cluster based multi-keyword search on outsourced encrypted cloud data. In: 2015 2nd International conference on computing for sustainable global development (INDIACom). IEEE, pp 115–120

Jiang ZL et al (2018) Efficient two-party privacy preserving collaborative k-means clustering protocol supporting both storage and computation outsourcing. In: Vaidya J, Li J (eds) Algorithms and architectures for parallel processing. Springer International Publishing, Cham, pp 447–460

Katz J, Sahai A, Waters B (2008) Predicate encryption supporting disjunctions, polynomial equations, and inner products. Advances in cryptology—EUROCRYPT 2008. Springer, Berlin, pp 146–162

Krishna CR, Handa R (2016) Dynamic cluster based privacy-preserving multi-keyword search over encrypted cloud data. In: 2016 6th International conference on cloud system and big data engineering (Confluence). IEEE, pp 146–151

Li P, Li J, Huang Z, Gao C-Z, Chen W-B, Chen K (2017) Privacy-preserving outsourced classification in cloud computing. Cluster Comput 21:1–10

Manasrah AM (2017) Dynamic weighted VM load balancing for cloud-analyst. Int J Inf Comput Secur 9:5–19

Manasrah AM, Al-Din BN (2016) Mapping private keys into one public key using binary matrices and masonic cipher: Caesar cipher as a case study. Secur Commun Netw 9:1450–1461

Manasrah AM, Gupta B (2017) An optimized service broker routing policy based on differential evolution algorithm in fog/cloud environment. Cluster Comput. https://doi.org/10.1007/s10586-017-1559-z

Manasrah AM, Smadi T, ALmomani A (2016) A variable service broker routing policy for data center selection in cloud analyst. J King Saud Univ Comput Inf Sci 29:365–377

Nagwani N (2015) Summarizing large text collection using topic modeling and clustering based on MapReduce framework. J Big Data 2:6

Nedjah N, Wyant RS, Mourelle L, Gupta B (2017) Efficient yet robust biometric iris matching on smart cards for data high security and privacy. Future Gener Comput Syst 76:18–32

Pasupuleti SK, Ramalingam S, Buyya R (2016) An efficient and secure privacy-preserving approach for outsourced data of resource constrained mobile devices in cloud computing. J Netw Comput Appl 64:12–22

Plageras AP, Psannis KE, Stergiou C, Wang H, Gupta BB (2018) Efficient IoT-based sensor BIG Data collection—processing and analysis in smart buildings. Future Gener Comput Syst 82:349–357

Porter MF, Boulton R, Macfarlane A (2002) The english (porter2) stemming algorithm. Retrieved 18 2011

Ramasubramanian C, Ramya R (2013) Effective pre-processing activities in text mining using improved porter's stemming algorithm. Int J Adv Res Comput Commun Eng 2:4536–4538

Ramos J (2003) Using tf-idf to determine word relevance in document queries. In: Proceedings of the first instructional conference on machine learning, pp 133–142

Ranjan KA, Pasupulati SK, Ramaligam S (2017) Privacy-preserving multi-keyword search over the encrypted data for multiple users in cloud computing. In: International conference on inventive computing and informatics (ICICI). IEEE, pp 1079–1084

Salton G, Buckley C (1988) Term-weighting approaches in automatic text retrieval. Inf Process Manag 24:513–523

Salton G, Wong A, Yang C-S (1975) A vector space model for automatic indexing. Commun ACM 18:613–620

Sarkar K (2012) Bengali text summarization by sentence extraction. arXiv preprint arXiv:12012240

Stergiou C, Psannis KE, Kim B-G, Gupta B (2018) Secure integration of IoT and cloud computing. Future Gener Comput Syst 78:964–975

Takabi H (2014) Privacy aware access control for data sharing in cloud computing environments. In: Proceedings of the 2nd international workshop on security in cloud computing. ACM, pp 27–34

Wang J, Ma H, Tang Q, Li J, Zhu H, Ma S, Chen X (2012) A new efficient verifiable fuzzy keyword search scheme. JoWUA 3:61–71

Witten IH, Frank E, Hall MA, Pal CJ (2016) Data mining: practical machine learning tools and techniques. Morgan Kaufmann, Los Altos

Xiangyang Z, Hua D, Xun Y, Geng Y, Xiao L (2017) MUSE: an efficient and accurate verifiable privacy-preserving multikeyword text search over encrypted cloud data. Secur Comm Netw. https://doi.org/10.1155/2017/1923476

Xu Y, Cui W, Peinado M (2015) Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In: 2015 IEEE symposium on security and privacy (SP). IEEE, pp 640–656

Yin H, Zhang J, Xiong Y, Huang X, Deng T (2018) PPK-means: achieving privacy-preserving clustering over encrypted multidimensional cloud data. Electronics 7:310

Zhao W, Ma H, He Q (2009) Parallel k-means clustering based on mapreduce. IEEE International Conference on Cloud Computing. Springer, Berlin, pp 674–679

Zhu X, Chen C, Tian X, Hu J (2015) HCSF: a hierarchical clustering algorithm based on swarm intelligence and fuzzy logic for ciphertext search. In: 2015 IEEE 10th Conference on industrial electronics and applications (ICIEA). IEEE, pp 290–295