Hindawi Publishing Corporation
The Scientific World Journal
Volume 2015, Article ID 706102, 17 pages
http://dx.doi.org/10.1155/2015/706102

Research Article

Universal Keyword Classifier on Public Key Based Encrypted Multikeyword Fuzzy Search in Public Cloud

Shyamala Devi Munisamy[1] and Arun Chokkalingam[2]

[1]*R.M.D Engineering College, R.S.M Nagar, Kavaraipettai, Chennai, Tamil Nadu 601206, India*
[2]*R.M.K College of Engineering and Technology, R.S.M Nagar, Puduvoyal, Chennai, Tamil Nadu 601206, India*

Correspondence should be addressed to Shyamala Devi Munisamy; shyamalapmr@gmail.com

Received 20 May 2015; Revised 17 July 2015; Accepted 29 July 2015

Academic Editor: Juan M. Corchado

Cloud computing has pioneered the emerging world by manifesting itself as a service through internet and facilitates third party infrastructure and applications. While customers have no visibility on how their data is stored on service provider's premises, it offers greater benefits in lowering infrastructure costs and delivering more flexibility and simplicity in managing private data. The opportunity to use cloud services on pay-per-use basis provides comfort for private data owners in managing costs and data. With the pervasive usage of internet, the focus has now shifted towards effective data utilization on the cloud without compromising security concerns. In the pursuit of increasing data utilization on public cloud storage, the key is to make effective data access through several fuzzy searching techniques. In this paper, we have discussed the existing fuzzy searching techniques and focused on reducing the searching time on the cloud storage server for effective data utilization. Our proposed *Asymmetric Classifier Multikeyword Fuzzy Search* method provides classifier search server that creates universal keyword classifier for the multiple keyword request which greatly reduces the searching time by learning the search path pattern for all the keywords in the fuzzy keyword set. The objective of using BTree fuzzy searchable index is to resolve typos and representation inconsistencies and also to facilitate effective data utilization.

1. Introduction

Cloud computing [1, 2] makes the infrastructure, platform, and software as a service for the worldwide users. The cloud paradigm makes the user outsource their personal data to the cloud storage [3] server which facilitates the users' access to their data anywhere at any time. The data users of the cloud storage have to pay only for the actual storage they use. Some companies will use the cloud storage for their data backup.

Problem Formulation. We highlight here that the cloud storage server has the responsibility to keep their customer data to be available and accessible all the time. The cloud storage must facilitate their customers to access their wide range of resources, application, and data through internet service interface immediately and fast. The number of customers utilizing the cloud storage increases significantly every day. The data on the cloud storage increases dynamically due to the increasing demands of existing customer and addition of new customers. This means the cloud storage is under a state to respond to increasing customer data and effective access to their data. To retain their customers, the cloud storage must optimize its computational time for searching the requested data. It must have some efficient searching method or additional provisions to serve their customers to provide the requested data immediately. So with this observation, we propose a new searching method named Asymmetric Classifier Multikeyword Fuzzy Search which utilizes the universal keyword classifier to store the search path pattern of all the keywords of their customers data. This allows the cloud storage server to use its time effectively to perform multiprocessing of their growing customers. Our scheme also resolves typos and representation inconsistencies since the searching is done on BTree fuzzy searchable index.

Our Contributions. In this paper, we propose new Scheme *Asymmetric Classifier Multikeyword Fuzzy Search (ACMFS)* which greatly reduces the time spent for searching the data

and delivers the requested file immediately to the users. It also utilizes the data effectively from the cloud storage through fuzzy search on BTree fuzzy searchable index. Experimental results shows effective data utilization and the search efficiency of the proposed scheme. Our contributions are summarized as follows:

(i) Asymmetric Searchable Encryption allows the server to search over encrypted BTree fuzzy searchable index thereby providing effective data utilization.

(ii) The cloud storage server would not disclose the files to illegal access as they do not have the information about the multiple keywords *MKW* and files.

(iii) As the BTree fuzzy searchable index is created from wild card fuzzy keyword set, it tolerates typos and representation inconsistencies of authorized users.

(iv) Classifier search server uses universal keyword classifier for traversing the storage efficient BTree wild card fuzzy searchable index which stores all the search path pattern of the multiple keywords of the entire encrypted files.

Paper Organization. The rest of the paper is organized as follows. The related modules are discussed in Section 2 along with the limitations of the existing searching methods. In Section 3, we formulate our problem by designing the system model and goals of the proposed solution. Then we provide the detailed description of *Asymmetric Classifier Multikeyword Fuzzy Search* scheme in Section 4 followed by Section 5, which discusses the detailed design and implementation of algorithms of our proposed method. The Experimental results and performance analysis with output are shown in Section 6. We conclude our paper in Section 7.

2. Background

2.1. Related Work. Although Cloud Service Provider (CSP) hosts several third party data, Liu et al. [4] pointed out that managing sensitive data leads to security and privacy concerns. Cryptographic methods can be used to disclose the key only to authorized users to protect the data from untrusted CSP.

Ren et al. [5] state that users have several types of typing behaviour for keywords which are commonly termed as typos, representation inconsistencies, and typing habits. They suggested fuzzy keyword search to overcome these inconsistencies. Though the fuzzy keyword search is prevalent in popular search engines like Google, Bing, and so forth, it still poses risk in cloud storage due to inherent security and privacy obstacles. The searchable encryption [6–8] is recommended which takes encrypted data as files labeled with keywords and lets user securely search over the files through predefined keywords for retrieving them.

Zhou et al. [9] created k-gram based fuzzy keyword set for keywords W of the encrypted files C and Jaccard coefficient to calculate the keywords similarity.

Wang et al. [10] pointed out that keyword holds sensitive information of the files and thus keyword privacy must be protected for effective data utilization. Xu et al. [11] identified that third party could access the files by knowing the keyword search trapdoor. Xu proposed public key encryption with fuzzy keyword search (PEFKS) in which each keyword corresponds to exact keyword search trapdoor and fuzzy keyword search trapdoor.

Wang et al. [12] discusses that the search over encrypted data not only involves information retrieval techniques such as data structures for representing the searchable index but also depends on efficient search algorithms that run on the index.

2.2. Limitation of the Existing Methods

(1) Secured and privacy preserving keyword search [4]:

(i) The encryption and decryption process incurs high communication and computational cost.

(2) Secured fuzzy keyword search [5]:

(i) It does not support fuzzy search with public key based searchable encryption.

(ii) It could not carry out multiple keywords semantic search.

(iii) The update operation on fuzzy searchable index is not much efficient.

(3) K-gram based fuzzy keyword Ranked search [9]:

(i) The k-gram based fuzzy keyword set size depends on the jaccard coefficient value.

(4) Verifiable fuzzy keyword search (VFKS) [10]:

(i) The symbol tree fuzzy searchable index occupies more space in this search.

(5) Public key encryption with fuzzy keyword search:

(i) Creating fuzzy keyword index and exact keyword index is not compatible with large database.

(6) Privacy-preserving multikeyword fuzzy search [12]:

(i) It demands files with relatively high score to reduce the false negative rate.

3. Methodology of Our Scheme

3.1. Cloud Data Utilization Service Architecture. In this paper, we consider our cloud data utilization service architecture which consists of four entities as data owner, cloud storage server, classifier search server, and data users and is shown in Figure 1. Here we assume that the authorization is suitably done between the data owner and data users.

Initially, the data owner generates user's public and private key pair as (PUB_{KEY}, $PRIV_{KEY}$). Data owner has a set of K data files $DF = \{DF_1, DF_2, \ldots, DF_K\}$ that

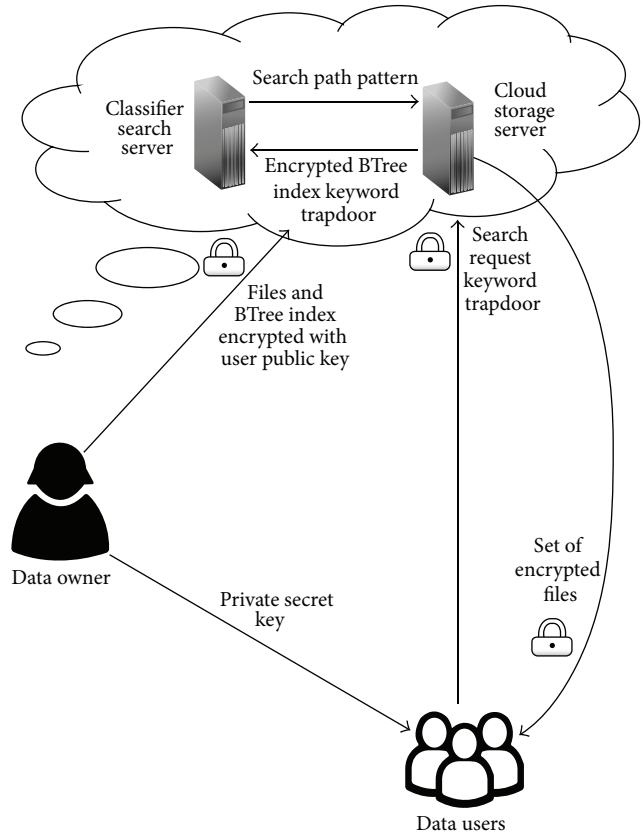


Figure 1: Cloud data utilization service architecture for Asymmetric Classifier Multikeyword Fuzzy Search.

are encrypted with user's public key PUB_{KEY} and are outsourced to the cloud storage server. Data owner predefine multiple keywords for each file. Data owner has a set of multiple keywords $MKW = \{(mk_{11}, mk_{12}, \ldots, mk_{1n}), (mk_{21}, mk_{22}, \ldots, mk_{2n}), \ldots, (mk_{k1}, mk_{k2}, \ldots, mk_{kn})\}$ of K data files. Data owner creates storage efficient wild card based fuzzy multikeyword set as $FMKS = \{(fmk_{11}[], fmk_{12}[], \ldots, fmk_{1n}[]), (fmk_{21}[], fmk_{22}[], \ldots, fmk_{2n}[]), \ldots, (fmk_{k1}[], fmk_{k2}[], \ldots, fmk_{kn}[])\}$ using wild card based technique with the predefined edit distance value. Data owner creates BTree wild card fuzzy searchable index BSI_{WC} from fuzzy multikeyword set. Data owner encrypts the files and index BSI_{WC} using user public key and is outsourced to the cloud storage server. Data owner sends the user private key as private secret key which is used by the data users for creating keyword trapdoor and for decrypting the file. Now the cloud storage server has the encrypted K data files DF and encrypted BTree wild card fuzzy searchable index BSI_E. The cloud storage server shares the encrypted BTree wild card fuzzy searchable index BSI_E to the classifier search server. The data user requests the multiple search keywords which are encrypted using the private secret key to create multikeyword trapdoor MKT_W which is sent to the cloud storage server. The server sends the request MKT_W to the classifier search server. The universal keyword classifier receives the request MKT_W to check whether the request is coming for the first time. If the request is arriving for the first time; then the keyword classifier captures and stores the path of the MKT_W by searching over the encrypted BTree wild card fuzzy searchable index BSI_E and sends the search path to the cloud storage server. If the request given by the user matches a previous request then it is a repeated multiple keyword. Then the classifier search server extracts the stored search path patterns of the repeated multikeyword from the universal keyword classifier and the search path is sent to the cloud storage server. After receiving the search path pattern of the multiple keywords from the classifier search server, the cloud storage server extracts the set of encrypted files from DF and is sent to the data users. After receiving the encrypted files, the data user decrypts the files using private secret key.

3.2. Design Goals. To effectively optimize the searching time for the multiple keywords in the cloud storage server and for tolerating the typos and representation inconsistencies of authorized users, our searching method seeks to achieve the following design goals.

Search efficiency goals are

(i) to construct the universal keyword classifier for BTree wild card fuzzy searchable index for optimizing search time and for tolerating typos and representation inconsistencies of authorized users.

Security goals are

(i) to avoid the cloud storage server from getting the knowledge of data files and keyword set. This is

achieved by outsourcing the encrypted files and index to the cloud storage server.

Privacy goals are

(i) to provide user privacy by abstracting the details of data files, keyword, and index to the cloud storage server;

(ii) to support data privacy by encrypting the files and index with user public key before outsourcing to the cloud storage server;

(iii) to attain keyword privacy by forming BTree wild card fuzzy searchable index from the fuzzy multikeyword set for the predefined set of multiple keywords;

(iv) to achieve query privacy by sending k-gram keyword trapdoor encrypted with the private secret key;

(v) to accomplish index privacy by creating encrypted BTree wild card fuzzy searchable index.

4. Asymmetric Classifier Multikeyword Fuzzy Search (ACMFS)

4.1. Notations and Preliminaries. They are as follows.

(i) **PSK**: private secret key;

(ii) **edit**: edit distance;

(iii) $\mathbf{DF} = \{\mathbf{DF_1}, \mathbf{DF_2}, \ldots, \mathbf{DF_K}\}$: set of K encrypted data files DF;

(iv) $\mathbf{PUB_{KEY}}$: user public key;

(v) $\mathbf{OPK_{PUB}}$: owner public key;

(vi) $\mathbf{PRIV_{KEY}}$: user private key;

(vii) $\mathbf{MKW} = \{(\mathbf{mk_{11}}, \mathbf{mk_{12}}, \ldots, \mathbf{mk_{1n}}), (\mathbf{mk_{21}}, \mathbf{mk_{22}}, \ldots, \mathbf{mk_{2n}}), \ldots, (\mathbf{mk_{k1}}, \mathbf{mk_{k2}}, \ldots, \mathbf{mk_{kn}})\}$: predefined multiple keywords set of DF;

(viii) $\mathbf{FMKS} = \{(\mathbf{fmk_{11}}[], \mathbf{fmk_{12}}[], \ldots, \mathbf{fmk_{1n}}[]), \ldots, (\mathbf{fmk_{k1}}[], \mathbf{fmk_{k2}}[], \ldots, \mathbf{fmk_{kn}}[])\}$: fuzzy multikeyword set;

(ix) $\mathbf{BSI_{WC}}$: BTree wild card fuzzy searchable index;

(x) $\mathbf{BSI_E}$: encrypted BTree wild card fuzzy searchable index of BSI_{WC};

(xi) **SPP**: search path pattern of multiple keywords;

(xii) $\mathbf{MKT_W}$: encrypted multikeyword trapdoor search request;

(xiii) $\mathbf{MFILE_{ENC}}$: set of encrypted multiple files matching search request;

(xiv) **MFILE**[]: decrypted multiple files of $\mathrm{MFILE_{ENC}}$;

(xv) **MSR**: multikeyword search request.

4.2. Searchable Encryption. Searchable encryption [13] is a cryptographic technique where the data users search the encrypted searchable index by the following steps.

(i) The encrypted tokens in the searchable index have the pointers to encrypted files. Token symbols are the encrypted keyword.

(ii) If the requested token found a match in the searchable index, then it extracts the file pointer without decryption.

(iii) If the token is not found in the searchable index, then it returns the null file pointer.

The two types of searchable encryption are symmetric and asymmetric (public key based) searchable encryption [14, 15]. In Symmetric Searchable Encryption, the data owner who outsources the encrypted index and data and the server that searches the data share the same secret keys. The efficiency of SSE is high since it uses symmetric cryptographic methods such as block ciphers, pseudorandom functions, and hash functions. The disadvantage of SSE is that the server has high probability to learn about the owner data and keywords. In Asymmetric Searchable Encryption [16, 17], the data owner outsources the index that is encrypted by the user's public key. The keyword trapdoor is created by the user's private key. So only the authorized users can request the search from the server. The advantage of ASE is that it supports conjunctive or disjunctive keywords searches. The disadvantage of SSE is that it suffers from KGA.

4.3. Edit Distance. Edit distance is the method of quantifying the similarity of the two strings. The edit distance (S_1, S_2) between two strings S_1 and S_2 is the smallest number of operations necessary to change one string to another. The three primitive operations are as follows:

(i) insertion: inserting one character into the string;

(ii) deletion: deleting one character from the string;

(iii) substitution: changing one character to another in the string.

The edit distance of the two strings in our system is analysed by dynamic programming. By dynamic programming strategy, the edit distance $\mathrm{ed}(x, y)$ of any two strings "x" and "y" is defined as follows and refer to Algorithm 1.

Assume the strings are $x[0, 1, \ldots, i-1]$ and $y[0, 1, \ldots, j-1]$.

If $x, y = 0$, then $\mathrm{ed}(x, y) = 0$.

If $y = 0$, then $\mathrm{ed}(x, 0) = i$.

If $x = 0$, then $\mathrm{ed}(0, y) = j$.

If $x \neq y$, then $\mathrm{ed}(x, y) = \min\{\mathrm{ed}(i-1, j) + 1, \mathrm{ed}(i, j-1) + 1, \mathrm{ed}(i-1, j-1) + \mathrm{diff}(x[i-1], y[j-1])\}$.

Note that if $x = y$ then $\mathrm{diff}(x, y) = 0$ else $\mathrm{diff}(x, y) = 1$.

Examples for $\mathrm{ed}(S_1, S_2) = 1$ are as follows:

insertion: S_1 = "SAMI" and S_2 = "SAMIY";

deletion: S_1 = "SAMI" and S_2 = "SAM";

substitution: S_1 = "SAMI" and S_2 = "SAME".

```
Input: Two strings x, y where x = x[0, ..., i − 1], y = y[0, ..., j − 1]
Output: Minimum edit distance
(1)  Declare the integer variables length1, length2, ed[ ][ ], i, j;
(2)  Declare the character variables char1, char2;
(3)  Declare the integer variables substitution, insertion, deletion, minimum;
(4) length1 = size of x;
(5) length2 = size of y;
(6) ed[ ][ ] = new int[length1 + 1][length2 + 1];
(7)  for (i = 0 to length1)
(8)        ed[i][0] = i;
(9)  end for // loop i
(10) for (j = 0 to length2)
(11)        ed[0][j] = j;
(12) end for // loop j
(13)  for (i = 0 to length1)
(14)        char1 = x.charAt(i);
(15)           for (j = 0 to length2)
(16)              char2 = y.charAt(j);
(17)                 if (char1 = = char2) then
(18)                    ed[i + 1][j + 1] = ed[i][j];
(19)                 else
(20)                 {
(21)                    substitution = ed[i][j] + 1;
(22)                    insertion = ed[i][j + 1] + 1;
(23)                    deletion = ed[i + 1][j] + 1;
(24)                    minimum = (substitution > insertion ? insertion: substitution);
(25)                    minimum = (deletion > minimum ? minimum: deletion);
(26)                    ed[i + 1][j + 1] = minimum;
(27)                 }
(28)                 end if;
(29)           end for // loop j
(30) end for // loop i
(31)    return ed[length1, length2]
```

Algorithm 1: EditDistance($x[0, \ldots, i-1]$, $y[0, \ldots, j-1]$).

```
Input: Search keyword key
Output: OUTFILE_w matching the keyword w
(1)  if key = FKS_i then
(2)       return OUTFILE_i
(3)  else if edit(key, FuzzyKS_i) ≤ edit) // edit is the user defined edit distance value
(4)       return OUTFILE_i
(5)  else
(6)       print "File Not Found"
(7)  end if
(8)  return OUTFILE_w
```

Algorithm 2: Fuzzy(key, $\text{FuzzyKS}_{\text{ed}=1}$).

4.4. Fuzzy Keyword Set. Since different users have various typing behaviors, they may misspell the keywords. So the fuzzy keyword set is formed to effectively utilize the data. Fuzzy keyword set can be created by wild card based technique, k-gram based technique, and symbol tree based technique. For example, fuzzy keyword set for "W = HEN" with edit distance = 1 is as follows:

$\text{FKS}_{\text{ed}=1}$ = {(A ⋯ Z)HEN, H(A ⋯ Z)EN, HE(A ⋯ Z)N, HEN(A ⋯ Z), EN, HN, HE, (A ⋯ Z)EN, H(A ⋯ Z)N, HE(A ⋯ Z), HEN}.

Total number of keywords = 186.

4.5. Fuzzy Keyword Search. For the set of M data files $D = \{D_1, D_2, \ldots, D_M\}$ with the predefined set of $KW = \{kw_1, kw_2, \ldots, Kw_n\}$, the fuzzy keyword search fuzzy (w, $\text{FKS}_{\text{ed}=1}$) is as shown in Algorithm 2.

4.6. Wild Card Based Technique. Wild card based technique is used to create storage efficient wild card based fuzzy keyword set. We use the wild card "#" character to represent the positions of three edit distance operations such as insertion, deletion, and substitution thereby creating tiny fuzzy keyword set. For example, the wild card based fuzzy keyword

set for w = "HEN" with edit distance ed = 1 is $FKS_{HEN,1}$ as follows:

$FKS_{HEN,1}$ = {#HEN, H#EN, HE#N, HEN#, EN, HN, HE, #EN, H#N, HE#, HEN}.

Total number of keywords = 11.

The length of fuzzy keyword set is $\mathbf{L} = ((\mathbf{2n+1}) * \mathbf{26}) + \mathbf{n} + \mathbf{1}$. The length of wild card based fuzzy keyword set is $\mathbf{L} = (\mathbf{2n} + \mathbf{1}) + \mathbf{n} + \mathbf{1}$.

4.7. K-Gram Based Technique. K-gram based technique is used to create k-gram based fuzzy keyword set for the predefined gram value k. The keywords of k-gram based fuzzy keyword set is the subset of keyword set. For example, the k-gram based fuzzy keyword set for w = "HEN" with gram value k = 1 is $FKS_{K=1}$ as follows:

$FKS_{K=1}$ = {KY, SY, KY}.

Total number of keywords = 3.

4.8. Assumptions of ADKEFS. Before we start our framework design, we have the following assumptions on our proposed scheme Asymmetric Classifier Multikeyword Fuzzy Search *ACMFS*.

(i) We assume that the cloud storage server concentrates on servicing more customers and not to leave partnership of their customers from the business.

(ii) Here we assume that the authorization is suitably done between the data owner and data users.

(iii) Data owner creates users private and public key pair.

(iv) We assume the wild card based fuzzy multikeyword set FMKS of multikeyword set MKW contains the original keyword as the first component.

(v) We assume that each file has multiple keywords and it is possible for a keyword to be the same for multiple files.

5. Implementation of Asymmetric Classifier Multikeyword Fuzzy Search (ACMFS)

Here we discuss our proposed scheme in detail with algorithm for all the functions involved.

5.1. Function Definitions of ADKEFS. The following functions are implemented to optimize the searching on the cloud storage server and to achieve the effective data utilization.

Functions on data owner are

(i) CreateKeyPairsForUser($SECRET_1$, $SECRET_2$),

(ii) EncryptMultiKeywordDataFile(PUB_{KEY}[]),

(iii) CreateWildCardFuzzyMultiKeywordSet(MKW[][], edit),

(iv) CreateBTreeWildcardFuzzySearchableIndex(FMKS[][][]),

(v) EncryptBTreeWildcardFuzzySearchableIndex (BSI_{WC}, PUB_{KEY}[]).

Functions on cloud storage server are

(i) ExtractMultipleFileUsingPattern(SPP[]).

Functions on classifier search server are

(i) SearchBTreeWildCardFuzzySearchableIndex(BSI_E, MKT_W[][]).

Functions on data user are

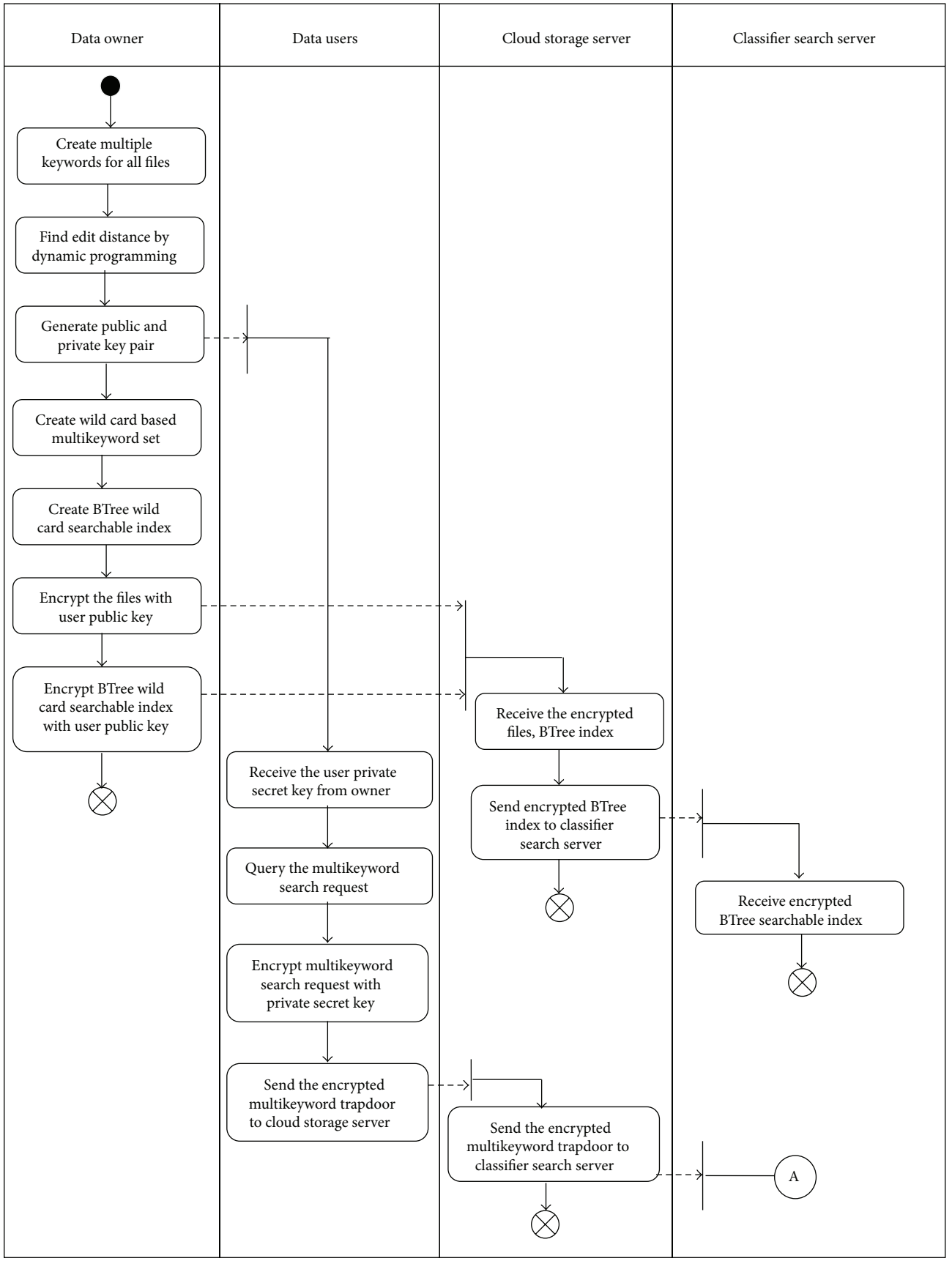
(i) ViewDecryptedMultiKeywordFile($PRIV_{KEY}$[], $MFILE_{ENC}$).

5.2. Overall Framework of Asymmetric Classifier Multikeyword Fuzzy Search (ACMFS). Our proposed method Asymmetric Classifier Multikeyword Fuzzy Search has classifier search server that creates the search path pattern for all the keywords of the encrypted set of K data files. Data owner creates encrypted BTree wild card fuzzy searchable index for the fuzzy multikeyword set and is outsourced to the cloud storage server. This overcomes the problem of typos and representation inconsistencies behaviour of the data users. The overall conceptual description of Asymmetric Classifier Multikeyword Fuzzy Search is shown in Figure 2 as an activity diagram. Please refer to Algorithm 3 for the pseudo-code.

5.3. Key Generation Algorithm. Here we use RSA public key algorithm for generating public and private key. Here this takes two secret keys $SECRET_1$ and $SECRET_2$ which is predefined by the data owner where both secret keys $SECRET_1$, $SECRET_2 = \{0, 1\}^*$. Algorithm 4 is executed by the data owner to generate public and private key pair. Data owner sends the user private key as private secret key which is used by the data users to create keyword trapdoor and for decrypting the file.

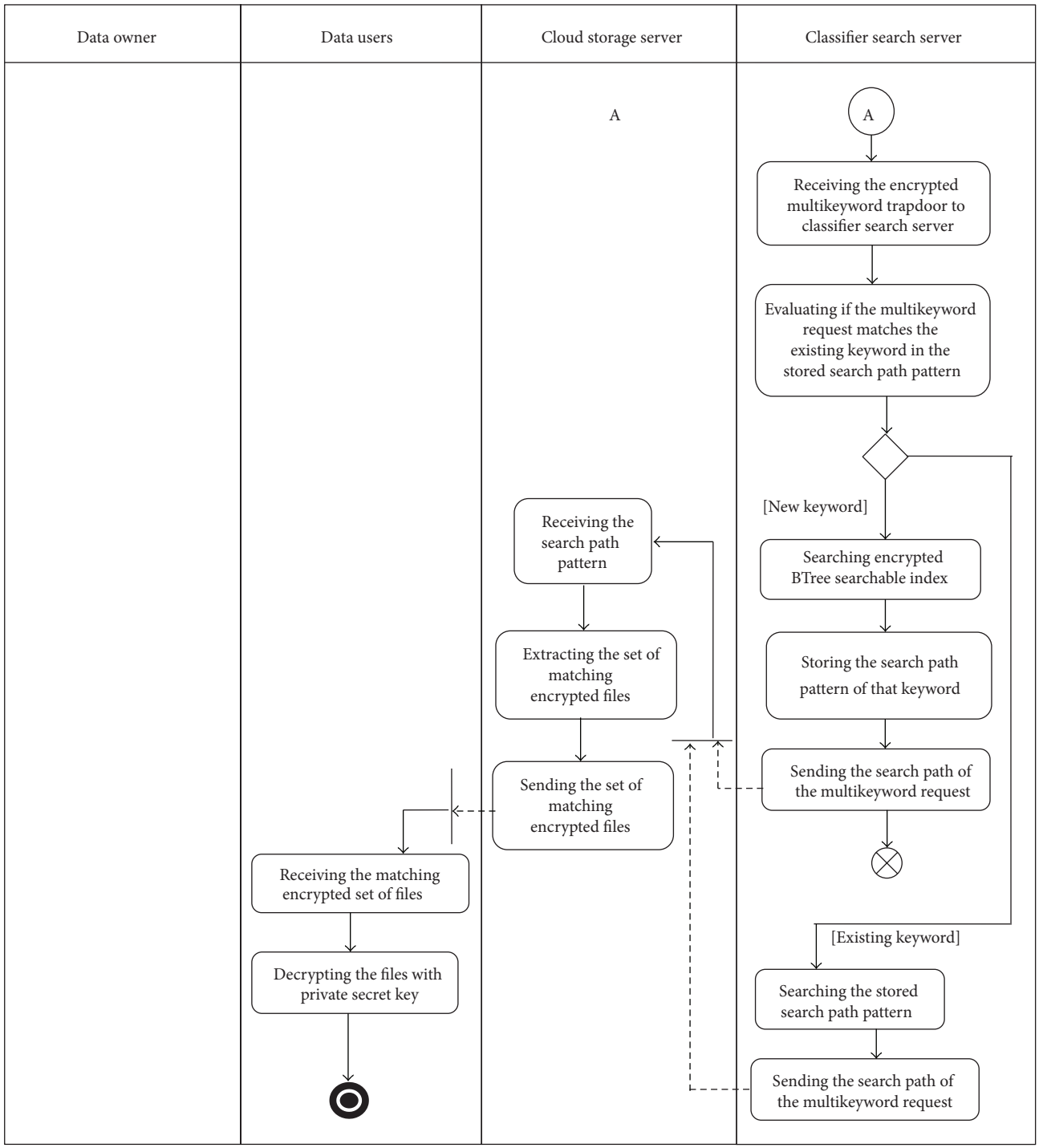
5.4. File Encryption Algorithm. Data owner executes Algorithm 5 to form encrypted set of K data files $DF = \{DF_1, DF_2, \ldots, DF_K\}$ that are encrypted with user's public key PUB_{KEY} and are outsourced to the cloud storage server.

5.5. File Decryption Algorithm. After receiving the search path pattern of the multiple keywords from the classifier search server, the cloud storage server extracts the set of encrypted files from DF and is sent to the data users. After receiving the encrypted files, the data user decrypts the files using private secret key by executing Algorithm 6.



(a)

Figure 2: Continued.



(b)

Figure 2: Activity diagram for Asymmetric Classifier Multikeyword Fuzzy Search.

5.6. Wild Card Based Fuzzy Multikeyword Set Algorithm. Data owner has MKW = {(mk11, mk12, ..., mk1n), (mk21, mk22, ..., mk2n), ..., (mkn1, mkn2, ..., mkkn)} a set of multiple keywords of K data files. Data owner creates storage efficient fuzzy multikeyword set FMKS = {(fmk11[], fmk12[], ..., fmk1n[]), (fmk21[], fmk22[], ..., fmk2n[]), ..., (fmkk1[], fmkk2[], ..., fmkkn[])} using wild card based technique with the predefined edit distance value. Data owner executes Algorithm 7 to form fuzzy multikeyword Set.

5.7. BTree Wild Card Fuzzy Searchable Index Algorithm. Data owner creates BTree wild card fuzzy searchable index BSIWC from fuzzy multikeyword set. Data owner executes

/* Variables Initialization */
(1) Initialize Secret Keys $\mathbf{SECRET_1}$ and $\mathbf{SECRET_2}$
(2) Predefine the edit distance "**edit**" value for Wildcard based Fuzzy Mulikeyword Set Creation
(3) Predefine Multiple Keywords for a single file and a keyword can be shared by multiple files.
/* KeyGeneration */
(4) DataOwner Creates User Public key and User Private Key pairs
(5) Call **CreateKeyPairsForUser(**$\mathbf{SECRET_1}$, $\mathbf{SECRET_2}$**)**
// User Public key and User Private Key
(6) Receive ($\mathbf{PUB_{KEY}}[\,]$,$\mathbf{PRIV_{KEY}}[\,]$)
(7) Encrypt the Kdata files using $\mathbf{PUB_{KEY}}[\,]$ User's Public key as $\mathbf{DF} = \{\mathbf{DF_1}, \mathbf{DF_2}, \ldots, \mathbf{DF_K}\}$
(8) Call **EncryptMultiKeywordDataFile(**$\mathbf{PUB_{KEY}}[\,]$**)**
(9) Receive the Encrypted file $\mathbf{DF}[\,] = \{\mathbf{DF_1}, \mathbf{DF_2}, \ldots, \mathbf{DF_K}\}$
(10) Send the Encrypted file $\mathbf{DF}[\,] = \{\mathbf{DF_1}, \mathbf{DF_2}, \ldots, \mathbf{DF_K}\}$ to the Cloud storage server
(11) Send the User's Private Key as **Private Secret Key** to the data users
(12) Data owner creates Wildcard basedFuzzy Multikeyword set for $\mathbf{MKW}[\,][\,]$
(13) Call **CreateWildCardFuzzyMultiKeywordSet(**$\mathbf{MKW}[\,][\,]$,**edit)**
(14) Receive $\mathbf{FMKS} = \{(\mathbf{fmk_{11}}[\,], \mathbf{fmk_{12}}[\,], \ldots, \mathbf{fmk_{1n}}[\,]), \ldots, (\mathbf{fmk_{n1}}[\,], \mathbf{fmk_{n2}}[\,], \ldots, \mathbf{fmk_{kn}}[\,])\}$
(15) Data ownerCreates B Ttree Wildcard searchable index $\mathbf{BSI_{WC}}$ for FMKS
(16) call **CreateBTreeWildcardFuzzySearchableIndex(**$\mathbf{FMKS}[\,][\,][\,]$**)**
(17) Receive B Ttree Wildcard searchable index $\mathbf{BSI_E}$
(18) Encrypts the B^+ tree wildcard searchable index $\mathbf{BI_{WC}}$
(19) call **EncryptBTreeWildcardFuzzySearchableIndex(**$\mathbf{BSI_{WC}}$, $\mathbf{PUB_{KEY}}[\,]$**)**
(20) Receive Encrypted BTree wildcard searchable index $\mathbf{BSI_E}$
(21) Send the Encrypted BTree wildcard searchable index $\mathbf{BSI_E}$ to the cloud storage server
(22) Cloud Storage server shares the BTree wildcard searchable index $\mathbf{BSI_E}$ to the Classifier Search Server
(23) Data User request Multikeyword search request **MSR**, and is encrypted using **Private Secret Key** to create MultiKeyword Trapdoor $\mathbf{MKT_W}$
(24) MultiKeyword Trapdoor $\mathbf{MKT_W}$ is sent to the cloud storage server
(25) Cloud Storage server send the $\mathbf{MKT_W}$ to the Classifier Search Server
(26) Classifier Search Server searches BTree wildcard searchable index $\mathbf{BSI_E}$
(27) If the Keyword $\mathbf{MKT_W}$ is the new keyword then
(28) call **SearchBTreeWildCardFuzzySearchableIndex(**$\mathbf{BSI_E}$, $\mathbf{MKT_W}[\,][\,]$**)**
(29) Keyword Classifier learn and Store the Search path pattern of $\mathbf{MKT_W}$
(30) Receive the Search path pattern of that keyword $\mathbf{MKT_W}$
(31) Send the Search path pattern of that keyword $\mathbf{MKT_W}$ to the cloud storage server
(32) else if $\mathbf{MKT_W}$ is already present in the Keyword Classifier then
(33) Classifier Search Server send the stored Search path pattern of $\mathbf{MKT_W}$ to cloud storage server
(34) End if
(35) Cloud Storage Server receives the Search path pattern of $\mathbf{MKT_W}$
(36) Cloud Storage Server extracts $\mathbf{MFILE_{ENC}}$ the encrypted multiple files of $\mathbf{MKT_W}$
(37) Call **ExtractMultipleFileUsingPattern(SPP**$[\,]$**)**
(38) Receive the set of Matching Encrypted files $\mathbf{MFILE_{ENC}}$
(39) $\mathbf{MFILE_{ENC}}$ is sent to theData user
(40) Data User receives the $\mathbf{MFILE_{ENC}}$ and decrypt the file using **Private Secret Key PSK**
(41) call **ViewDecryptedMultiKeywordFile(**$\mathbf{PRIV_{KEY}}[\,]$,$\mathbf{MFILE_{ENC}}$**)**
(42) Receive the Superset of matching Decrypted file Decrypted File $\mathbf{MFILE}[\,]$
(43) Now the user can view the file needed and large number of files viewed by the user represents the effective data utilization.

Algorithm 3: ACMFS( ) (Asymmetric Classifier Multikeyword Fuzzy Search).

Algorithm 8 in Appendix to create BTree wild card fuzzy searchable index BSIWC for the wild card based fuzzy keyword set FKS.

5.8. Encrypting BTree Fuzzy Searchable Index Algorithm. Data owner encrypts the BTree wild card fuzzy searchable index BSIWC using user public key and is outsourced to the cloud storage by executing Algorithm 9 to create encrypted BTree wild card fuzzy searchable index BSIE and is outsourced to the cloud storage server.

5.9. Searching Encrypted BTree Fuzzy Searchable Index Algorithm. The data user encrypts the multiple search keywords using the private secret key to create multikeyword trapdoor MKTW to the cloud storage server. The server sends the request MKTW to the classifier search server. The universal keyword classifier receives the request MKTW to check

Input: Two Secret keys $\mathbf{SECRET}_1$, $\mathbf{SECRET}_2$// Predefined SECRET_1, $\text{SECRET}_2 = \{0, 1\}^*$
Output: User Public and Private key pair $\mathbf{PUB}_{\mathbf{KEY}}[\,]$, $\mathbf{PRIV}_{\mathbf{KEY}}[\,]$
(1) Declare the integer variables S1, S2, S3, S4, private, public and key
(2) Assign S1 = SECRET_1, S2 = SECRET_2
(3) Find key = S1 ∗ S2
(4) Compute S3 = (S1 − 1) ∗ (S2 − 1), S4 = S3 − (S1 + S2 − 1)
(5) Pick random integer "public".Check gcd(public, S4) = 1
// gcd is Greatest Common Denominator
(6) Compute private = e^{-1} mod S4
(7) $\mathbf{PUB}_{\mathbf{KEY}}[\,]$ = {key, public} // Public key pair
(8) $\mathbf{PRIV}_{\mathbf{KEY}}[\,]$ = {key, public} // Private key pair
(9) return ($\mathbf{PUB}_{\mathbf{KEY}}[\,]$, $\mathbf{PRIV}_{\mathbf{KEY}}[\,]$)

Algorithm 4: CreateKeyPairsForUser(SECRET_1, SECRET_2) // *Key Generation Algorithm.*

Input: $\mathbf{PUB}_{\mathbf{KEY}}[\,]$ User Public key
Output: **Encrypted File DF**[ ]
(1) Declare integer variables key and public, i
(2) Assign key = $\mathbf{PUB}_{\mathbf{KEY}}[1]$, public = $\mathbf{PUB}_{\mathbf{KEY}}[2]$
(3) For i = 1 to k//K Data files
(4) Encrypt the file by computing DF[i] = $\text{Files}^{\text{public}}$ mod key
(5) end for // i loop
(6) return Encrypted file DF[ ] // Encrypted set of K data files $DF = \{DF_1, DF_2, \ldots, DF_K\}$

Algorithm 5: EncryptMultiKeywordDataFile($PUB_{KEY}[\,]$)//File Encryption Algorithm.

Table 1: Analysis of time taken for creating the wild card based fuzzy multikeyword set.

Number of users	Number of files	Time taken for creating wild card based fuzzy multikeyword set (ms)
10	300	420
12	350	450
14	400	453
16	450	530
18	500	650
20	550	655
22	600	710
24	650	730
26	700	810
28	750	980
30	800	1173

whether the request is coming for the first time. If the request is arriving for the first time, then the keyword classifier captures and stores the path of the MKTW by searching over the encrypted BTree wild card fuzzy searchable index BSIE by executing Algorithm 10 and sends the search path to the cloud storage server. If the request given by the user matches a previous request stored then it is a repeated multiple keyword. Then the classifier search server extracts the stored search path patterns of the repeated multikeyword from the universal keyword classifier and the search path is sent to the cloud storage server.

6. Implementation Results

6.1. Implementation Setup. The implementation of the proposed work was accomplished through Asymmetric Classifier Multikeyword Fuzzy Search (ACMFS) cloud data utilization service architecture using Jelastic PaaS LayerShift cloud storage provider which offers infrastructure, platform, and application as a service for the customers. The experimentation was carried out with the code programmed in JAVA for data owner, users, classifier search server, and cloud storage server. Microsoft SQL MYSQL 5.5.42 was enabled to act as the database for the proposed system. The simulation was performed with the setup of data owner, data users from our side, and classifier search server, cloud storage server on the Jelastic cloud storage. The data owner authenticates 100 users and defines the multikeyword set for each data files. Prior to evaluating the results, the data owner outsourced 1000 encrypted files to the Jelastic cloud storage. The data owner creates the wild card fuzzy multikeyword set FMKS for edit distance 1, 2 and BTree fuzzy searchable index BSI_{WC}. The data owner outsources the encrypted BSI_{WC} and 1000 encrypted files to Jelastic cloud storage server. The cloud storage now contains the encrypted 1000 files and encrypted BSI_{WC}. With this simulated setup, the authorized users are allowed to access the files in the cloud storage using their individual identity. The users are now allowed to access the

Input: $\mathbf{PRIV_{KEY}}[\,]$ Private Secret key, $\mathbf{MFILE_{ENC}}$ - Encrypted files
Output: Decrypted File**MFile**[ ]
(1) Assign PSK[1] = $\mathbf{PRIV_{KEY}}[1]$
(2) Assign PSK[2] = $\mathbf{PRIV_{KEY}}[2]$
// PSK – Private Secret Key
(3) For i = 1 to K
// K Data files
(4) MFILE[i] = $\mathbf{MFILE_{ENC}}^{\mathrm{PSK}[2]}$ mod PSK[1]
// Decrypting the file
(5) end for // i loop
(6) return Decrypted file MFILE[ ]

Algorithm 6: *ViewDecryptedMultiKeywordFile*($PRIV_{KEY}[\,]$, $\mathrm{MFILE_{ENC}}$) // File Decryption Algorithm.

Input: Multikeyword set MKW $=\{(mk_{11}, mk_{12}, \ldots, mk_{1n}), (mk_{21}, mk_{22}, \ldots, mk_{2n}), \ldots, (mk_{n1}, mk_{n2}, \ldots, mk_{nn})\}$
edit – Edit Distance
Output: FMKS$=\{(fmk_{11}[\,], fmk_{12}[\,], \ldots, fmk_{1n}[\,]), (fmk_{21}[\,], fmk_{22}[\,], \ldots, fmk_{2n}[\,]), \ldots, (fmk_{k1}[\,], fmk_{k2}[\,], \ldots, fmk_{kn}[\,])\}$
(1) Declare integer variable $i1$, $p1$, $q1$, edit
(2) Char *MultiKeywordFuzzy; Initialize FMKS[ ][ ] to be empty;
(3) if edit > 1 then
(4) call **CreateWildCardFuzzyMultiKeywordSet(MKW**[ ][ ],**ed-1)**
(5) end if
(6) if edit = 0 then
(7) FMKS = MKW
(8) else
(9) For $i1$ = 1 to length(MKW) do
(10) For $p1$ = 1 to length(FMKS[i][edit-1]) do
(11) For $q1$ = 1 to 2 ∗ length(FMKS[i][edit-1][$p1$]) + 1 do
(12) if $q1$ is odd value then
(13) MultiKeywordFuzzy = FMKS[i][edit-1[$p1$]]
(14) Insert "#" at ($q1$ + 1/2) position
(15) else
(16) Assign MultiKeywordFuzzy = length(Mkw[$p1$])
(17) Insert "#" at ($q1$/2) position
(18) end if
(19) if MultiKeywordFuzzy is not in FMKS[$i1$][edit-1] then
(20) Include MultiKeywordFuzzy in FMKS[$i1$][edit]
(21) end if
(22) end for // $q1$ loop
(23) end for // $p1$ loop
(24) end for // $i1$ loop
(25) end if
(26) return FMKS[ ][ ][ ] // Fuzzy MultiKeyword Set

Algorithm 7: CreateWildCardFuzzyMultiKeywordSet(MKW[ ][ ],edit).

files in the cloud storage by entering the multiple keyword search request.

6.2. Performance Analysis. The performance of the proposed method was evaluated taking into account the search time efficiency and data utilization from the Jelastic cloud storage by giving the multiple keyword search request from the data users with classifier search server. The experimental results obtained by ACMFS cloud data utilization system architecture are shown in Tables 1–5 and Figures 3–7. Table 1 shows the analysis of time taken for creating the wild card based fuzzy multikeyword set with different number of users and files and its analysis chart is shown in Figure 3. Here data owner predefines five keywords for each file.

Table 2 shows the analysis of data utilization efficiency for correct keyword in terms of number of files retrieved from the cloud storage and its analysis chart is shown in Figure 4.

Table 3 shows the analysis of data utilization efficiency for misspelled keyword in terms of number of files retrieved from the cloud storage and its analysis chart is shown in Figure 5.

Input: FMKS = {(fmk_{11}[], fmk_{12}[], ..., fmk_{1n}[]), (fmk_{21}[], fmk_{22}[], ..., fmk_{2n}[]), ..., (fmk_{k1}[], fmk_{k2}[], ..., fmk_{kn}[])}
WildCard Fuzzy Multikeyword Set
Output: **BSI_{WC}**: BTree Wildcard Fuzzy Searchable Index
(1) **Start AddNodeBTree**
(2) Declare the ChildLimit, NumberofChild, ChildrenValue inside the class BTreeNode
(3) Declare TreeHeight to denote height of the BTree, *i*, *j*;
(4) Declare the objects KeyNumber, KeyIndex, NodeValue, NextNode, RootNode for AddNode
(5) Declare KeyIndexNumber that denote the number of keyIndex and Nodevalue pairs in B Tree
(6) Declare the object ChildrenNodeValue for the class AddNode
(7) Define Constructor AddNode(KeyIndex, NodeValue, NextNode)
(8) this.KeyIndex = KeyIndex;
(9) this.NodeValue = NodeValue;
(10) this.NextNode =NextNode;
(11) End ConstrutorAddNode
(12) Initialize RootNode = new Node(0)
(13) for *i* = 1 to Number of keywords in FMKS[*i*] do
(14) for *j* = 1 to Number of Fuzzy Keywords in FMKS[*i*][*j*]
(15) call InsertNodeIntoBTree(RootNode, FMKS[*i*], FMKS[*i*][*j*], TreeHeight)
(16) Receive BTreeNode;
(17) KeyIndexnumber = KeyIndexnumber + 1;
(18) if (BTreeNode == NULL) then
(19) return NULL;
(20) end if
(21) Create two AddNode object as ChildNode
(22) ChildNode.childrenNodeValue [0] = new AddNode(RootNode.childrenNodeValue[0].KeyIndex, Null, RootNode);
(23) ChildNode. childrenNodeValue [1] = new AddNode(u. childrenNodeValue[0].KeyIndex, null, BTreeNode);
(24) RootNode = ChildNode;
(25) TreeHeight = TreeHeight + 1;
(26) **End AddNodeBTree**
(27) **Function AddNodeInsertNodeIntoBTree (RootNode, WFKS, WFKS, TreeHeight)**
(28) Declare the variable *i*, *j*, *k*, *m*, *n* for processing loop
(29) for *i* = 1 to Number of keywords in FMKS[*i*] do
(30) for *j* = 1 to Number of Fuzzy Keywords in FMKS[*i*][*j*]
(31) AddNodeCurrentNode = new AddNode(FMKS[*i*], FMKS[*i*][*j*], TreeHeight);
(32) if (TreeHeight == 0) then
(33) For *k* = 0 to RootNode. NumberofChild do
(34) if (WFKS[*i*] <RootNode. childrenNodeValue [*k*].FMKS[*i*]) then
(35) break;
(36) end if
(37) end for // *k* loop
(38) else
(39) for *m* = 0 to RootNode. NumberofChild do
(40) if ((*m* + 1 == RootNode. NumberofChild) || FMKS[*i*] <RootNode. childrenNodeValue [*m* + 1].FMKS[*i*]) then
(41) AddNodeNewNode = InsertNodeIntoBTree (RootNode. childrenNodeValue[*m*++].NextNode, FMKS[*i*], FMKS[*i*][*j*], TreeHeight -1);
(42) if (NewNode == NULL) then
(43) return null;
(44) end if
(45) CurrentNode. FMKS[*i*] =NewNode.childrenNodeValue[0]. FMKS[*i*];
(46) CurrentNode.NextNode = NewNode;
(47) break;
(48) end if
(49) end for // *m* loop
(50) for *n* = 0 to RootNode. NumberofChild do
(51) RootNode. childrenNodeValue [*n*] = RootNode. childrenNodeValue[*n* − 1];
(52) RootNode. childrenNodeValue [*j*] = CurrentNode;

Algorithm 8: Continued.

```
(53)            RootNode. NumberofChild = RootNode. NumberofChild + 1;
(54)         if (RootNode. NumberofChild < ChildLimit) then return null;
(55)            else    return SplitNodeInBTree(RootNode);
(56)         end if
(57)      end for // n loop
(58)     end for // j loop
(59) end for // i loop
(60) End Function AddNodeInsertNodeIntoBTree
(61)  Function SplitNodeInBTree(RootNode)
(62)     Declare the variable i for processing loop
(63)     AddNodeSplitNode = new AddNode (ChildLimit/2);
(64)     RootNode. NumberofChild = ChildLimit/2;
(65)     for i = 0 to ChildLimit/2 do
(66)        SplitNode. childrenNodeValue [i] = RootNode. childrenNodeValue[childLimit/2 + i];
(67)       return SplitNode;
(68)     end for // i loop
(69) End Function SplitNodeInBTree
```

Algorithm 8: CreateBTreeWildCardFuzzySearchableIndex(FMKS[][][]).

```
Input: B+ tree fuzzy searchable index BSI_WC
       PUB_KEY[ ] - user's public key
Output: Encrypted BTree WildCard Fuzzy searchable Index BSI_E
(1)   Declare integer variables key1 and public1
(2)   Assign key1 = PUB_KEY[1]
(3)   Assign public1 = PUB_KEY[2]
(4)    Find the number of elements "Enum" in BSI_WC
(5)    For i = 1 to Enum do
            /* Encrypt the fuzzy keyword present in each node */
(6)           BSI_E = BSI_WC[i] ^ public1 mod key1
(7)    end for // i loop
(8)    return Encrypted BSI_E
            // Encrypted BTreeWildCard Fuzzy searchable Index
```

Algorithm 9: EncryptBTreeWildcardFuzzySearchableIndex(BSI_{WC}, $PUB_{KEY}[\,]$).

```
Input: BSI_E – Encrypted BTree Wildcard Fuzzy Searchable Index
       MKT_W[ ]–Encrypted MultiKeyword Trapdoor
Output: Set of Matched Encrypted Files
(1) Declare the variable i, k for processing loop
(2)  Find the height of the B Tree index BSI_E and assign it to TreeHeight
(3)     AddNode childrenNodeValue = BSI_E. childrenNodeValue;
(4)     if (TreeHeight == 0) then
(5)        for i = 0 to NumberofChild do
(6)           if (MKT_W[i] = = childrenNodeValue [i]. MKT_W[i]) then
(7)              return (NodeValue) childrenNodeValue [i].NodeValue;
(8)           end if
(9)        end for // i loop
(10)     else
(11)     for k = 0 to BSI_E.NumberofChild do
(12)        if ((k + 1 = = BSI_E. NumberofChild)||(MKT_W[i] < childrenNodeValue [k + 1].KeyIndex)) then
(13)           return SearchBTreeWildCardFuzzyIndex (childrenNodeValue[k].NextNode,
                                                     MKT_W[k], TreeHeight – 1);
(14)        end if
(15)     end for // k loop
(16)   return NULL
(17) end if
(18) End SearchBTreeWildCardFuzzyIndex
```

Algorithm 10: SearchBTreeWildCardFuzzySearchableIndex(BSI_E, $MKT_W[\,][\,]$).

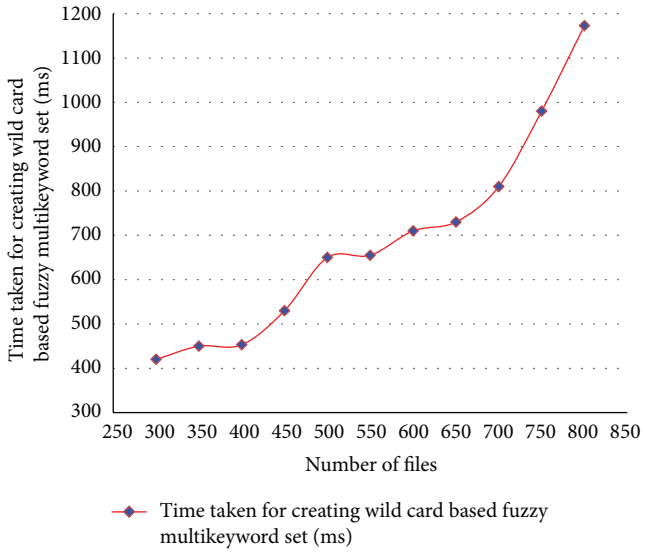


Figure 3: Time taken for creating wild card fuzzy multikeyword set for ACMFS.

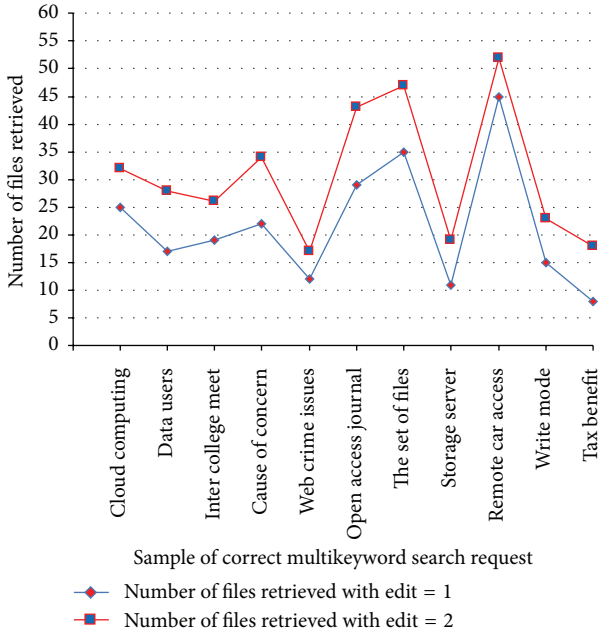


Figure 4: Data utilization efficiency for correct keywords in ACMFS.

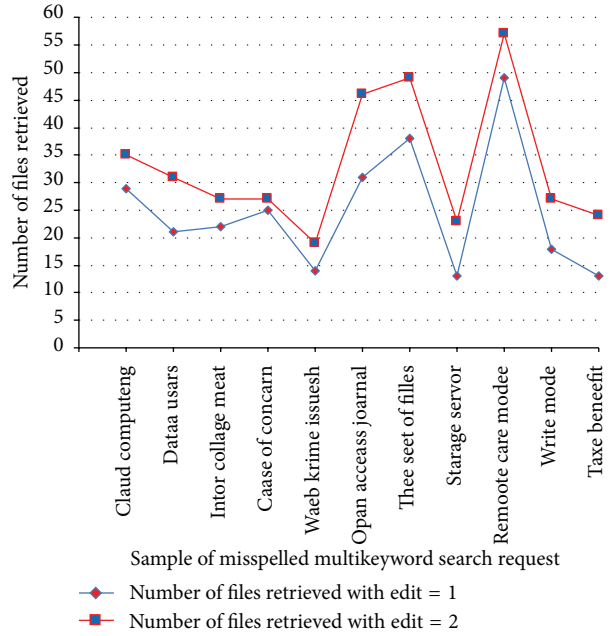


Figure 5: Data utilization efficiency for misspelled keywords in ACMFS.

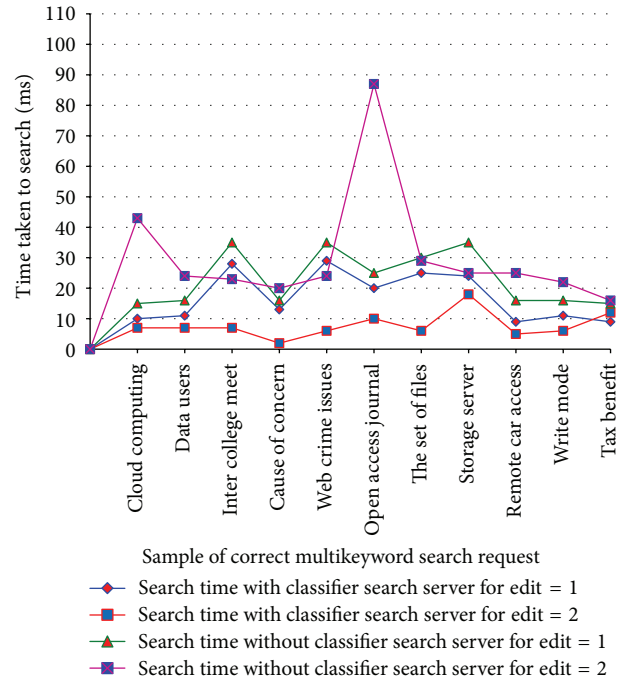


Figure 6: Search efficiency for correct keywords in ACMFS.

Table 4 shows the analysis of search time efficiency for correct multikeywords with and without classifier search server for edit 1, 2 and its analysis chart is shown in Figure 6.

Table 5 shows the analysis of search time efficiency for misspelled multikeywords with and without classifier search server for edit 1, 2 and its analysis chart is shown in Figure 7.

7. Conclusion

This work Asymmetric Classifier Multikeyword Fuzzy Search presented a method that can be successfully used to enhance data utilization and improved search efficiency for public cloud storage. Provided that the data owner stores the set of encrypted files in cloud storage, we showed that the user experience improved search time due to the presence of classifier search server which searches the BTree wild card fuzzy searchable index. The proposed system extracts the wild card fuzzy multikeyword set for the multiple keyword

Table 2: Analysis of data utilization efficiency for correct keywords.

Number of users	Number of files	Sample of correct multikeyword search request	Number of files retrieved with edit = 1	Number of files retrieved with edit = 2
10	300	Cloud computing	25	32
12	350	Data users	17	28
14	400	Inter college meet	19	26
16	450	Cause of concern	22	34
18	500	Web crime issues	12	17
20	550	Open access journal	29	43
22	600	The set of files	35	47
24	650	Storage server	11	19
26	700	Remote car access	45	52
28	750	Write mode	15	23
30	800	Tax benefit	8	18

Table 3: Analysis of data utilization efficiency for misspelled keywords.

Number of users	Number of files	Sample of misspelled multikeyword search request	Number of files retrieved with edit = 1	Number of files retrieved with edit = 2
10	300	Claud computeng	29	35
12	350	Dataa usars	21	31
14	400	Intor collage meat	22	27
16	450	Caase of concarn	25	27
18	500	Waeb krime issuesh	14	19
20	550	Opan acceass joarnal	31	46
22	600	Thee seet of filles	38	49
24	650	Starage servor	13	23
26	700	Remoote care accesh	49	57
28	750	Writee modee	18	27
30	800	Taxe beneefit	13	24

Table 4: Analysis of search efficiency for correct keywords with and without classifier search server.

Number of users	Number of files	Sample of correct multikeyword search request	Search time with classifier search server for edit = 1 (ms)	Search time with classifier search server for edit = 2 (ms)	Search time without classifier search server for edit = 1 (ms)	Search time without classifier search server for edit = 2 (ms)
10	300	Cloud computing	10	7	15	43
12	350	Data users	11	7	16	24
14	400	Inter college meet	28	7	35	23
16	450	Cause of concern	13	2	16	20
18	500	Web crime issues	29	6	35	24
20	550	Open access journal	20	10	25	87
22	600	The set of files	25	6	30	29
24	650	Storage server	24	18	35	25
26	700	Remote car access	9	5	16	25
28	750	Write mode	11	6	16	22
30	800	Tax benefit	9	12	15	16

Table 5: Analysis of search efficiency for misspelled keywords with and without classifier search server.

Number of users	Number of files	Sample of misspelled multikeyword search request	Search time with classifier search server for edit = 1	Search time with classifier search server for edit = 2	Search time without classifier search server for edit = 1	Search time without classifier search server for edit = 2
10	300	Claud computeng	17	17	158	47
12	350	Dataa usars	28	13	98	32
14	400	Intor collage meat	9	29	39	32
16	450	Caase of concarn	21	29	50	16
18	500	Waeb krime issuesh	11	30	78	47
20	550	Opan accass joarnal	25	31	34	46
22	600	Thee seet of filles	20	26	38	32
24	650	Starage servor	23	25	49	32
26	700	Remoote care accesh	13	15	20	31
28	750	Writee modee	11	10	28	47
30	800	Taxe beneefit	8	9	17	31

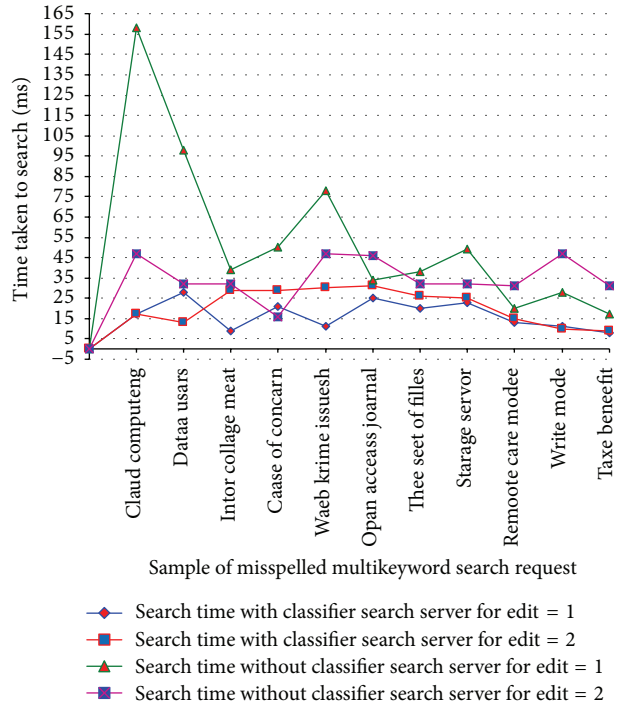


Figure 7: Search efficiency for misspelled keywords in ACMFS.

search request resulting in increased data utilization in terms of number of files retrieved for the corresponding users. The proposed system's performance is demonstrated with the advent of classifier search server which stores the pattern of search and helps reducing the search time for repeated multiple keyword search request. The classifier search server concept adds a new paradigm to cloud storage server serving several thousands of data owners and their users.

Appendix

See Algorithms 1 and 3–10.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] J. Geelan, "Twenty one experts define cloud computing," *Virtualization*, 2008, http://virtualization.sys-con.com/node/612375.

[2] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Proceedings of the Grid Computing Environments Workshop (GCE '08)*, pp. 1–10, Austin, Tex, USA, November 2009.

[3] http://searchsmbstorage.techtarget.com/feature/Understanding-cloud-storage-services-A-guide-for-beginners.

[4] Q. Liu, G. Wang, and J. Wu, "Secure and privacy preserving keyword searching for cloud storage services," *Journal of Network and Computer Applications*, vol. 35, no. 3, pp. 927–933, 2012.

[5] K. Ren, C. Wang, and Q. Wang, "Towards secure and effective data utilization in public cloud," *IEEE Transactions on Network*, vol. 26, no. 6, pp. 69–74, 2012.

[6] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pp. 79–88, Alexandria, Va, USA, 2006.

[7] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM '10)*, pp. 441–445, IEEE, San Diego, Calif, USA, March 2010.

[8] L. Sarga, "Cloud computing: an overview," *Journal of Systems Integration*, 2012, http://si-journal.org/index.php/JSI/article/viewFile/131/101.

[9] W. Zhou, L. Liu, H. Jing, C. Zhang, and S. Yao, "K-gram based fuzzy keyword search over encrypted cloud computing," *Journal of Software Engineering and Applications, Scientific Research*, no. 6, pp. 29–32, 2013.

[10] J. Wang, H. Ma, Q. Tang et al., "Efficient verifiable fuzzy keyword search over encrypted data in cloud computing,"

Computer Science and Information Systems, vol. 10, no. 2, pp. 667–684, 2013.

[11] P. Xu, H. Jin, Q. Wu, and W. Wang, "Public-key encryption with fuzzy keyword search: a provably secure scheme under keyword guessing attack," *IEEE Transactions on Computers*, vol. 62, no. 11, pp. 2266–2277, 2013.

[12] B. Wang, S. Yu, W. Lou, and Y. T. Hou, "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud," in *Proceedings of the 33rd Annual IEEE International Conference on Computer Communications (INFOCOM '14)*, pp. 2112–2120, IEEE, Toronto, Canada, April-May 2014.

[13] S. Kamara and K. Lauter, "Cryptographic cloud storage," in *Financial Cryptography and Data Security*, vol. 6054 of *Lecture Notes in Computer Science*, pp. 136–149, Springer, Berlin, Germany, 2010.

[14] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Advances in Cryptology—EUROCRYPT 2004*, vol. 3027 of *Lecture Notes in Computer Science*, pp. 506–522, Springer, Berlin, Germany, 2004.

[15] J. Baek, R. Safavi-Naini, and W. Susilo, "Public key encryption with keyword search revisited," in *Computational Science and Its Applications—ICCSA 2008*, vol. 5072 of *Lecture Notes in Computer Science*, pp. 1249–1259, Springer, Berlin, Germany, 2008.

[16] S.-T. Hsu, C.-C. Yang, and M.-S. Hwang, "A study of public key encryption with keyword search," *International Journal of Network Security*, vol. 15, no. 2, pp. 71–79, 2013, http://ijns.jalaxy.com.tw/contents/ijns-v15-n2/ijns-2013-v15-n2-p71-79.pdf.

[17] Y. Zhao, H. Ma, X. Chen, Q. Tang, and H. Zhu, "A new trapdoor-indistinguishable public key encryption with keyword search," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, vol. 3, no. 1-2, pp. 72–81, 2012.