# IGSK: Index Generation on Split Keyword for Search over Cloud Data

Raghavendra S*, Girish S*, Geeta C M*, Rajkumar Buyya†, Venugopal K R*, S S Iyengar‡ and L M Patnaik§
*Department of Computer Science and Engineering, University Visvesvaraya College of Engineering,
Bangalore University, Bangalore, India. e-mail: raghush86@gmail.com.
†Cloud Computing and Distributed Systems (CLOUDS) Lab, Department of Computing and Information Systems,
The University of Melbourne, Australia.
‡Department of Computer Science and Engineering, Florida International University, USA
§Indian Institute of Science, Bangalore, India.

***Abstract*—Storage as a Service (Saas) of cloud computing has become an alternative option for data owners of various organizations to store their data into the cloud. Usually sensitive data is encrypted to achieve data security and then it is outsourced into cloud. Many traditional search schemes allow data user to search over encrypted cloud data through keywords and retrieve the files of interest selectively. In this paper, we propose an efficient approach for keyword search over encrypted cloud data. The main contribution of this paper involves index generation method for keywords by using split factor. The keywords are stored in wildcard based technique within the index tree that is stored securely with low storage cost. Extensive experimental results on real-time data sets shows that the proposed solution is effective as well as efficient in Index generation and storage cost.**

***Index Terms*—Keyword Search, Data Privacy, Searchable Encryption, Cloud Computing, Index Generation.**

## I. INTRODUCTION

The explosive growth of data in the age of Information Technology has led to acute storage and maintanence problem. Cloud computing helps customers to store data in the cloud on pay on-demand basis, so that data owners need not worry about data storage space and maintenance [1]. Data owners may not have faith on Cloud Service Providers (CSPs), hence they encrypt their data and then outsource the data into the cloud, so that other users cannot understand the encrypted data. It is not just the problem of data storage, but the important thing is keyword search on encrypted text rather than on plain text. User can download the data into the local machine from the cloud space, decrypt and then search over plain text. Evolutionary approach for index generation [2] [3], utilizes more bandwidth and practically difficult as it degrades the performance. Hence, the challenge is to search the keyword over encrypted data within the cloud storage itself.

As cloud computing is easily accessible, the sensitive data being stored or outsourced into the cloud, like banking sector records, profiles of social networking sites, data of public sector like army, data of government research organizations, medical records [4], [5] and so on needs protection. The fact is, that data stored in the cloud server is not safe even though Cloud Service Providers (CSPs) use firewall mechanism for data security. CSPs may share the data of one user with another user who has also stored the data in the same cloud server or some employee belonging to CSP may access the information stored in cloud server for unauthorized purpose or to leak somebody's data to another user illegally. So, it is always recommended for data owners to encrypt the data and then outsource into the cloud. The encryption of data makes the effective use of data utilization, but the challenge is effective retrieving of relevant files among the large number of files that are stored in the cloud using keyword based search. The user can retrieve only the interested files using keyword search technique. Considering the large number of data users and a large number of outsourced encrypted data files into the cloud, the keyword search problem is challenging as it has to meet the requirements of performance, system stability and scalability.

As most of the information stored by various organizations is in textual format, all the applications have to support similarity keyword search on the data stored. The user might have used the synonym of some keyword in various places in the file. So, it is important to retrieve the synonym of a keyword being searched along with the exact keyword matching. For example, in a file, the term $college$ is stored in some documents and in some other files $institution$ is used in the place of $college$. When the user searches for a $college$, it has to also retrieve the files which contain $institution$ to the user so that user finds more interesting and relevant results. Though the similarity search and synonym search is possible in plain text, it is also necessary and a challenging problem for searching over outsourced encrypted cloud data due to privacy and security reasons. Ranked search [6] [7] is useful, as it sends back only the relevant data and avoids network traffic. This is quite important as most of the users use $pay - as - you - use$ cloud parameter. Moreover for privacy preserving, these ranking operations should not leak any information related to keyword. The ranking system should support multiple keyword search as well as synonym keyword search to improve the accuracy of the search result.

*Motivation:* Once the data is stored in cloud, the most important operation is to search. Even though the data is accessible only to a few authorized users, who perform search operations frequently, search operation should preserve the privacy of users as well as of data and it should also return efficient results. To achieve the privacy of data during search, many methods or techniques for privacy-preserving have been studied. Most of the work focus on single keyword and multi-keyword search [8], [9], while few works focus on fuzzy keyword search [10], [11]. Few techniques have also been proposed for similarity search on text [12] and on images [13]. The ranked search scheme helps the users to find out the most relevant documents.

*Contribution:* A flexible searching technique has been proposed in this paper which is efficient that supports fuzzy keyword, multi-keyword and synonym based keyword search. This protocol generates index using inverted indexing where the keyword is mapped to documents. This inverted index scheme provides the technique for scoring the search results. If a number of keywords being searched map to a large number of individual documents, then that is considered as a relevant document. The use of balanced binary tree improve the search efficiency. The relevant documents can be retrieved by traversing the tree. This search scoring technique uses Term Frequency-Inverse Document Frequency for weighing the results.

The contribution of this paper is summarized as follows:

1) Index Generation on Split Keyword(IGSK) has been proposed, which supports fuzzy based multi-keyword search over encrypted cloud data. When the user enters the keyword to search from the cloud data, the user obtains the fuzzy keyword matching in addition to the files which contains the synonym of few predefined keywords.
2) IGSK fulfills the functionality of secured keyword search which maintains keyword privacy; relevant files based on fuzzy keyword and synonym keyword search are retrieved.
3) IGSK ensures that search results using synonym based keyword searches over encrypted cloud data is authenticated by using index-based tree structure.
4) Extensive experimental result shows the effectiveness and efficiency of the IGSK scheme.

*Organisation:* The rest of the paper is organized as follows: Section II presents the reviews of the features of related work. Section III briefly describes the necessary background for the techniques used in this paper. The design goals are explained in Section IV. Section V describe the overview of proposed multi-keyword ranked search scheme which supports synonym query and our proposed scheme. Performance analysis for index time construction, storage cost of index and search time and security analysis are presented in Section VI. Section VII contains the conclusions.

## II. Related Works

Fuzzy keyword search are investigated in [14], [15], [16], [17], [18], [19]. Xu *et al.,* [20] takes linear time to store the searchable cipher-text as keywords and resists keyword guess attack but unaffordable for large database. Tuo *et al.,* [21] propose a semantically secure fuzzy keyword search scheme using the bloom filter which translates the keyword into attribute set and uses independent hash functions to map the elements to some random number using Bilinear mapping technique. This method extends from fuzzy identity encryption scheme to fuzzy keyword search scheme but does not support multi-keyword search and synonym search. Chuah *et al.,* [14] introduced a bedtree index tree construction and storage cost is efficient compared to the symbol-based trie-traversed based and listing based approach. Wildcard-based fuzzy set construction technique is used in [22], [23], [15], [19]. Jie *et al.,* [15] used dictionary based schemes to remove unwanted keywords and results in relatively small constructed index. Wang *et al.,* [16] achieves similarity search for $top-k$ ranked keywords and it uses Keyword Fingerprint Extraction which converts a string into a fingerprint vector. The kNN encryption provides two tier of protection for keywords being searched but does not support for synonym search. Fu *et al.,* [17] used Vector space model to construct the index for document; a balanced binary tree-based index structure is used for searching the keyword. Synonyms of predefined keywords are searched but requires increased storage space. Shekokar *et al.,* [18] supports privacy-preserving fuzzy keyword search to achieve effective usage of encrypted data stored remotely in cloud. Indexing technique is not used for mapping the keywords and takes more search time. In [19] Weighted ranking algorithm has been used to compute the weight of each word in the fuzzy set, that involves high computation overhead.

## III. Background

Researchers have been working on finding out the efficient way of searching keyword over encrypted cloud data to perform the searchable encryption in cloud computing. Many search schemes have been proposed like fuzzy keyword search scheme which uses wildcard-based technique to generate fuzzy keyword sets. Other schemes use TF-IDF method to find out keyword weight, that can generate $top-k$ relevant data files. But, this supports single keyword search. A multi-keyword search scheme has been proposed in [24] based on vector space model which supports more accurate results as it stores the weight of each keyword. The authors [25] use searchable index tree which is a balanced binary tree where each internal node is stored with keywords. This scheme takes more space for index storage and it has to traverse a large number of nodes for searching the keyword. Moreover, this scheme does not support fuzzy search and semantic based search.

## IV. Problem Statement and System Model

### A. System Model

We consider the cloud computing architecture with three modules, i.e., Data Owner($DO$), Data User($DU$) and Cloud

Server $(CS)$ as shown in Figure 1. Data owner has a set of $n$ data files/documents $D = F_1, F_2, F_3, \ldots, F_n$ which are stored in the cloud server. All these files are encrypted and a set of encrypted files $ED = (EF_1, EF_2, EF_3, \ldots, EF_n)$ are generated. The data owner extracts the keywords from the data files $D$. The data owner then removes/filters the stop words from the extracted keywords to form a set $KW = k_1, k_2, \ldots, k_n$. Next, the data owner obtains the synonyms for keywords $k_1, k_2, \ldots, k_n$ and generates a searchable encrypted index $EI$ for keywords and its synonyms, the encrypted index $EI$ and a set of encrypted files $ED$ are outsourced into the cloud server. When the authorized user performs the search operation, he submits an input keyword for search which is considered as Query KeyWord $(QKW)$. The Data User gets the synonyms $QS_1, QS_2, \ldots, QS_n$ for $QKW$, then search for $QKW$ and its synonyms. The Cloud Server$(CS)$ returns the search results as per the following rules.

1) If the Query keyword matches exactly with any keyword stored in the index, the server returns the files which contain the keyword.
2) If any file contains any synonym of the QKW, the cloud server returns also those files.

### B. Design Goals

The design of our system model supports multi-keyword search over outsourced encrypted data in cloud with the following security and performance paradigms.

1) *Synonym Search* : Our search scheme supports fuzzy keyword search as well as synonym search for the input keyword and returns the files containing keyword or its synonyms
2) $Privacy - Preserving$ : Our method is designed to meet the privacy challenge and prevents a cloud server and other cloud users from reading any information from any data files or from index being stored
3) $Efficiency$: The above functionalities are achieved with low storage, low network traffic and with low computation and search time.

### C. Tree Based Structure:

Searchable index is a balanced binary tree. The index tree EI is built from the set of data files $D = (F_1, F_2, F_3, \ldots, F_n)$ . The tree is built using the following procedure, which is expressed as *generateIndex(D)* as follows:

1) A leaf node in a tree is generated for each document/data file $F_i$ in $D$, which stores the file identifier $F_i$ and the index list.
2) Futher, the tree is constructed using postorder traversal with all the leaf nodes generated in first step. Each internal node of index tree contains an linked list, where each list stores the details of keywords in fuzzy/wildcard based format where $k_i$ belongs to $F_i$.
3) The linked list is generated in each internal node. If $k_{sf}$ belongs to $F_i$, where $k_{sf} = (k_{sf}*)$, it adds the length of * and file $F_i$ into the list; characters of * in $k_i$ is also encrypted and added into the list. Similarly, a list node for each keyword is created.

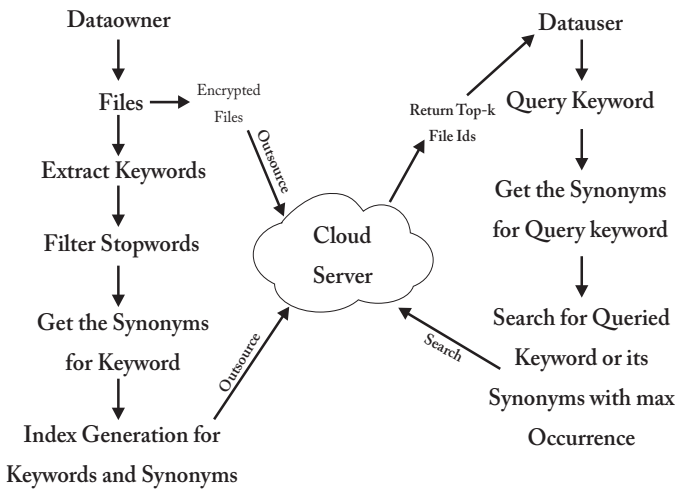


Fig. 1. Index Generation on Split Keyword System Architecture

TABLE I
NOTATIONS

| *Symbols* | *Definition* |
|---|---|
| $D$ | The plain-text document collection to be outsourced as a set of $n$ data files $D = (F_1, F_2, F_3, \ldots, f_n)$. |
| $ED$ | A set of encrypted documents collection to be outsourced as a set of $n$ data files $ED = (EF_1, EF_2, EF_3, \ldots, EF_n)$. |
| $EI$ | Encrypted Index. |
| $MF$ | Merge Factor. |
| $F_i$ | The file identifiers $F_i$ to locate uniquely the actual file. |
| $KW$ | The extracted distinct keywords from the document collection $D$, denoted as a set of $m$ keywords $KW = (k_1, k_2, \ldots, k_m)$. |
| $QS_i$ | Synonyms for each extracted distinct keyword, denoted as $(QS_1, QS_2, \ldots, QS_m)$. |
| $k_{sf}*$ | Keyword denoted as wildcard format. |
| $k_{sf}$ | Keyword with first sf number of characters. |
| $sf$ | Split factor to divide the keywords. |
| $k_{n-sf}$ | Remaining characters of keyword after splitting it. |
| $QKW_{sf}$ | Query keyword with first sf number of characters. |
| $QKW_{n-sf}$ | Remaining characters of query keyword after splitting it. |
| $R_i$ | Set of $k_{sf}$ for many keywords where $k_{sf}$ is same. |
| $k_i$ | Individual keyword. |
| $QKW$ | User interested Queried keyWord. |

### D. Tree based search:

The sequential search method for keywords in input search is as follows: Procedure starts from the root node and then it searches for an internal node, it checks with $QKW_{sf}$ and $k_{sf}$ in the linked list. If both matches, then it searches to match for remaining characters of $k_i$ inside the list. If it is found, searching stops in the subtree, otherwie, it continues to search in the child nodes. When it traverses the node, it gets the file $Id$ and the number of occurrences of the keyword in each file. In this search method, it traverses lower number of nodes as it stores $k_i$ in the wildcard based format and each file $id$ $F_i$

**Algorithm 1:** IGSK: Index Generation on Split Keyword

**input** : set of data files $D = F_1, F_2, \ldots, F_n$
**output** : A encrypted index $EI$

**Initialization Phase**;
Create index for each document separately using generateIndex$(D)$ method;
Merge the group of indices based on Merge Factor $MF$;
Create a stack for indices;
Create index for all the documents as explained in section 5.1;
Push new index being created into stack;
Assume merge factor $MF = 5$;
$n = \infty$;
**while** $(max\_size \geq 1 \&\& max\_size \leq n)$ **do**
  read current;
  **if** *there are $MF$ number of index with $max\_size$ docs on top of stack* **then**
    Pop $MF$ number of indexes from stack;
    Merge them into one index;
    Push the merged one into stack;
  **else**
    Break;
  **end**
**end**
$max\_size* = MF$;

of that $k_i$ occurrence in the same list and also because it uses inverted index structure.

## V. Proposed work

This section gives the overview of the IGSK scheme. An efficient multi-keyword ranked search scheme using synonyms is designed as follows:

1) $Initialization$: This phase is executed by $DO$ to initialize the scheme. It generates secret key $sk$ for encrypting the data before outsourcing into cloud.
2) *Index generation*: The Index phase is executed by $DO$ to build the index using $sk$ and a set of distinct keywords $k_i$ of data files $D$ and outputs the index..
3) *Query generation*: The query function is generated by $DU$ to generate the query out of the keyword being given for search as input. It splits the keyword $QKW$ and its synonym $QS_i$ based on the split factor $sf$ and generates query.
4) $Search$: The search phase is executed by $CS$ to search for the files $F_i$ which contains the keyword $QKW$ or its synonyms $QS_i$. It takes query and index as input and returns the set of files where the keyword and its synonyms are present

### A. Steps in the Algorithm IGSK

1) The $DO$ generates the secret key for encrypting the data files before they are outsourced into the cloud.
2) In this procedure, A tree based index structure is built for fuzzy keyword set which enables multi keyword search. The $DO$ reads all the data files $D$ and builds a set of distinct keyword for each file. In addition, a set of file Identifier for all files $i$ built and then outsourced to $CS$. The text documents can be expressed as follows.

$$\begin{aligned} F_1 &= k_1^1, k_2^1, \ldots, k_{n-1}^1, k_n^1 \\ F_2 &= k_1^2, k_2^2, \ldots, k_{n-1}^2, k_n^2 \\ &\vdots \\ F_m &= k_1^m, k_2^m, \ldots, k_{n-1}^m, k_n^m. \end{aligned}$$

*1) Extracting Synonyms:*

$$\begin{aligned} F_1 &= k_1^1, s_1^1, \ldots k_2^1, s_2^1, \ldots, k_{n-1}^1, s_{n-1}^1, k_n^1, s_n^1 \\ F_2 &= k_1^2, s_1^2, \ldots k_2^2, s_2^2, \ldots, k_{n-1}^2, s_{n-1}^2, k_n^2, s_n^2 \\ &\vdots \\ F_m &= k_1^m, s_1^m, ..k_2^m, s_2^m, ....., k_{n-1}^m, s_{n-1}^m, k_n^m, s_n^m. \end{aligned}$$

The fuzzy keyword set generated here is based on the wildcard. Index is created for each document by splitting the keyword based on the split factor $sf$, which divides the keyword $k_i$ into two tokens $k_{sf}$ and $k_{n-sf}$, where $n$ is the length of $k_i$. The indexed linked lists stored in each node contains the list for keyword starting with $k_{sf}$. The linked list is denoted as $k_{sf}, n-sf, F_i, k_{n-sf}$.

*2) Generating Index for individual files:*

$$\begin{aligned} IF_1 :&= R_i^1 \left( \left(l_{n-i}^1, ER_{n-1}^1\right) \ldots \left(l_{n-i}^N, ER_{n-1}^N\right) \right); \\ &= R_i^2 \left( \left(l_{n-i}^1, ER_{n-1}^1\right) \ldots \left(l_{n-i}^N, ER_{n-1}^N\right) \right); \\ &\vdots \\ &= R_i^n \left( \left(l_{n-i}^1, ER_{n-1}^1\right) \ldots \left(l_{n-i}^N, ER_{n-1}^N\right) \right); \end{aligned}$$

where $R_i + ER_{n-i} = K_i$.

After creating the index for each document, merging of index is done using the merge factor $MF$. Initially, all the individuals index are stored into stack. Then MF number of indexes are extracted and merged as below.

*3) Merging index files:*

$$\begin{aligned} I &= \{R_i^1\{\left(l_{n-i}^1, F_n, ER_{n-1}^1\right) \ldots \left(l_{n-i}^N, F_n, ER_{n-1}^N\right)\}; \\ &= R_i^2\{\left(l_{n-i}^1, F_n, ER_{n-1}^1\right) \ldots \left(l_{n-i}^N, F_n, ER_{n-1}^N\right)\}; \\ &\vdots \\ &= R_i^m\{\left(l_{n-i}^1, F_n, ER_{n-1}^1\right) \ldots \left(l_{n-i}^N, F_n, ER_{n-1}^N\right)\}\}; \end{aligned}$$

where

$$F_n = \{F_1, F_2, ..., F_n\}, i \leq j \leq n, K_i \in F_i.$$

$$T = R_i^j\{(l_{n-i}^j, F_n, ER_{n-1}^j)\}$$

The merged index is pushed onto the stack and the merging of indexes is continued till $MF$ number of indexes are available.

3) *Query generation*: When the keyword $QKW$ is given for search, the $QKW$ is split into two tokens based on splitting factor $sf$ used in index generation like $QKW_{sf}$ and $QKW_{n-sf}$. Then the split tokens are encrypted and query set containing $(QKW_{sf}, n-sf, QKW_{n-sf})$ and its synonyms are formed and sent to the cloud server.

4) If the linked list in any node in the index tree matches with $QKW_{sf}$, then it searches inside the linked list for $n-sf$. If a internal node in the linked list matches with $n-sf$, it proceeds to search inside that node only for finding out the match for $QKW_{n-sf}$. If there exists $QKW_{n-sf}$ inside the list, it returns the list of file $Ids$ where the keyword QKW and its synonyms.

### B. Algorithm Example:

In the example, we have assumed two files $F_1$ and $F_2$ with various keywords. Initially, the index is created for first file. Each keyword is divided into two sub keywords based on the split factor. The keyword $university$ is divided into $univ$ and $ersity$. While generating the index, all keywords starting with $univ$ are combined together and index is formed as shown in table II i.e., *university, universe and universal*. Let $n$ be the length of keyword, $sf$ the split factor, the index for keyword $k_i$, first $i$ added to the initial characters of keyword, then the length of remaining characters i.e., $n-sf$ is added. The file $id$ and the remaining characters are added after encrypting. If there are a number of keywords starting with the same characters, then index for each $k_i$ is separated by the delimiter and are added based on length of $n-sf$ in the ascending order.

After the index is created for the index file, the index for the second file is generated in the same manner and both are merged. Check if the index for the keyword is already existing in the index file. If it exists, then only the file $id$ has to be added. In the above example, file $id2$ is added for $university$ and the index for the new keyword $computer$ is also added. In a similar manner, we have to continue adding the new keywords for new files.

## VI. Performance

Our proposed scheme is verified by implementing the search scheme on the cloud server. We have used real time data sets. Our experiment includes a user and a server. The search scheme is implemented using Java language in Windows machine with Dual CPU running at 1.46 GHz and the encrypted collection of files are stored on the commercial public cloud, Amazon cloud services like $S3$ (simple storage service). The performance is evaluated for index generated time,index storage space,search time and security analyis.

TABLE II
EXAMPLE FOR ALGORITHM

| | Number of Files | |
|---|---|---|
| | $F_1$ | $F_2$ |
| Extracted Keywords | University<br>Universe<br>Universal<br>Optimize<br>Optical | University<br>College<br>Optical<br>Computer |

| Index Format | | | | |
|---|---|---|---|---|
| File Id | $R_i$ | n-sf | ER | frequency |
| $F_1$ | Univ | 6 | ersity | 4 |
| | | 4 | erse | 4 |
| | | 5 | ersal | 4 |
| | Opti | 4 | mize | 4 |
| | | 3 | cal | 4 |
| $F_2$ | Univ | 6 | ersity | 4 |
| | Opti | 3 | cal | 4 |
| | Coll | 3 | ege | 4 |
| | Comp | 4 | uter | 4 |

| Merged Index File | | | | |
|---|---|---|---|---|
| $R_i$ | n-sf | ER | frequency | File Ids |
| Univ | 6 | ersity | 4 | $F_1, F_2$ |
| | 4 | erse | 4 | $F_1$ |
| | 5 | ersal | 4 | $F_1$ |
| Opti | 4 | mize | 4 | $F_1$ |
| | 3 | cal | 4 | $F_1, F_2$ |
| Coll | 3 | ege | 4 | $F_2$ |
| Comp | 3 | uter | 4 | $F_2$ |

### A. Index Generations Time

Figure 2 indicates the time complexity for generating the index which is directly proportional to the number of data files. Although, index generation is performed only once before outsourcing the data into cloud, this operation can be ignored. While generating the index, encryption of the keyword is an additional operation. As shown in Figure 2, the B-tree method [25] takes 12.64 secs to generate index for 10000 files whereas the proposed scheme costs 12.40 secs to perform the same operation on the same set of data files. The comparison shows that IGSK is already efficient at this stage. Similarly for 20000 files, B-tree requires 29.21 secs and the proposed IGSK scheme requires 27.84 secs. Both the methods grow linearly as the number of files also increments. It can be seen that the index time generation for the proposed scheme is more efficient than the B-tree scheme [25].

### B. Index Storage Space

The index file consumes less amount of storage space in the cloud environment. Data security is important rather than for storage space of index. Figure 3 indicates the size of index tree in both the schemes. The storage cost for 10000 files in B-Tree method is 7.5 MB whereas the proposed scheme costs just 6.33 MB. B-Tree scheme for 20000 files costs 14.43 MB, while the proposed IGSK scheme requries 12.41 MB for the index file. Extensive experiment shows that IGSK scheme takes less space for storage of index file

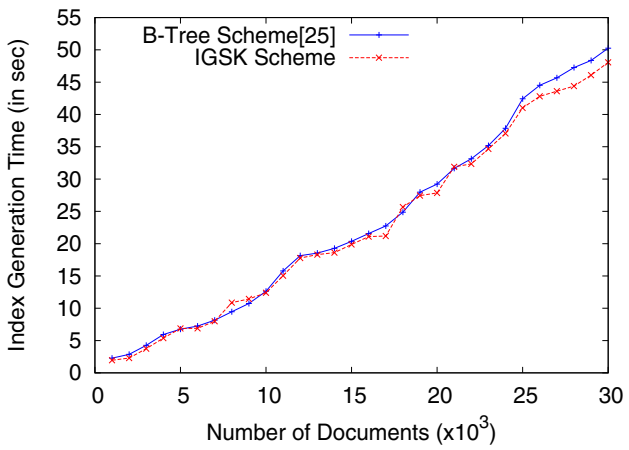


Fig. 2. Index Generation Time Over Number of Documents

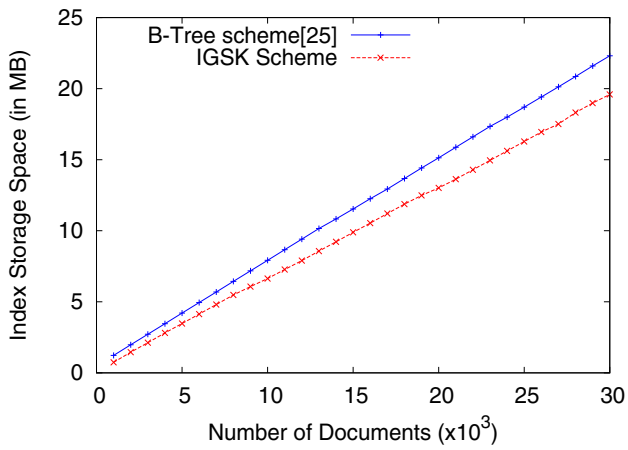


Fig. 3. Storage Cost Over Number of Documents

compared to B-tree method [25].

### C. Security Analysis

We evaluate the security of the proposed scheme by analyzing the security of data files $F_n$ that are stored in the cloud server as they should not be read or altered by any unauthorized data user. The data files $D$ are encrypted before outsourcing into cloud and the encrypted files do not contain any normal plain text. So, it is difficult to read and understand even if it is attacked by any unknown data user. Even though the index file $EI$ is also stored in the cloud server, the keywords in index are split into two using the split factor; then both are encrypted separately and outsourced into the cloud server. So it is not possible to decrypt the index file without the secret key. Even if the unauthorized user gets the secret key of the encryption, the unauthorized user cannot break the cipher text and find the data stored as the keywords are split. The file $Id$ is also encrypted and stored in the index while storing the keywords $k_1, k_2, \ldots, k_n$ in the index tree; the order of the keywords in the file is not stored in index. Thus, IGSK scheme provides the privacy of data.

## VII. Conclusions and Future Work

In this paper, we solve the problem of synonym and fuzzy based keyword index generation over the cloud data. The proposed IGSK scheme includes multi-keyword search with enhanced accuracy. The inverted index method used for indexing and wildcard based technique for indexing keywords, takes less time for searching. The TF*IDF technique is used for ranking mechanism. The IGSK scheme is simulated on real-time datasets to the verify efficiency of storage cost and security. It is observed that the IGSK scheme consumes less index generation time, lower storage space than B-tree scheme[25].

Further, we would like to explore synonym based search over encrypted cloud data which supports synonym queries, as the cloud user may gives synonym of keyword as input, instead of exact matching keywords due to lack of proper knowledge of data.

## References

[1] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. *Future Generation Computer Systems*, 25(6):599–616, 2009.

[2] P Deepa Shenoy, KG Srinivasa, KR Venugopal, and Lalit M Patnaik. Dynamic Association Rule Mining using Genetic Algorithms. *Intelligent Data Analysis*, 9(5):439–453, 2005.

[3] P Deepa Shenoy, KG Srinivasa, KR Venugopal, and Lalit M Patnaik. Evolutionary Approach for Mining Association Rules on Dynamic Databases. *Advances in Knowledge Discovery and Data Mining*, pages 325–336, 2003.

[4] S Vishwa Kiran, Ramesh Prasad, J Thriveni, KR Venugopal, and LM Patnaik. Cloud Enabled 3D Tablet Design for Medical Applications. In *in Proceedings of the 9th International Conference on Industrial and Information Systems (ICIIS)*, pages 1–6. IEEE, 2014.

[5] Vishwa Kiran, Ramesh Prasad, J Thriveni, KR Venugopal, and LM Patnaik. Mobile Cloud Computing for Medical Applications. In *in Proceedings of the Annual IEEE India Conference (INDICON 2014)*, pages 1–6. IEEE, 2014.

[6] S Raghavendra, C M Geeta, K Shaila, Rajkumar Buyya, K R Venugopal, S S Iyengar, and L M Patnaik. "MSSS: Most Significant Single-keyword Search over Encrypted Cloud Data". *in Proceedings of the 6th Annual Intrernational Conference on ICT: BigData, Cloud and Securit*, 2015.

[7] S. Raghavendra, S Girish, C. M. Geeta, Rajkumar Buyya, K. R. Venugopal, S. S. Iyengar, and L. M. Patnaik. MSIGT: Most Significant Index Generation Technique for Cloud Environment. In *12th IEEE India International Conference on $E^3$-$C^3$(INDICON 2015)*. IEEE, IEEE, 2015.

[8] Ning Cao, Cong Wang, Ming Li, Kui Ren, and Wenjing Lou. "Privacy-preserving Multi-keyword Ranked Search over Encrypted Cloud Data". *IEEE Transactions on Parallel and Distributed Systems*, 25(1):222–233, 2014.

[9] Wenhai Sun, Bing Wang, Ning Cao, Ming Li, Wenjing Lou, Y Thomas Hou, and Hui Li. "Verifiable Privacy-Preserving Multi-keyword Text Search in the Cloud Supporting Similarity-Based Ranking". *IEEE Transactions on Parallel and Distributed Systems*, 25(11):3025–3035, 2014.

[10] Jianfeng Wang, Hua Ma, Qiang Tang, Jin Li, Hui Zhu, Siqi Ma, and Xiaofeng Chen. "Efficient Verifiable Fuzzy Keyword Search Over Encrypted Data in Cloud Computing". *Journal of Computer Science and Information Systems*, 10(2):667–684, 2013.

[11] Bing Wang, Shucheng Yu, Wenjing Lou, and Y Thomas Hou. "Privacy-Preserving Multi-Keyword Fuzzy Search Over Encrypted Data in the Cloud". *in Proceedings IEEE INFOCOM*, pages 2112–2120, 2014.

[12] Cong Wang, Kui Ren, Shucheng Yu, and Karthik Mahendra Raje. "Achieving Usable and Privacy-Assured Similarity Search over Outsourced Cloud Data". In *in Proceedings IEEE INFOCOM*, pages 451–459. IEEE, 2012.

[13] Zhihua Xia, Yi Zhu, Xingming Sun, and Jin Wang. "A Similarity Search Scheme over Encrypted Cloud Images based on Secure Transformation". *International Journal of Future Generation Communication and Networking*, 6(6):71–80, 2013.

[14] M Chuah and W Hu. "Privacy-Aware Bedtree Based Solution for Fuzzy Multi-Keyword Search over Encrypted Data". *in Proceedings of the 31st International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 273–281, 2011.

[15] Wang Jie, Yu Xiao, Zhao Ming, and Wang Yong. "A Novel Dynamic Ranked Fuzzy Keyword Search Over Cloud Encrypted Data". *in Proceeding of the 12th International Conference on Dependable, Autonomic and Secure Computing (DASC)*, pages 91–96, 2014.

[16] Dongsheng Wang, Shaojing Fu, and Ming Xu. "A Privacy-Preserving Fuzzy Keyword Search Scheme over Encrypted Cloud Data". *in Proceedings of the 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, 1:663–670, 2013.

[17] Zhangjie Fu, Xingming Sun, Zhihua Xia, Lu Zhou, and Jiangang Shu. "Multi-Keyword Ranked Search Supporting Synonym Query over Encrypted Data in Cloud Computing". *in Proceedings of the IEEE 32nd International Performance Computing and Communications Conference (IPCCC)*, pages 1–8, 2013.

[18] Narendra Shekokar, Kunjita Sampat, Chandni Chandawalla, and Jahnavi Shah. "Implementation of Fuzzy Keyword Search over Encrypted Data in Cloud Computing". *Procedia Computer Science*, 45:499–505, 2015.

[19] Wei Zhou, Lixi Liu, He Jing, Chi Zhang, Shaowen Yao, and Shipu Wang. "K-Gram Based Fuzzy Keyword Search over Encrypted Cloud Computing". *Journal of Software Engineering and Applications*, 6(1):29–32, 2013.

[20] Peng Xu, Hai Jin, Qianhong Wu, and Wei Wang. "Public-Key Encryption with Fuzzy Keyword Search: A Provably Secure Scheme Under Keyword Guessing Attack". *IEEE Transactions on Computers*, 62(11):2266–2277, 2013.

[21] He Tuo and Ma Wenping. "An Effective Fuzzy Keyword Search Scheme in Cloud Computing". *in Proceedings of the 5th International Conference on Intelligent Networking and Collaborative Systems (INCoS)*, pages 786–789, 2013.

[22] Cong Wang, Qian Wang, and Kui Ren. "Towards Secure and Effective Utilization Over Encrypted Cloud Data". *in Proceedings of the 31st International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 282–286, 2011.

[23] Jin Li, Qian Wang, Cong Wang, Ning Cao, Kui Ren, and Wenjing Lou. "Fuzzy Keyword Search Over Encrypted Data in Cloud Computing". *in 2010 Proceedings IEEE INFOCOM*, pages 1–5, 2010.

[24] Wenhai Sun, Bing Wang, Ning Cao, Ming Li, Wenjing Lou, Y Thomas Hou, and Hui Li. "Privacy-Preserving Multi-Keyword Text Search in the Cloud Supporting Similarity-Based Ranking". *in Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, pages 71–82, 2013.

[25] Zhangjie Fu, Xingming Sun, Nigel Linge, and Lu Zhou. "Achieving Effective Cloud Search Services: Multi-keyword Ranked Search over Encrypted Cloud Data Supporting Synonym Query. *IEEE Transactions on Consumer Electronics*, 60(1):164–172, 2014.