Spring 2013

Semantic Search over Encrypted Data in Cloud Computing

Kam Ho Ho
San Jose State University

Semantic Search over Encrypted Data in Cloud Computing

A Written Report

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Computer Science

by

Kam Ho Ho

CS298 Written Report

May 2013

Final Version

The Designated Project Committee Approves the Project Titled

Semantic Search over Encrypted Data in Cloud Computing

by

Kam Ho Ho

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

San José State University

May 2013

Dr. Teng Moh	Department of Computer Science
Dr. Mark Stamp	Department of Computer Science
Dr. Frank Butt	Department of Computer Science

Abstract

Semantic Search over Encrypted Data in Cloud Computing

by Kam Ho Ho

Cloud storage becomes more and more popular in the recent trend since it provides various benefits over the traditional storage solutions. Along with many benefits provided by cloud storage, many security problems arise in cloud storage which prevents enterprises from migrate their data to cloud storage. These security problems induce the data owners to encrypt all their sensitive data such as social security number (SSN), credit card information, and personal tax information before they can be stored in cloud storage. The encryption approach may have strengthened the data security of cloud data, but it degrades the data efficiency because the encryption reduces the searchability of the data. Many schemes were proposed in recent researches which enable keyword search over encrypted data in cloud computing, and these schemes contain weaknesses which make them impractical when applying these schemes in real-life scenarios. In this project, we developed a system to support semantic search over encrypted data in cloud computing with three different schemes. The three schemes that we developed are “Synonym-Based Keyword Search (SBKS)”, “Wikipedia-Based Keyword Search (WBKS)”, and “Wikipedia-Based Synonym Keyword Search (WBSKS)”. Based on our experiment data, it demonstrated that the indexes created by our schemes are 95% smaller and reduced the average search time by 95% if compared to the schemes proposed previously. These improvements illustrated that our developed schemes are more practical than the former proposed schemes.

.

Table of Contents

List of Tables

List of Figures

Semantic Search over Encrypted Data in Cloud Computing

by Kam Ho Ho

I. Introduction

Cloud storage is becoming more and more popular in the recent trend as it provides many benefits over the traditional storage solutions. With cloud storage, corporations can purchase only the needed amount of storage from the cloud storage provider (CSP) to fulfill their storage needs instead of maintaining their own data storage infrastructures. They can rely on CSP to handle all data maintenance tasks such as backup and recovery. It also allows all data to be accessed remotely in order to streamline their operation among different locations. With all these benefits, companies can significantly reduce their operation cost by simply outsourcing their business data to cloud storage.

Beside these benefits that provided by the cloud storage, however, many security problems arise in cloud storage that prevent companies from migrating their data to cloud storage [7]. Due to the facts that cloud storage is usually hosted by third party provider other than the data owners and cloud storage infrastructure is usually shared among different users, data stored in cloud storage can be easily targeted by the masquerade attack [1, 8] and the insider data theft attack [9, 10]. These attacks threaten the data security and the data privacy of the stored data, as result, the data owners cannot rely on CSP to secure their confidential data. These attacks also induce the data owners to encrypt all their sensitive data such as the social security numbers (SSN), credit card information, and personal tax information before they can be saved in cloud storage. The encryption approach may have strengthened the data security of cloud data, but it has also degraded the data efficiency because the encryption will reduce the searchability of the data. Especially in the cloud computing environment, it is impractical for the user to download and decrypt the entire encrypted data from the remote cloud server before a search can occur. Therefore, an efficient scheme that supports search over encrypted data in cloud computing becomes very significant before many enterprises can take advantage of the cloud storage.

Many schemes were proposed in recent researches that enable keyword search over encrypted data in cloud computing. The most common approach of these schemes is indexing the keywords contain in each uploading data file to

create an index file. The index file will be uploaded along with the encrypted data files to the cloud storage for later search operation. Instead of indexing the actual keywords, authors in [3] proposed a scheme called “Wildcard-based Fuzzy Set Construction (WFSC)” to support fuzzy keyword search over encrypted cloud data by expanding each keyword into variations of the keywords with wildcard character based on a predefine edit distance value. The authors in [4] modified WFSC and proposed another scheme called “Dictionary-based Fuzzy Set Construction (DFSC)” that uses a dictionary to expand each keyword which based on a predefine edit distance value. Both WFSC and DFSC support fuzzy keyword search over encrypted data in cloud computing.

Even those WFSC and DFSC support fuzzy keyword search over encrypted cloud data, these schemes contain weaknesses that make them impractical in real-life scenarios. The main weakness of the previously proposed schemes is that they do not capture the true intention of the users that initiate the search command. These schemes build their index files based on the actual keywords extracted or variations of keywords that generated based on the predefine edit distance value. These indexes can only support search with keyword that are identical to the actual keyword or keywords that have very similar structures. For example, if the user tries to search with keyword “paper”, WFSC will only return data containing keywords such as “paper”, “page”, or “papers” which share similar word structure with keyword “paper”, but keywords such as “article” and “thesis” that share similar meanings with keyword “paper” will be ignored. Another example would be keywords like “steal” and “stole” which are basically the same word but with different tenses. This problem can occur frequently when the data users do not have precise knowledge over the encrypted cloud data. The users may predefine the edit distance value to a larger number to increase the range of the search result, but, at the same time, it will degrade the search quality because more unrelated keywords will be returned in the search result and degrade in search performance because the size of the index will increase. Due to these drawbacks, previously proposed schemes are impractical when the users do not have precise knowledge over the encrypted data.

In this project, we developed a system to support semantic search over encrypted data in cloud computing with three different schemes. The three schemes that we developed are called “Synonym-Based Keyword Search (SBKS)”, “Wikipedia-Based Keyword Search (WBKS)”, and “Wikipedia-Based Synonym Keyword Search (WBSKS)”.

The first scheme we developed is called “Synonym-based Keyword Search (SBKS)”. SBKS is an improved version of WFSC which uses the “Synonym Set Construction (SSC)” process to expand each keyword with synonyms of the keyword. Using the expanded keyword set, SBKS is able to improve the search quality by extending the search coverage to cover keywords that share similar meanings and reducing the number of unrelated keywords in the search result.

The second scheme we developed is called “Wikipedia-based Keyword Search (WBKS)”. WBKS adopted the Wikipedia Similarity Matching technique that was proposed recently to resolve the content-targeted advertising problem. WBKS uses a set of Wikipedia articles which we called the “Wikipedia Key Set (WKS)” as a reference point in between the data files and the search keyword. WBKS decides if a data file is relevant to a keyword by comparing the cosine similarity between the data file and the WKS with the cosine similarity between the keyword and WKS. A similarity score between the data file and the keyword will be generated from this comparison, and then it will be used to determine if the keyword is relevant to the data file. Unlike the schemes proposed previously, each index entry in WBKS is indexing each uploading data file instead of the extracted keywords. Due to this reason, the index created by WBKS is much smaller than the other schemes formerly proposed.

The third scheme we developed is called “Wikipedia-based Synonym Keyword Search (WBSKS)”. WBSKS is a hybrid between SBKS and WBKS. WBFKS takes advantage of the Wikipedia Similarity Matching technique to create index on data file level. It also takes advantage of SSC from SBKS to expand the search keyword with the synonyms to ensure the words with similar meanings will be taken into consideration for the search result. WBSKS inherits the advantages of both extended search coverage and the reduced index size benefits from SBKS and WBKS.

We have implemented ours system with these 3 schemes that we developed to support semantic search over encrypted data in cloud computing. We also implemented the WFSC that formerly proposed in [3] as benchmark. We have conducted different tests against each scheme with our own implementations to collect statistical test data for careful analysis. In storage analysis, it shows that the indexes created by our schemes are 95% smaller than the index created by WFSC with edit distance equaled to 2 (WFSC-ED2). In performance analysis, it shows that ours schemes have reduced the average search time by 95% compare to WFSC-ED2. In search quality analysis, we classified the search results and showed that 0% of keywords resulted from

SBKS searches are unrelated while 92% of keywords resulted from WFSC-ED2 searches are unrelated. In data security analysis, we examined ours schemes and showed that our schemes maintain the same level of security as WFSC if the shared key and the Wikipedia key set remain hidden. Ours analysis indicated that ours developed schemes are more efficient than the WFKS scheme in term of storage usage, search performance, and search quality while maintaining the same level of data security. These improvements show that our developed schemes are more practical than the WFSC.

The rest of this paper is organized as follows: In Section II, we will review the related literatures. In Section III, we will give a high level overview of the system we developed. In Section IV, we will provide a detailed description of the schemes we developed to support semantic search over encrypted data in cloud computing. In Section V, we will describe our test implementation. In Section VI, we will present our analysis over the collected test data. In Section VII, we will conclude this paper and give recommendations for future work.

II. Review of Related Literature

In this section, we will review some of the schemes that proposed previously to support keyword search over encrypted cloud data and we will review the Wikipedia Similarity Matching technique proposed previously to resolve the content-targeted advertising problems.

Keyword Search over Encrypted Cloud Data

Koletka and Hutchison in [2] proposed a unique data structure called Secure File Object (SFO) to enable keyword search over encrypted cloud data. When a data owner wants to upload a data file to the cloud storage, the client-side application will create and attach a SFO to the encrypted data file before uploading to the cloud storage. Each SFO contains information that describes the uploading data file. During SFO creation, the client-side application extracts unique keyword from the uploading data file and encrypts them to create a list of encrypted keywords that will be stored in the SFO. When a user wants to search for a specific keyword, the user will submit the keyword to the data owner and the data owner will compute the search capability by encrypting the keyword with the same key that used to generate the list of encrypted keywords in the SFO. The user can submit the returned search capability from the data owner to the cloud server. The cloud server will return the encrypted data file if the list of encrypted keywords in the SFO contains the search capability. This proposed scheme with

SFO is implemented by the authors to provide simple keyword search over encrypted cloud data.

The major drawback of the SFO scheme is that the scheme only supports keyword search using the exact keyword as it appears in the data file. If there are any typos of the keywords that used to generate the search capability, the cloud server will fail to locate the correct encrypted data file.

To overcome the drawback in [2], Li, Wang, et al in [3] proposed the "Wildcard-based Fuzzy Set Construction (WFSC)" scheme to enable fuzzy keyword search over encrypted cloud data. The key concept behind WFSC is maintaining an index that covers all possible variations of a keyword within a predefine edit distance. Instead of simply encrypting the keywords extracted from the data file, WFSC expands each extracted keyword into a set of modified keywords by inserting wildcard character into the keyword. The number of wildcard character used to modify the keyword is based on a predefine edit distance value. Table 1 shows the modified keyword set of the keyword "student" using the wildcard character '*' when the predefine edit distance value equals to 1. Each modified keywords in the set will be hashed with a secured hash function to create a trapdoor. The trapdoor will be appended by the encrypted information that describes the uploading data files that contain the keyword and the original keyword to form an index entry. The collection of index entries will form an index file and it will be uploaded to the cloud storage along with all the encrypted data files that addressed by the index file.

Table 1: Modified keyword set of keyword "student" when the predefine edit distance value equals to 1

Edit Distance	Modified Keyword Set
1	*student, *tudent, s*tudent, s*udent, st*dent, st*udent, stu*dent, stu*ent, stud*ent, stud*nt, stude*nt, stude*t, studen*, studen*t, student, student*

When a user needs to search for a specific keyword, the user will inject the keyword with wildcard character to compute the modified keyword set based on the predefine edit distance value and hash each modified keywords to create the trapdoor set. The trapdoor set will be submitted to the cloud server and the cloud server will search the index file and compare the trapdoor in each index entry with each trapdoor in the received trapdoor set. The cloud server will return

the matched encrypted index entries to the user and the user can decrypt the index entry to retrieve the information of the data files that contain the keyword.

There are several weaknesses in WFSC if the user tries to expand the search coverage by increasing the predefine edit distance value. The increase in the edit distance value will cause a huge increase in the size of the modified keyword set. Table 2 shows the modified keyword set of the keyword “student” using the wildcard character ‘*’ when the predefine edit distance value equals to 2. Our examples of keyword “student” show the size of the modified keyword set is 16 when the edit distance value equals to 1 and the size of the modified keyword set has increased hugely to 122 by simply increase the edit distance value to 2. The size of the index file will increase rapidly by increasing the edit distance value and the search performance will be degraded due to the increases in the index size. Furthermore, the quality of the search will also degraded due to more unrelated keywords can be returned in the result.

Table 2: Modified keyword set of keyword "student" when the predefine edit distance value equals to 2

Edit Distance	Modified Keyword Set
2	**student, **tudent, **udent, *s*tudent, *s*udent, *st*dent, *st*udent, *stu*dent, *stu*ent, *stud*ent, *stud*nt, *stude*nt, *stude*t, *studen*, *studen*t, *student, *student*, *t*dent, *t*udent, *tu*dent, *tu*ent, *tud*ent, *tud*nt, *tude*nt, *tude*t, *tuden*, *tuden*t, *tudent, *tudent*, s**dent, s**tudent, s**udent, s*t*dent, s*t*udent, s*tu*dent, s*tu*ent, s*tud*ent, s*tud*nt, s*tude*nt, s*tude*t, s*tuden*, s*tuden*t, s*tudent, s*tudent*, s*u*dent, s*u*ent, s*ud*ent, s*ud*nt, s*ude*nt, s*ude*t, s*uden*, s*uden*t, s*udent, s*udent*, st**dent, st**ent, st**udent, st*d*ent, st*d*nt, st*de*nt, st*de*t, st*den*, st*den*t, st*dent, st*dent*, st*u*dent, st*u*ent, st*ud*ent, st*ud*nt, st*ude*nt, st*ude*t, st*uden*, st*uden*t, st*udent, st*udent*, stu**dent, stu**ent, stu**nt, stu*d*ent, stu*d*nt, stu*de*nt, stu*de*t, stu*den*, stu*den*t, stu*dent, stu*dent*, stu*e*nt, stu*e*t, stu*en*, stu*en*t, stu*ent, stu*ent*, stud**ent, stud**nt, stud**t, stud*e*nt, stud*e*t, stud*en*, stud*en*t, stud*ent, stud*ent*, stud*n*, stud*n*t, stud*nt, stud*nt*, stude**, stude**nt, stude**t, stude*n*, stude*n*t, stude*nt, stude*nt*, stude*t, stude*t*, studen*, studen**, studen**t, studen*t, studen*t*, student, student*, student**

Liu, Zhu, et al in [4] modified WFSC from [3] and proposed another scheme called “Dictionary-based Fuzzy Set Construction (DFSC)”. Instead of injecting keyword with the wildcard character, DFSC uses a dictionary to pull in only the valid words that are within the range of the predefine edit distance to form the modified keyword set. The authors showed the size of the index file created by DFSC is much smaller than WFSC when the predefine edit distance increased. Even DFSC shows better storage usage than WFSC, DFSC inherited the search quality degradation problem because more unrelated keywords can still be pulled into the modified key set from the dictionary as edit distance value increases. Furthermore, DFSC does not support variations of newly invented words or keywords that contain multiple typos because they are words that cannot be found in the dictionary. This problem decreases the search coverage of DFSC and makes it less accurate than WFSC.

Beside the problem described above, the major weakness of the previously proposed schemes is that they do not capture user’s true intention of the search. The indexes created by both DFSC and WFSC are established using only the actual keyword and variations of keyword computed which based on the predefine edit distance value. These indexes can only support search with keyword that has very minor difference with the original keyword in term of word structure. These schemes do not support search with keyword that shares similar meanings with the original keyword that has a very different word structure. For example, when the user wants to search with keyword “paper”, WFSC will return data files that contain keyword such as “paper”, “page”, or “papers”, but it will ignore other keywords such as “article” and “thesis” that share similar meanings with keyword “paper”. The article and thesis may be what the user actually was searching for when he/she does not have exact knowledge of the uploaded data. Another example would be verbs like “steal” and “stole” which are basically the same word that would be ignored by the previously proposed schemes when the predefine edit distance value is small. This problem has made the previous proposed schemes impractical because they do not capture user’s true intention of the search.

Wikipedia Similarity Matching in Content Targets Advertising

The problem of content-targeted advertising is the problem to associate ads with a web page based on the content of the web page. The challenges in content-targeted advertising are that the wordings used in ads are usually more general than the web pages which typically very specific on a topic.

Ribeiro-Neto, Cristo, et al in [6] proposed a scheme called “impedance coupling strategies” to overcome the low keyword intersection problem between the ads and the web page. The idea behind is to add new keywords to the web page from pages that share common topic to increase the number of keywords intersecting between the web page and the ads. The expansion technique is referred as impedance coupling. The authors also purposed different strategies to match the ads with the expanded web page and showed the accuracy of the matching done with their scheme is higher than the traditional matching strategies.

Pak in [5] has purposed another scheme called “Wikipedia Matching (WM)” that takes advantage of the impedance coupling technique to resolve the content-targeted advertising problem. Instead of using pages that share common topic, WM select a set of Wikipedia articles to serve as a reference point between the ad and the page. Term frequency-inverse document frequency (TF-IDF) is being used to form the vector representation with the keywords for each page, ad, and Wikipedia articles. With the vector representations, WM computes the cosine similarity between each article to the page and each article to the ad. The resulted cosine similarities will be used to compute the Euclidean distance and the ad with the shorter distance to the page is say to has more relevance. The author has shown the quality of the resulted pairs using WM is more accurate than the traditional keyword matching technique and semantic matching technique.

Wu, Xu, et al in [11] have purposed a similar scheme called “Selective Wikipedia Matching (SIWI)” that uses Wikipedia articles as a reference point between the page and the ads. The authors showed how different set of Wikipedia articles being used as the reference point will affect the matching accuracy. The authors also showed the number of Wikipedia articles used as the reference point will have a direct effect on the matching accuracy.

III. System Architecture

In this section, we will give a high-level overview of our developed system architecture and detail descriptions for each component. Figure 1 shows the system architecture and all the major components of our developed system.

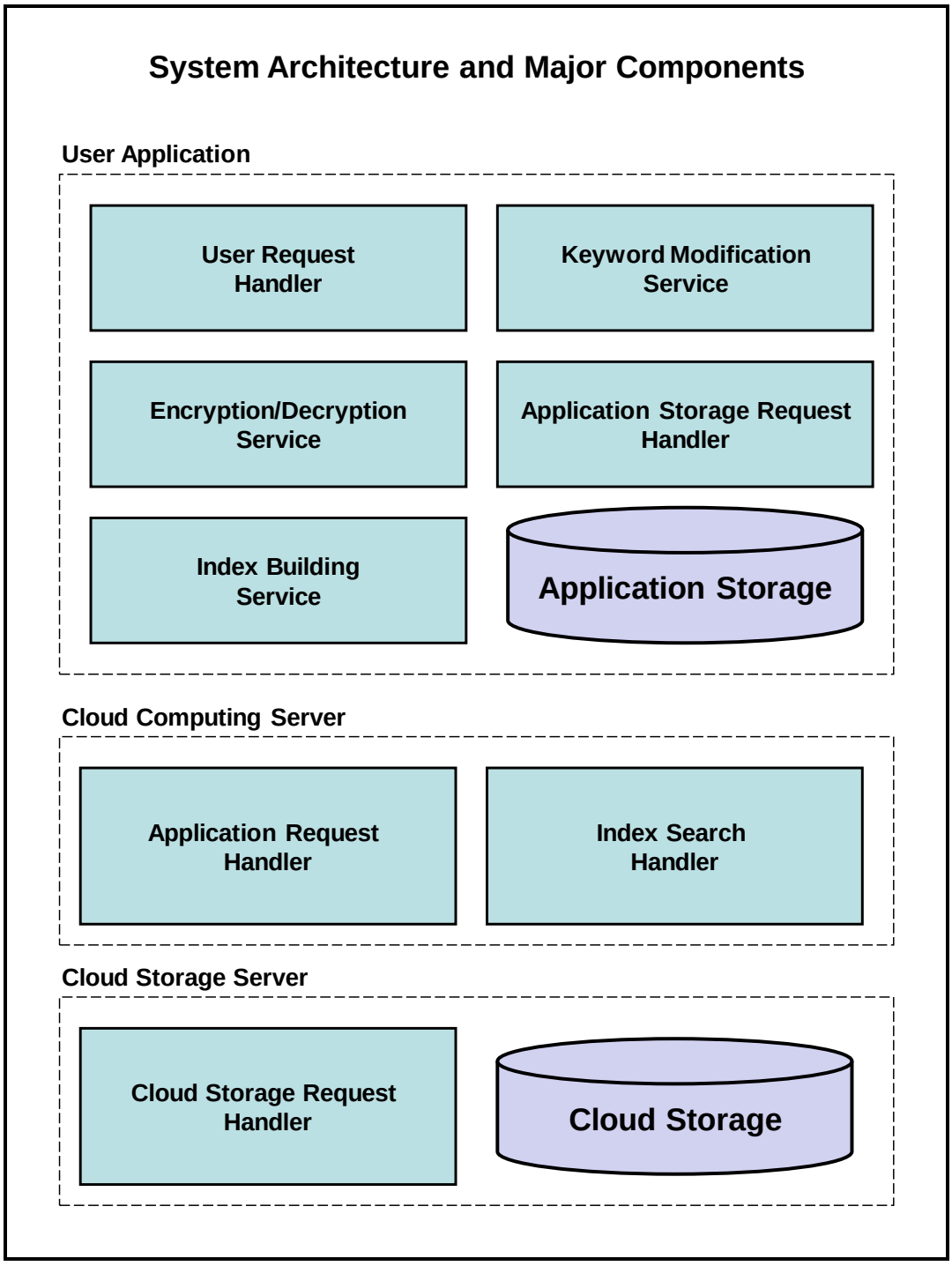

Figure 1: Overview of our system architecture

Our developed system includes an user application, a cloud computing server, and a cloud storage server. The user application is hosted and maintained by the user party and is considered as a trusted component while the

cloud computing server and the cloud storage server is hosted and maintained by a third party CSP and are considered as non-trusted component. Any communication channel between the user application and the cloud servers is also considered as non-trusted because it can be targeted by other attackers.

The user application in our system is acting as the interface to handle all communications between the user and the cloud computing servers. The user application contains a small amount of local storage to hold different dictionaries and Wikipedia Key Set (WKS) that needed for index creation and trapdoor creation. The user application is responsible to create the index and encrypt the data file when the data owner wants to upload the files to cloud storage. When the user needs to do a search with a specific keyword, the user application is responsible to modify the keyword into a trapdoor and submit the search to the cloud computing server. The user application is responsible to decrypt the returned index entries in order to retrieve the file information for the user when the cloud computing server returned from the search. Table 3 gives descriptions for each component in the user application.

Table 3: User application and its components

User Application	
Components	**Descriptions**
User Request Handler	Handle and process different user request such as UPLOAD, SEARCH, or DOWNLOAD
Encryption/Decryption Service	Perform encryption and decryption services such as secure hashing and symmetric-key encryption
Index Building Service	Extract keyword from the data file and create the index based on different schemes supported
Keyword Modification Service	Modify the keyword into the modify keyword set based on different schemes supported
Application Storage Request Handler	Handle internal request from the application to retrieve data stored in the local storage
Application Storage	Local storage used to store different dictionaries and Wikipedia Key Set that used by different schemes

The cloud computing server is acting as a controller which decides how the uploaded data file should be stored in the cloud storage server and retrieve the data file from the cloud storage server. The cloud computing server also performs the actual search operation using the trapdoor and the index file and returns the index entries that fulfill the search requirements. Table 4 gives descriptions for each component in the cloud computing server.

Table 4: Cloud computing server and its components

Cloud Computing Server	
Components	**Descriptions**
Application Request Handler	Handle requests from the user application and decide how the uploaded file should be stored in the cloud storage server
Index Search Handler	Perform search operation over the index file stored in the cloud storage based on different schemes supported

The cloud storage server is simply a storage server that used to store the uploaded data. There can be more than one cloud storage server in the system and the cloud computing server can be tuned to decide how the uploaded data files should be stored across different cloud storage servers. Table 3 gives descriptions for each component in the cloud storage server.

Table 5: Cloud storage server and its components

Cloud Storage Server	
Components	**Descriptions**
Cloud Storage Request Handler	Handle requests from the cloud computing server to store or retrieve the data in cloud storage
Cloud Storage	Storage used to store the uploaded data and index file

IV. Semantic Search Schemes

In this section, we will describe three schemes we developed to enable semantic search over encrypted cloud data. The three schemes are “Synonym-Based Keyword Search (SBKS)”, “Wikipedia-Based Keyword Search (WBKS)”, and “Wikipedia-Based Synonym Keyword Search (WBSKS)”.

Figure 2 shows an overview of our schemes for semantic search over encrypted data in cloud computing.

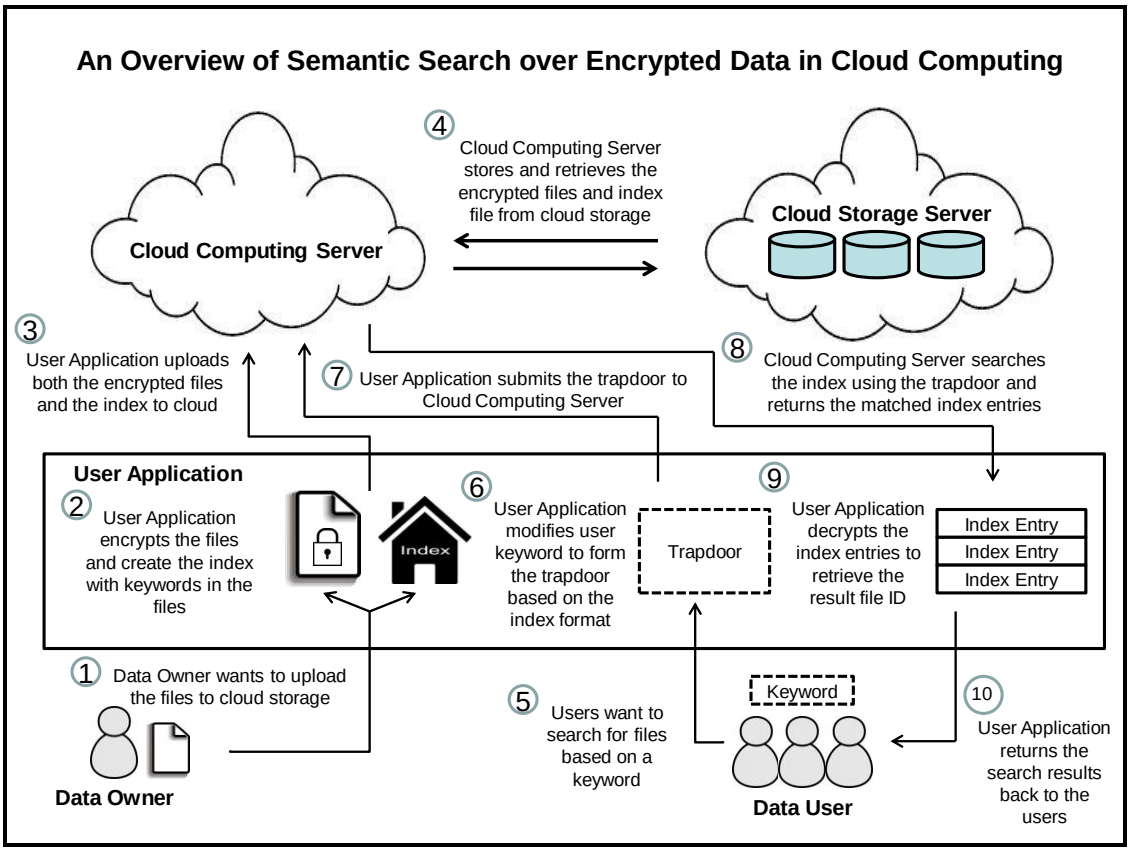

Figure 2: An overview of semantic search over encrypted data in cloud computing

Our developed schemes begin when the data owner wants to upload the data files to the cloud storage. The data owner will first submit the data files and a shared key to the user application. The user application will create an index file with different formats based on each developed scheme. After created the index file, the user application will encrypt the data files using a symmetric-key algorithm with the shared key from the data owner. Both the index file and the

encrypted data files will be uploaded together by the user application to the cloud computing server. The cloud computing server will store the index file and the encrypted data files in cloud storage server.

When the data users want to conduct a search with a specific keyword, they will submit the search keyword and the shared key to the user application. The user application will modify the search keyword to form the trapdoor with different formats based on each developed scheme. The user application will then submit the trapdoor to the cloud computing server. The cloud computing server will search the index with the trapdoor based on each developed scheme. After searching through the whole index, the cloud computing server will return all index entries that matched the trapdoor back to the user application. The user application will decrypt the returned index entries and return the decrypted index entries back to the data users. The data users can use the information from the decrypted index entries to decide which data files should be retrieved from the cloud storage.

Synonym-Based Keyword Search (SBKS)

The first scheme we developed is called “Synonym-Based Keyword Search (SBKS)”. SBKS is an improved version of WFSC proposed in [3]. Instead of expanding the keyword with wildcard character as in WFSC, SBKS expands the keyword with the synonyms of the keyword. SBKS captures the user true intention of the search by including keywords with similar meanings in the index and the search.

The index construction of SBKS begins with the user application extracts distinct keywords from each data files. The user application will use the Synonym Set Construction (SSC) process to expand each extracted keyword into the synonym keyword set. The SSC process will first add the keyword to the keyword set and check the keyword against dictionary to determine if the keyword is misspelled. If the keyword is misspelled, spell check will be performed on the keyword to generate a list of keyword suggestions with correct spellings. Each keyword in the list of keyword suggestions will be added to the keyword set. After the spell check, the keyword set will go through synonym dictionary to retrieve all synonyms of each keyword in the keyword set. The SSC process will return all distinct synonyms retrieved and all distinct keywords in the keyword set to form the synonym keyword set. Figure 3 shows an overview of the Synonym Set Construction process.

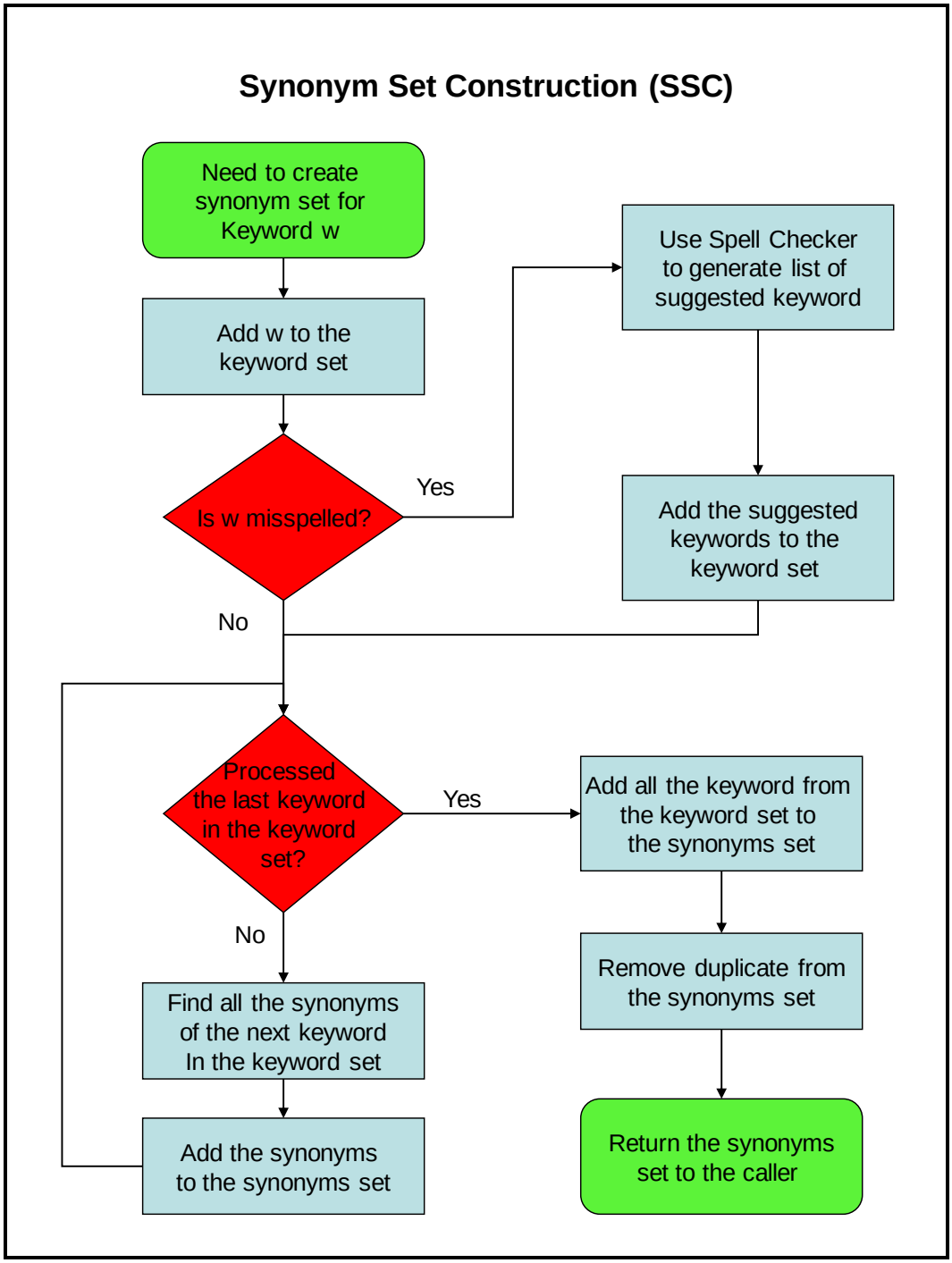

Figure 3: Synonym Set Construction (SSC) process

Table 6 shows examples of the synonym keyword set returned by the SSC process.

Table 6: Examples of Synonym keyword set

Keyword	Synonym Keyword Set
student	bookman, educatee, pupil, scholar, scholarly person, student
dragon	draco, dragon, firedrake, flying dragon, flying lizard, tartar
holllday (misspell)	Holladay, holiday, holllday, vacation

Each keyword in the synonym keyword set will be hashed with a secure hash function to create a trapdoor. An index entry will be created for each trapdoor the following format:

H(Synonym_Keyword)|Enc(List_of_File_ID|Shared_Key|Original_Keyword)

H(Synonym_Keyword) is the trapdoor created with the secure hash function. Enc(List_of_File_ID|Shared_Key|Original_Keyword) contains the original keyword that used to generate the synonym keyword set, the shared key submitted by the data owner, and the list of data files ID that contains the original keyword, and all these information will be encrypted by the symmetric-key encryption with the shared key. For example, the following 4 index entries will be created for keyword “holllday” that extracted from FILE1 and FILE2:

H(Holladay)|Enc(FILE1:FILE2|K|holllday)

H(holiday)|Enc(FILE1:FILE2|K|holllday)

H(holllday)|Enc(FILE1:FILE2|K|holllday)

H(vacation)|Enc(FILE1:FILE2|K|holllday)

The index construction will end when all the keywords extracted from the data files have been processed with index entries created. All the index entries will form an index file that will be updated to the cloud storage.

When the user want to search with a specific keyword, the user application will use the SSC process to generate the synonym keyword set for the search keyword. Each of the keyword in the synonym keyword set will be

hashed by the secure hash function to generate the list of trapdoors. For example, the keyword “student” will generate the following trapdoors:

H(bookman)|H(educate)|H(pupil)|H(scholar)|H(scholarly person)|H(student)

The list of trapdoors will be submitted to the cloud computing server. The cloud computing server will search the index by comparing the trapdoor in each index entry to each trapdoor in the list. The cloud computing server will return all matched index entries back to the user application. The user application will decrypt the information portion of the index entry and return the information to the data users.

Wikipedia-Based Keyword Search (WBKS)

The second scheme we developed is called “Wikipedia-based Keyword Search (WBKS)”. WBKS adopted the Wikipedia Similarity Matching technique that was proposed recently to resolve the content-targeted advertising problem. WBKS uses a set of Wikipedia articles which we called the “Wikipedia Key Set (WKS)” as a reference point between the data files and the search keyword. Unlike the schemes proposed previously, each index entry in WBKS indexes each uploading data file instead of the keyword extracted.

To construct the index for WBKS, the data owner needs to pre-compute the WKS using a set of Wikipedia articles. The arrangement of WKS will act as the shared key to form the trapdoor when building the index and creating the search trapdoor.

To pre-compute WKS, the user application will use the term frequency-inverse document frequency (TF-IDF) to generate the vector representation for each selected Wikipedia article. The following formula of TF-IDF will be used in WBKS:

$$\mathrm{tf}(t,d) = \frac{\mathrm{f}(t,d)}{\max\{\mathrm{f}(w,d) : w \in d\}}$$

$$\mathrm{idf}(t,D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

$$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d) \times \mathrm{idf}(t, D)$$

The collection of vector representations of each Wikipedia articles will form the WKS.

To construct the Wikipedia index for WBKS, the user application first uses TF-IDF to generate the vector representation for each uploading data files. With the vector representations created, the user application will compute the cosine similarity between the vector representation of each file and vector representation of each Wikipedia articles in WKS. The following formula of cosine similarity will be used in WBKS:

$$\mathrm{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

Each index entry for each data file will have the following format:

CS(FileN|a0)|CS(FileN|a1)|..CS(FileN|ai)|Enc(FileN|Shared_Key)

CS(FileN|ai) is the cosine similarity between File N and Wikipedia article i in WKS. The list of the cosine similarity between the File N and each Wikipedia articles in the entry is called similarity trapdoor. Enc(FileN|Shared_Key) contains the file ID and the shared key submitted by the data owner, and this information is encrypted by the symmetric-key encryption. The index construction will end when all the data files have been processed with index entries created. All the index entries will form an index file that will be updated to the cloud storage. Figure 4 shows an overview of the Wikipedia index construction process.

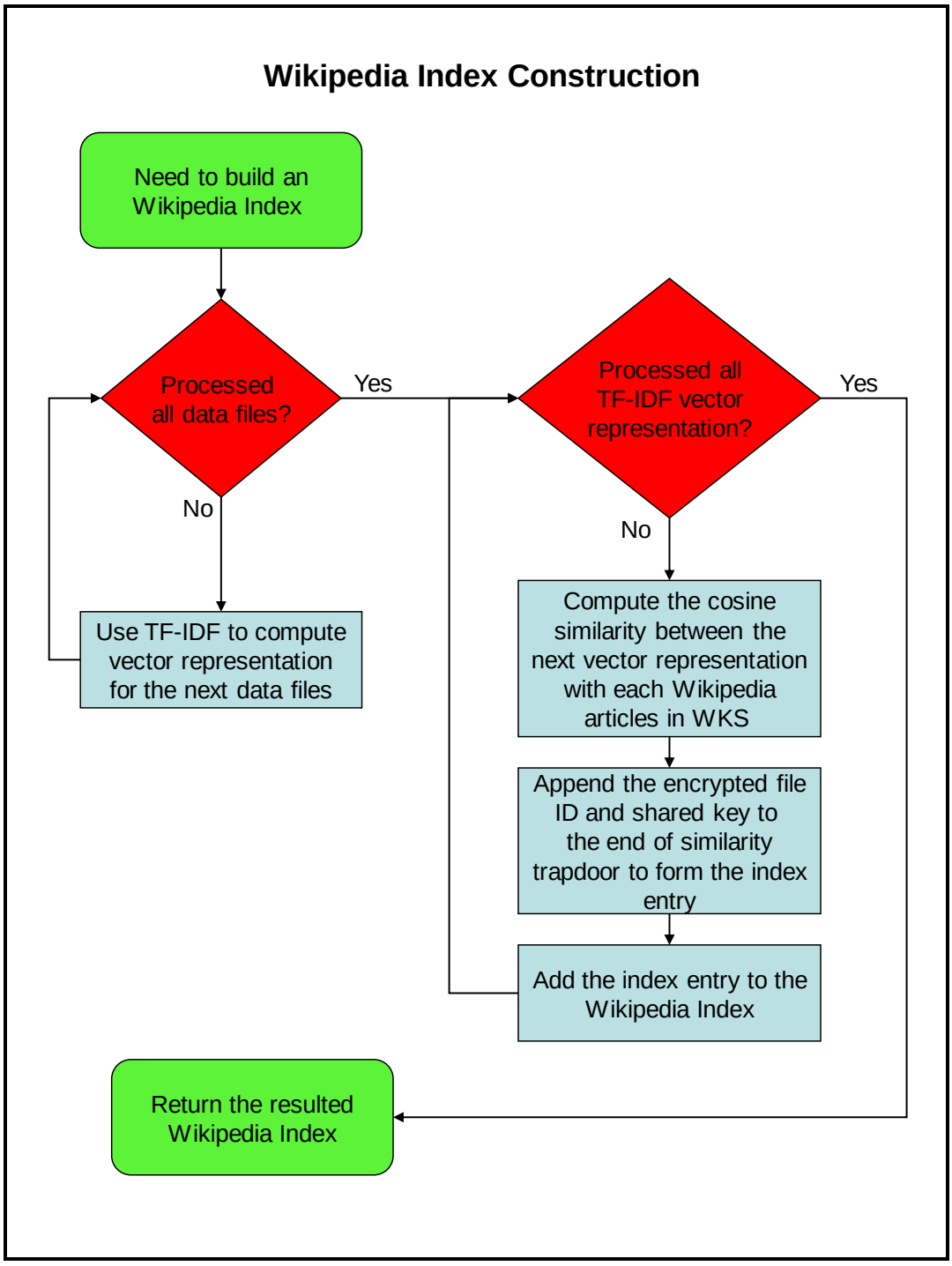

Figure 4: Overview of the Wikipedia index construction

When the user wants to search with a specific keyword, the user application will use TF-IDF to generate the vector representation of the search keyword. The user application will then compute the cosine similarity between

the vector representation of the keyword and the vector representation of each Wikipedia articles in WKS. The resulted list of cosine similarity will form the similarity trapdoor and the user application will submit the similarity trapdoor to the cloud computing server. The cloud computing server will compute the cosine similarity between the similarity trapdoor and the similarity trapdoor in each index entry to generate the similarity score for each entry. The index entries with the higher similarity score will be returned to the user application. The user application will decrypt the information portion of the index entry and return the information back to the data users.

Wikipedia-based Synonym Keyword Search (WBSKS)

The third scheme we developed in this project is called “Wikipedia-based Synonym Keyword Search (WBSKS). It is a hybrid of SBKS and WBKS. The Key of WBSKS is to take advantage of SSC from SBKS and the Wikipedia index from WBKS to expand the search to cover synonyms of the search keyword while maintaining small index.

WBSKS uses the same procedure to construct the Wikipedia index as in WBKS. When the data user wants to do a search with a specific keyword, the user application converts the keyword into the modified keyword set using the same SSC process from SBKS. SSC process corrects the possible typos in the keyword and pull in the synonyms of the keyword from the dictionary. The user application then uses TF-IDF to form the vector representation of the modified keyword set and computes the similarity trapdoor between the modified keyword set and WKS. With the similarity trapdoor, the cloud server computes the similarity score between the similarity trapdoor and each index entries and returns entries with higher similarity score.

V. Test Implementation

In this section, we will describe the implementation of our test system. We will also describe the different tests we have conducted to collect experimental data that we will analyze in the next section.

Our test system is implemented in Java including a fully functional user application and a cloud server simulator with all the required functions as described in Section III. The user application supports our developed schemes (SBKS, WBKS, and WBSKS) and WFSC from [3]. WFSC is implemented and is used as the benchmark in our tests. We have implemented our keyword extractor to filter out common stop words such as “the”, “which”, “is”, and “in” when

extracting keywords from the data file. We have used the Data Encryption Standard (DES) algorithm for all our symmetric-key encryption and decryption, and we have used the MD5 Message-Digest Algorithm as our secured hash function. We used the Basic Suggester Java software, provided by SoftCorporation, LLC, as the spell checker and the WordNet® lexical database, provided by Princeton University, as the synonym dictionary. Java API for WordNet Searching (JAWS) is being used to access the WordNet® lexical database.

For Wikipedia articles, we have extracted all 3807 featured articles from the Wikipedia dump created as of February 25th, 2013. The Wikipedia featured articles are considered as the best articles in the Wikipedia that are selected from different categories with careful review to ensure the content is complete and accurate. All 3807 featured articles will be pre-computed to form the WKS used in WBKS and WBSKS.

We conducted tests using our developed schemes and WFSC with predefine edit distance value equals 1 (WFSC-ED1) and 2 (WFSC-ED2). We measured the size of the indexes and the performance of index construction by uploading our data files collection through our user application. Our data file collection contains 50 short English literatures such as “The Little Match Girl” and “The Elves and the Shoemaker”. We have randomly selected 20 distinct keywords from the data files and performed searches using each selected keywords to measure the search performance. We also carefully examined the index files and the trapdoors created by our developed scheme to measure the data security for our schemes. At last, we measured the search quality by classifying the search results returned from the searches for WFSC-ED1, WFSC-ED2, and SBKS. We only compared the SBKS with WFSC because they return results in similar manner based on the keywords. Results returned by WBKS and WBSKS are based on the data file; therefore, different tests will be conducted to measure the search quality for WBKS and WBSKS. To measure the search quality for WBKS and WBSKS, we added 10 randomly selected Wikipedia articles to our collection as target articles and performed searches using the title of each target articles to determine how likely the target articles will return with similarity score in the top 3.

All our tests are conducted on a Desktop PC with Intel® Core™ i5-750 CPU and 12.0 GB RAM.

VI. Test Data and Analyses

In this section, we will review the data collected from our tests and present our analysis in term of storage requirements, performances, data security, and search quality. Appendix A, Test Data, of this paper presents the actual data values that used to generate all the figures presented in this section.

Storage Requirements

In our tests, we uploaded different number of files in our data file collection using the user application and measured the changes in the size of the index file and the number of index entries in the index when the number of files uploaded equals to 10 (5004 distinct keywords), 20 (9067 distinct keywords), 30 (10732 distinct keywords), 40 (11649 distinct keywords), and 50 (13137 distinct keywords). Figure 5 shows the comparison of the sizes of the indexes that were created by each supported schemes.

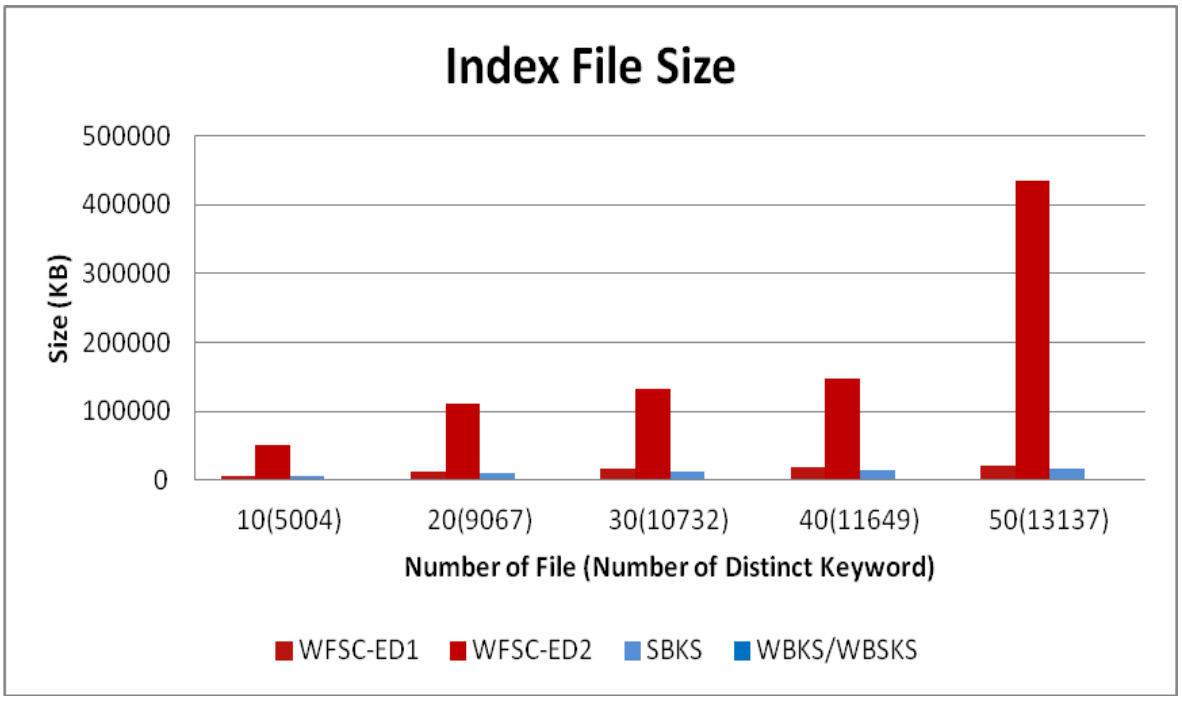

Figure 5: Comparison of the sizes of the index files

According to our data in Figure 5, the index files created by our developed schemes are smaller than both the index files created by WFSC-ED1 and WFSC-ED2. The data show the index file of SBKS has an average of 23% size reduction if compared to WFSC-ED1 and an average of 92% size reduction if compared to the WFSC-ED2. The data show the index files of WBKS and

WBSKS have an average of 94% size reduction if compared to WFSC-ED1 and an average of 99% size reduction if compared to WFSC-ED2. The data also show the index files of WFSC has an average of 974% size increase by simply increasing the edit distance from 1 to 2.

Beside the actual size of the index files, we also measured the number of index entry in each index files created in our tests. Figure 6 shows the comparison of the number of index entry in the indexes that were created by each supported schemes.

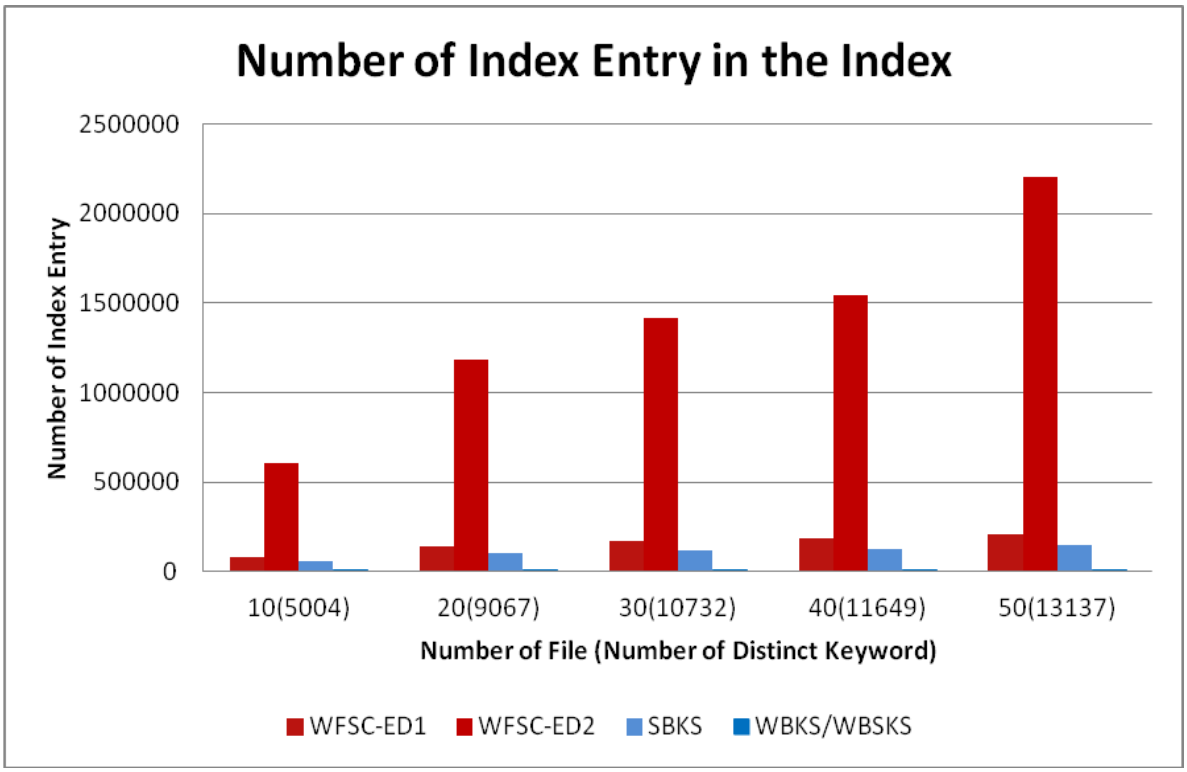

Figure 6: Comparison of the numbers of index entry in the index

According to our data in Figure 6, the index files created by our developed schemes contain lesser index entry than the index files created by both WFSC-ED1 and WFSC-ED2. The data show SBKS has an average of 29% reduction in the number of index entry if compared to WFSC-ED1 and an average of 92% reduction if compared to WFSC-ED2. The data show the numbers of index entry in the index files created by WBKS and WBSKS have an average of more than 99% reduction if compared to both WFSC-ED1 and WFSC-ED2. The data also show the increase in the numbers of index entry in WBKS and WBSKS are

depending only on the number of data files while WFSC and SBKS are depending on the number of distinct keywords in the data files.

These significant size reductions and index entry reductions proved our developed schemes are more efficient than WFSC in terms of storage requirements.

Beside the index file in cloud storage, we also measured the storage requirements for items, such as the dictionaries and WKS that are taking up local storage. These items are only taking up storage in the user application and can be shared among different users. Table 7 shows the sizes of items that take up space in local storage. The total amount of local storage used in our test system is 82MB.

Table 7: Sizes of items that take up space in local storage

Items in Local Storage	Sizes
WordNet® lexical database	36 MB
Basic Suggester Java software	1 MB
WKS with 3807 articles	45 MB

Since [11] suggested that the accuracy of the Wikipedia matching can be improved by using more Wikipedia articles as the reference point, we have measured the changes in WKS when number of articles increases. Figure 7 shows the changes in WKS size when the number of the Wikipedia articles changes, and it shows the sizes of the Wikipedia articles before they are converted to WKS.

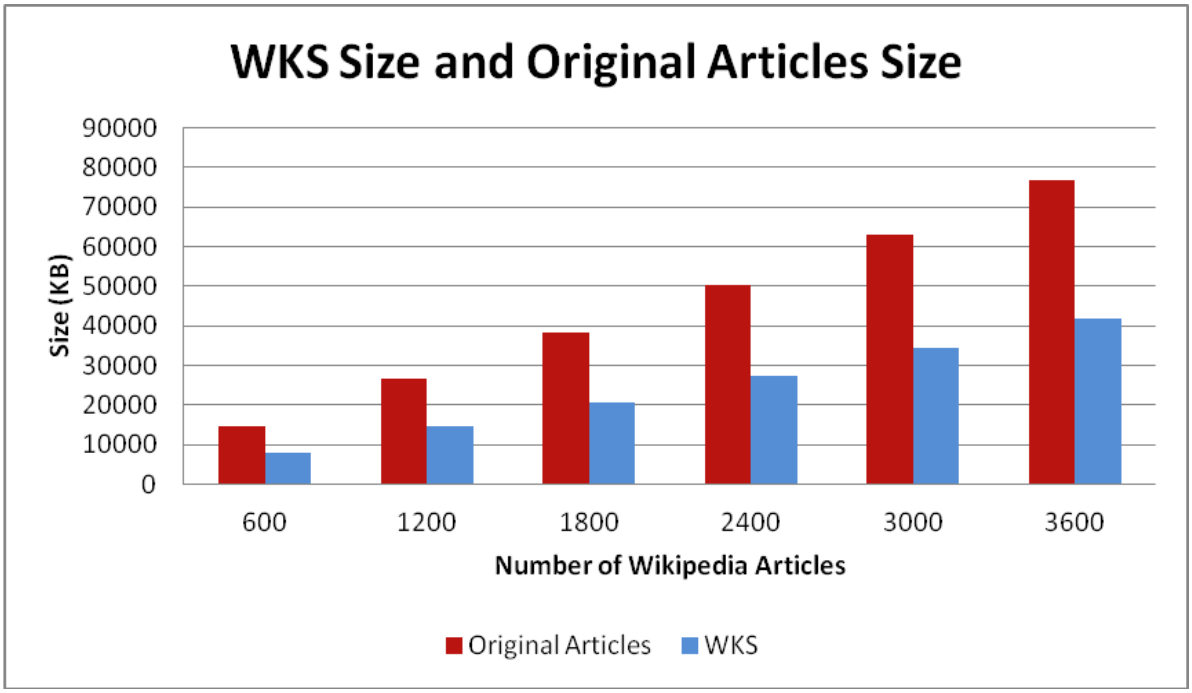

Figure 7: Comparison of Wikipedia articles sizes and WKS sizes

According to the data in Figure 7, the conversion from the Wikipedia articles to WKS has an average of 45.68% size reduction if compared to the original articles. The data also show an average of 6813 KB size increase in WKS for every 600 articles added.

Figure 8 shows the changes in the size of the index file for 50 data files as the number of articles included in the WKS increases.

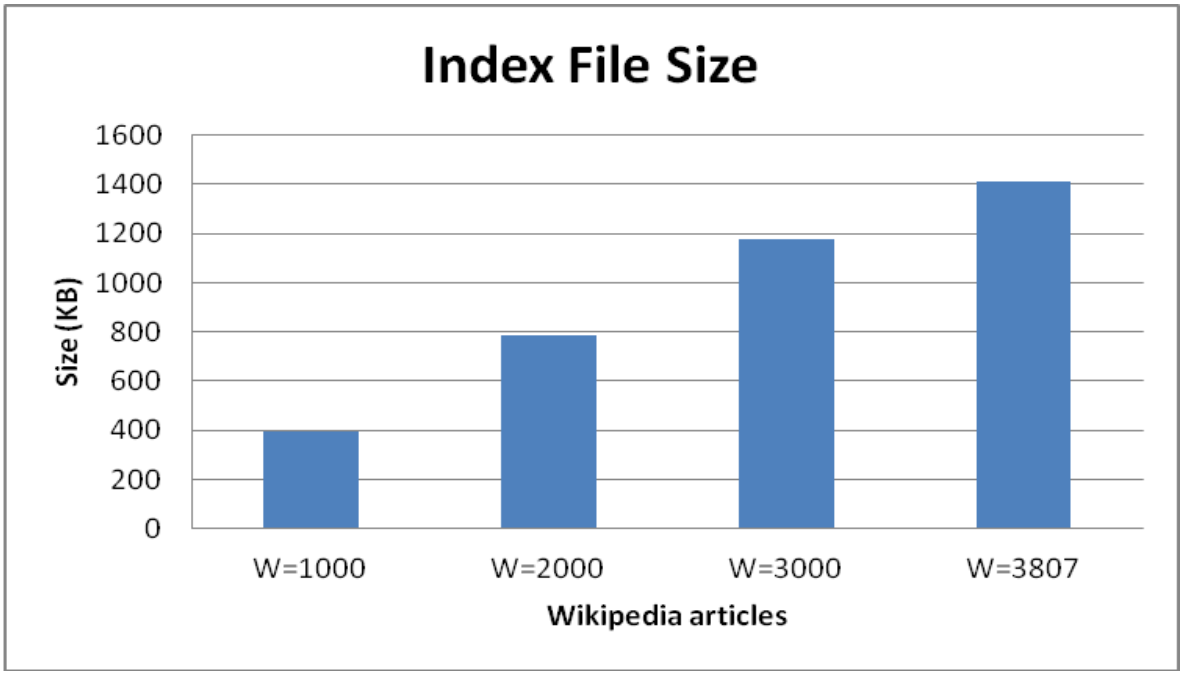

Figure 8: Index file size increases when the number of articles in WKS increases

According to the data in Figure 8, the index file increases as the number of articles in WKS increases. The data show the index size is increasing approximately 390KB for every 1000 articles we added to the WKS.

With the small increases in WKS size and index file size, the data suggested we can increase the number of Wikipedia articles in WKS to further improve the search quality without having a huge increase in the size of the index file.

Performance

While measuring the size of the index files, we have also captured the time used for index construction with each schemes when the number of files uploaded equals to 10 (5004 distinct keywords), 20 (9067 distinct keywords), 30 (10732 distinct keywords), 40 (11649 distinct keywords), and 50 (13137 distinct keywords). Figure 9 shows the changes in average time used for index construction when the number of file uploaded increases.

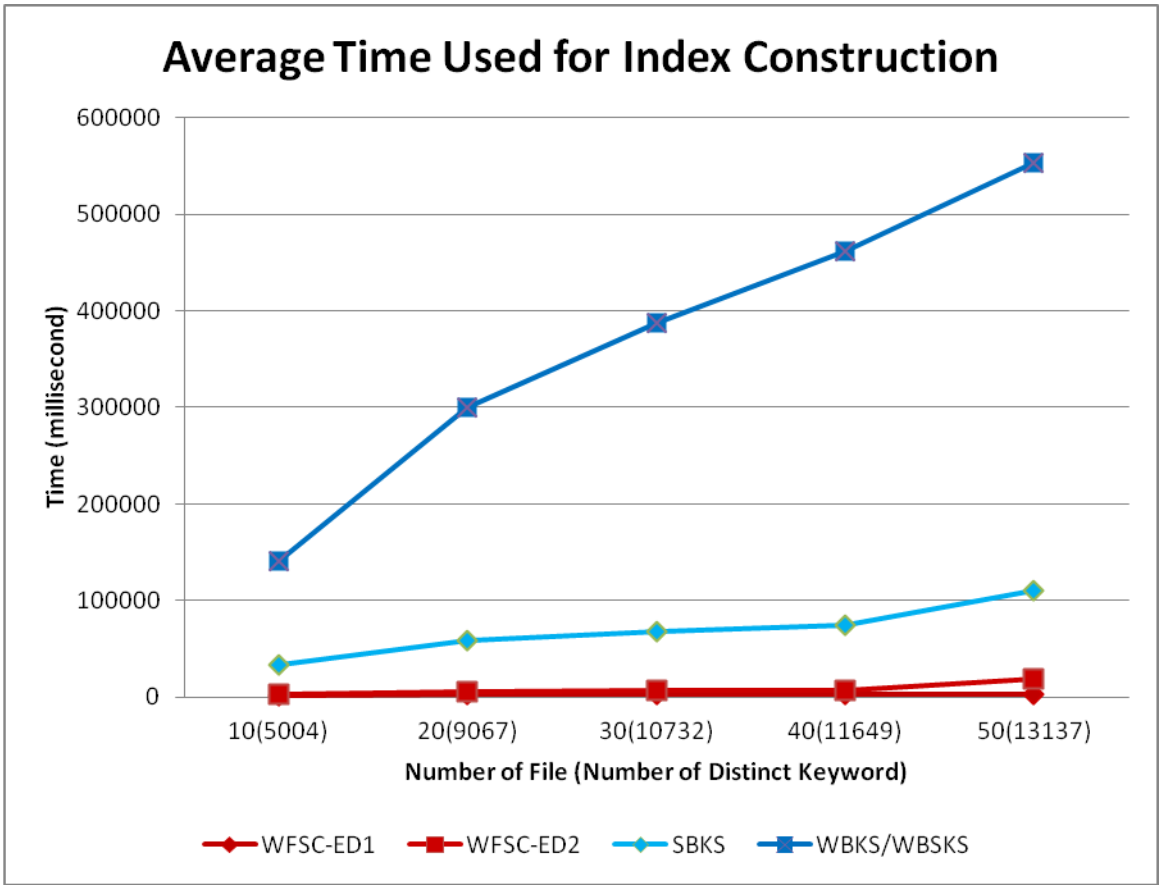

Figure 9: Comparison of average time used to construct the index files

According to the data shown in Figure 9, the performance of index construction for our developed scheme is lower than both WFSC-ED1 and WFSC-ED2. The data demonstrate the time used for index construction with SBKS has an average of 2367% increase if compared to WFSC-ED1 and an average of 856% increase if compared to WFSC-ED2. The data show the time used for index constructions with WBKS and WBSKS have an average of 12861% increase if compared to WFSC-ED1 and an average of 4882% increase if compared to WFSC-ED2. The degradation in performance of index construction is expected because our developed schemes request to query multiple data against various dictionaries and WKS that reside in the local storage during index construction while WFSC requires only simple string modification.

Besides measuring the performance of index construction, we also measured the time used for the cloud computing server to search against the uploaded index files with a keyword. We have performed searches with 20

randomly selected keywords against indexes that created when the number of files uploaded equals 10 (5004 distinct keywords), 20 (9067 distinct keywords), 30 (10732 distinct keywords), 40 (11649 distinct keywords), and 50 (13137 distinct keywords). Figure 10 shows the changes in average times used to search the index when the number of files uploaded increase.

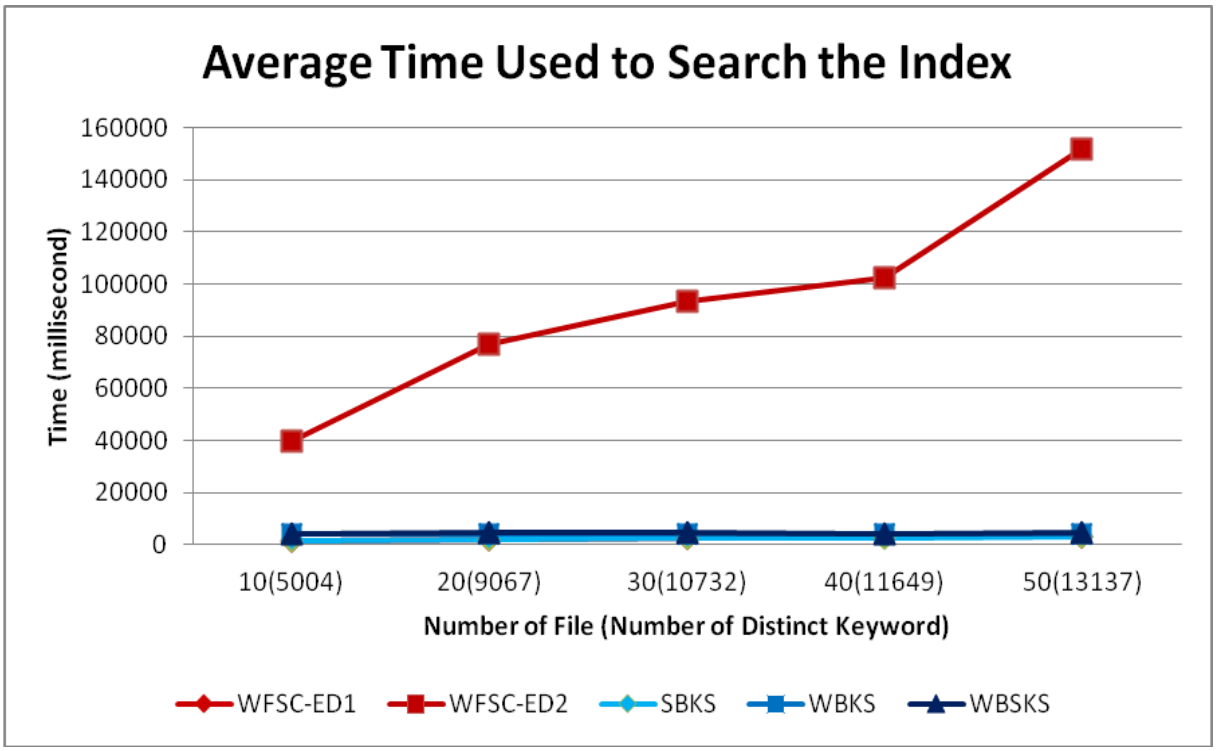

Figure 10: Comparison of average times used to search the index

According to the data in Figure 10, our developed schemes demonstrated higher search performance than WFSC-ED1 and WFSC-ED2. The data show SBKS has an average of 3% reduction in average time used to search the index if compared to WFSC-ED1 and 97% reduction if compared to WFSC-ED2. The data show WBKS and WBSKS have an average of 95% increase in average time used to search the index if compared to WFSC-ED1, but an average of 94% reduction if compared to WFSC-ED2. The data also indicate the times used for WBKS and WBSKS are staying in a constant range of around 4 seconds even with the number of data files increased while the times used in SBKS and WFSC are increasing linearly.

Although the index construction performance of our developed schemes is lower than WFSC, our developed schemes show an average of more than 95% reduction in time used to search the index if compared to WFSC-ED2. Since the

number of search execution is far greater than index construction, our developed schemes are more efficient than WFSC in terms of overall performance.

Beside the index construction and search, we also measured the time used to pre-compute the WKS in our tests. Figure 11 shows the time used to pre-compute the WKS with different numbers of Wikipedia articles.

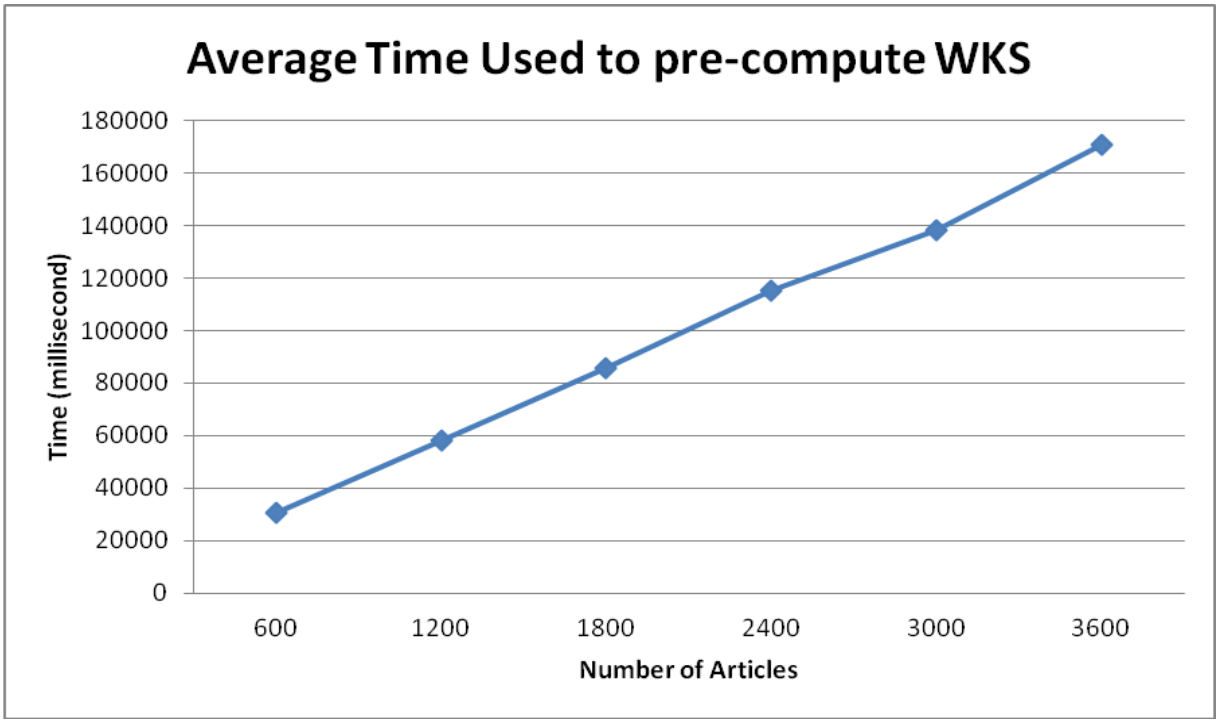

Figure 11: Average time used to pre-compute WKS increases as the number of Wikipedia articles increases

According to the data in Figure 11, the average time used to pre-compute WKS increases as the number of Wikipedia articles increases. The data show an average of 28 seconds increase for every 600 articles added to WKS.

Data Security

In our analysis, we have examined the index entries and trapdoor generated in each of our developed schemes to determine the data security of our developed schemes. Table 8 shows examples of index entry that are generated by our test system in each of our developed schemes. In our examples, the <TP> tag indicates the beginning of the trapdoor for SBKS and similarity trapdoor for WBKS and WBSKS, and the <FE> tag indicates the beginning of the encrypted file information.

Table 8: Index entries that are generated by our developed schemes

Schemes	Index Entry Generated
SBKS	<TP>jvavB£»ãVž§?Ûì!<FE>9D00ypbDl1e8vNgFcTVg88rcpG8/2EGD2CVN9/mkJQWPzODmT3nV/EonFzt6ffjE
WBKS/WBSKS	<TP>:0.11065:0.12282:0.07326:0.39366:0.06878:0.10882:0.09187:0.06852:0.09960:0.03527:0.04153:0.15833:0.04637… (the rest of the cosine similarity vector) <FE>9D00ypbDl1e8vNgFcTVg88rcpG8/2EGDRg9jWibg/lw=

The index entry example of SBKS shows the keyword indexed by this entry is concealed into a trapdoor with the secure hash function. The data file information in SBKS entry is concealed by the symmetric-key encryption. For the WBKS and WBSKS index entry example, the similarity trapdoor that representing a data file is a vector of cosine similarity values and no information will be revealed without knowing the Wikipedia articles arrangement in the WKS. The data file information in WBKS and WBSKS is also concealed by the symmetric-key encryption. These examples show the index entries created by our developed schemes do not reveal any information of the data files uploaded.

Table 9 shows examples of trapdoor used for search that are generated by our test system in each of our developed schemes.

Table 9: Trapdoors that are generated by our developed schemes for search

Schemes	Trapdoors Created for Search
SBKS	<TP>2_ ® ñ©®`…õJl¹+—<TP>Só]—Û † K ¼¹ †<TP>´ ¿?œ òõý : Y<TP> íûÚï„<u?;[5]9‹`Ô
WBKS	0.00000:0.00000:0.00000:0.00000:0.00000:0.00000:0.00000:0.00000:0.00304:0.00000:0.00000:0.00000:0.00000:0.00000: … (the rest of the cosine similarity vector)
WBSKS	0.00000:0.00000:0.00000:0.00000:0.00000:0.01054:0.00251:0.00000:0.00124:0.00000:0.00000:0.00383:0.00000:0.00000: … (the rest of the cosine similarity vector)

Our example for SBKS shows the list of trapdoors generated for the modified keyword set from SCC that will be used by the cloud computing server for search. All keywords in the modified keyword set are concealed by the secure hash function similar to the trapdoor in the index entry. The WBKS and WBSKS examples also show the similarity trapdoor is a vector of cosine similarity values and no information will be revealed without knowing the Wikipedia articles arrangement in the WKS. These examples show the trapdoor and similarity trapdoor generated for search in our developed schemes do not reveal any information of the data files uploaded if WKS remains hidden.

Using our examples, we shows the index entries and the trapdoors created by our developed schemes are secured unless the shared key or WKS arrangement is compromised. These examples show our developed schemes are preserving the data security and privacy of the uploaded data file if the shared key and the WKS arrangement remain hidden.

Search Quality

To measure the search quality, we analyzed the resulted data from searches that were conducted under each supported schemes. We have first compared the result data collected from SBKS and WFSC searches because both of them return their search result in the keyword level. The search quality of WBKS and WBSKS will be analyzed separately from SBKS and WFSC because the results from WBKS and WBSKS search are in the form of similarity score on the data file level.

WFSC and SBKS

We have randomly selected 20 keywords from our data files collection and conducted 20 searches against each schemes using each keywords to collect search result data. The average length of the randomly selected keywords is 5. We have measured the search quality of SBKS and WFSC based on the number of keyword returned from each search and the quality of the returned keywords. Figure 12 shows the average number of distinct keywords returned from a search.

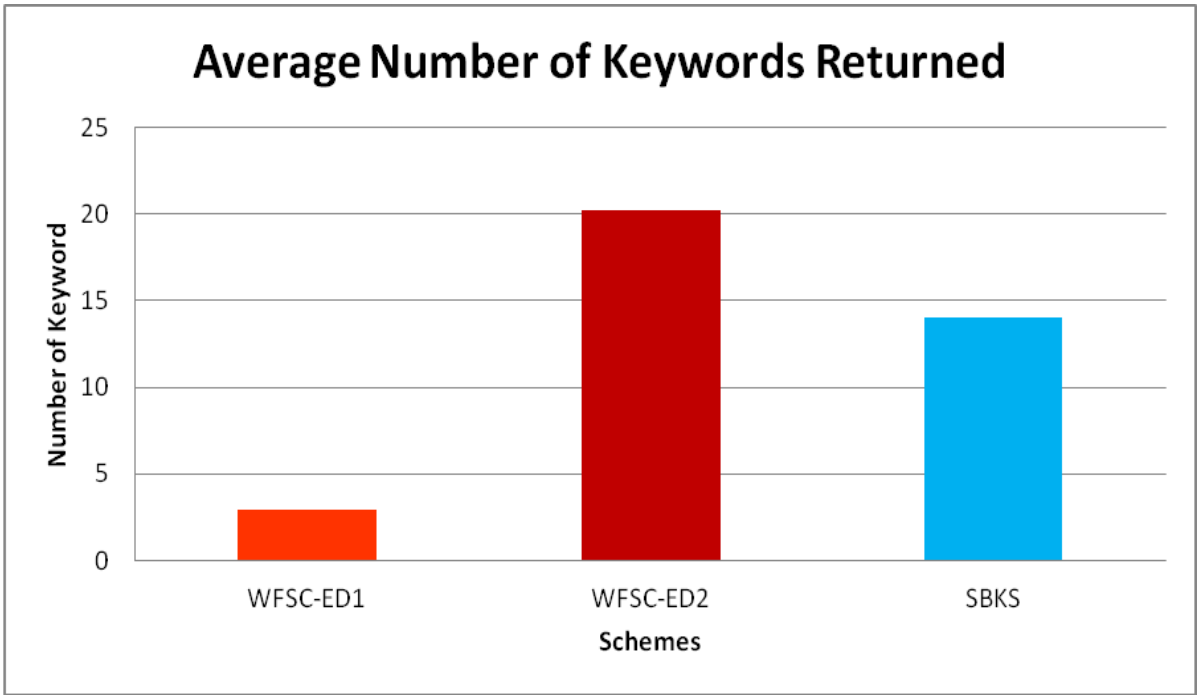

Figure 12: Comparison of average number of distinct keywords returned from a search

According to the data in Figure 12, WFSC-ED1 returns an average of 2.9 keywords from each search while WFSC-ED2 returns an average of 20.2 keywords. These data indicate the search coverage of WFSC-ED1 is too narrow while the search coverage of WFSC-ED2 is too board. The data also show SBKS returns an average of 14 keywords from each search.

To analyze the quality of the search results, we have classified keyword returned from the searches by their relationships to the search keyword. Table 10 describes the search result classifications that we used.

Table 10: Search result classifications

Classification Types	Descriptions
Type 1	The returned keyword is an exact match or shared same word stem with the search keyword
Type 2	The returned keyword belongs to the keyword synonym set of search keyword
Type 3	The returned keyword shares the same synonym

	with the search keyword
Type 4	The returned keyword is unrelated to the search keyword

For example, a search with search keyword “happy” was returned with keywords “happy”, “blessed”, “hallowed”, and “unknown” as the results. Keyword “happy” will be classified as Type 1 because it is an exact match with the search keyword. Keyword “blessed” will be classified as Type 2 because it is contained by the synonym set of keyword “happy”. Keyword “hallowed” will be classified as Type 3 because it is not in the synonym set of “happy”, but it is sharing a common synonym “blessed” in their synonym set. Keyword “unknown” will be classified as Type 4 because it is not related to keyword “happy” in any ways.

All returned keywords should fall into Type 1, 2, or 3, and any keywords fall into Type 4 degrade the search quality. Figure 13 shows the comparison in the search result classifications.

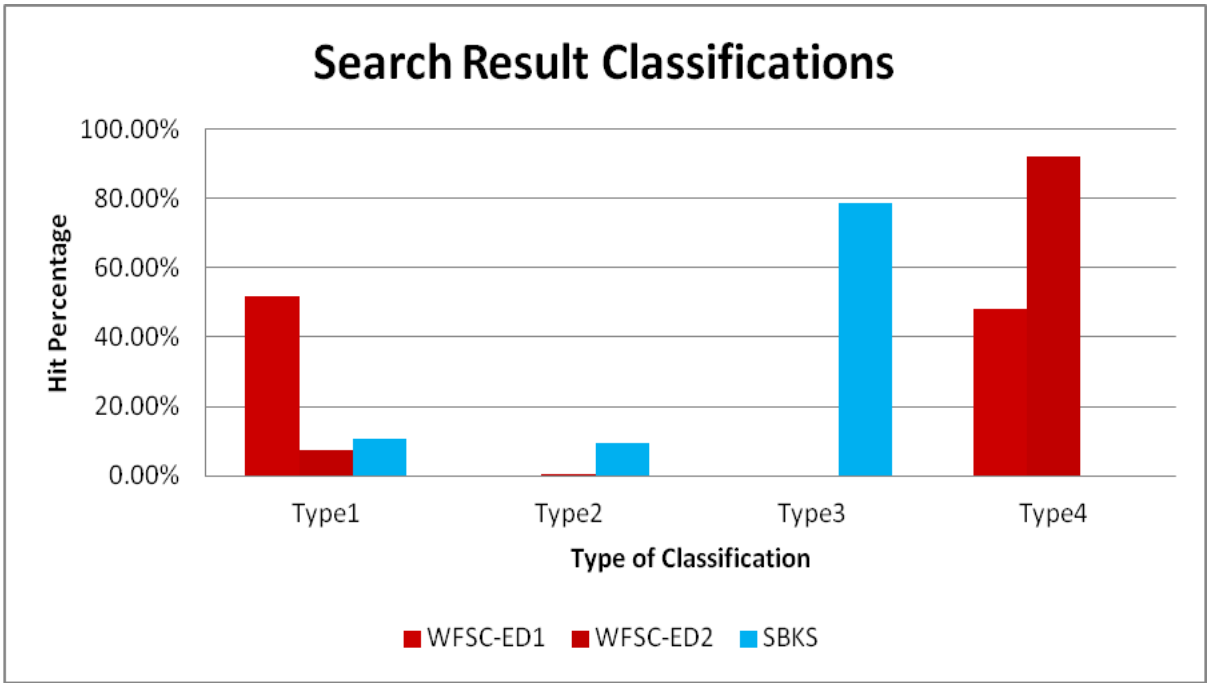

Figure 13: Comparison of search result classifications

According to the data in Figure 13, the quality of the search results returned by SBKS is higher than both WFSC-ED1 and WFSC-ED2. The data

show WFSC-ED1 has 51.72% of the keywords fall into Type 1 and 48.28% of the keywords fall into Type 4. The data show WFSC-ED2 has 7.43% of the keywords fall into Type 1, 0.50% of the keywords fall into Type 2, and 92.08% of keywords fall into Type 4. The data also show SBKS has 10.71% of the keywords fall into Type 1, 9.29% of the keywords fall into Type 2, and 78.57% of the keywords fall into Type 3. These data indicates 100% of the keywords returned by SBKS are somehow related to the search keyword while WFSC-ED2 has 92.08% of keywords returned and WFSC-ED1 has 48.28% of keywords are considered as unrelated. The high percentages of keywords returned by WFSC-ED1 in Type 1 are due to the way we selected our search keywords. In our tests, we selected our keywords directly from the data files; therefore, there is at least 1 keyword will return in the result.

We have demonstrated that search results returned from SBKS is better than both WFSC-ED1 and WFSC-ED2 and we have shown the search coverage for WFSC is either too narrow or too board with different edit distance values; therefore, the search quality of SBKS is better than WFSC.

WBKS and WBSKS

To measure the search quality of WBKS and WBSKS, we have randomly added 10 different Wikipedia articles with various topics into our data files as target articles. By extracting a keyword from the title of each target articles, we form the regular keyword set with 10 keywords. We also formed the typo keyword set by randomly replacing 1 character of each keyword in the regular keyword set and formed the related keyword set by replacing each keyword in the regular keyword set with a related keyword. With the 3 different keywords set, we conducted search using each keyword in the 3 keywords set to determine how likely the corresponding target article will return with similarity score in the top 3. Figure 14 shows the percentages of target articles returned with similarity score in the top 3.

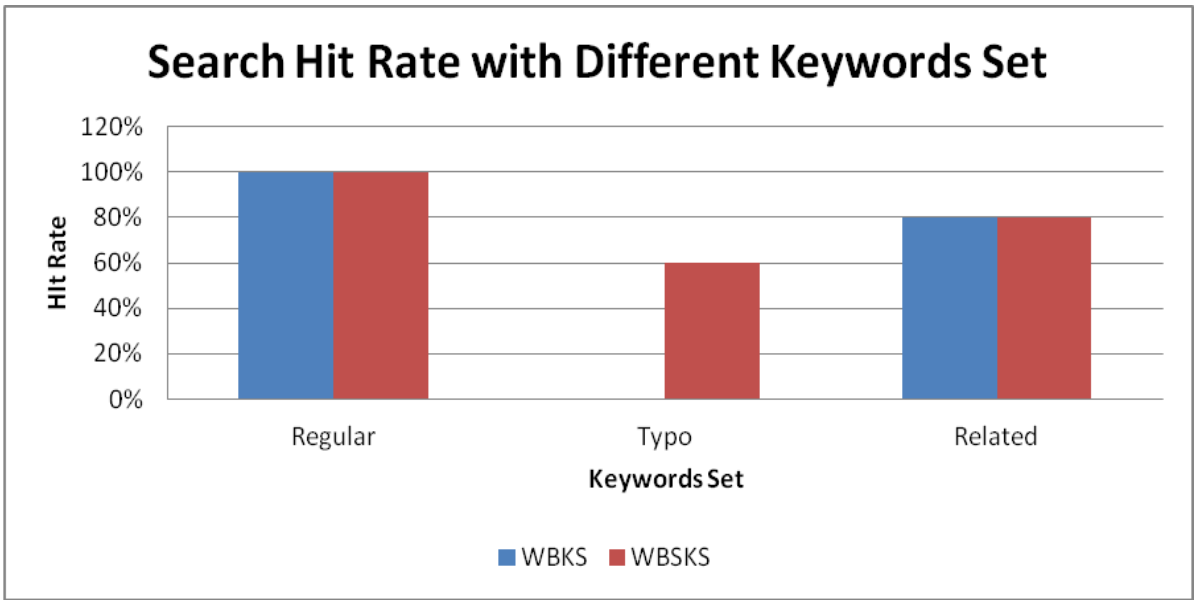

Figure 14: Comparison of search hit rate with different keywords set

According to the data in Figure 14, both WBKS and WBSKS have 100% hit ratio to return the target articles with similarity score in the top 3 when the searches were done using the regular keyword set. The data show the WBKS has 0% hit rate while WBSKS has 60% hit rate when searches were done using the typo keyword set. This indicates WBKS will not work properly when there is typo in the search keyword. The search failures from the typo keyword set in WBSKS were due to the spell checker didn't correct the typo correctly. Typo keywords that failed are proper noun such as "Samsung" and "Batman" that are not contained in our dictionaries. The data also shows both WBKS and WBSKS have 80% hit rate when searches are done using the related keyword set. The search failures with related keyword set are due to the related keyword is used commonly by other data files. We do see the target articles returned in the failed searches and ranked as the 4^{th} and 5^{th} most relevant in the result.

Our analysis shows both WBKS and WBSKS are very accurate when the search keyword used is error-free and fairly accurate when handling related keywords. The analysis also shows WBSKS can handle typo in search keyword based on the quality of the dictionaries implemented in our scheme.

VII. Conclusion and Recommendations

In this project, we have developed a system to support semantic search over encrypted data in cloud computing with three different schemes. We

reviewed several schemes that were proposed in previous literatures to support keyword search over encrypted data in cloud computing and identified the problems existing in each of the schemes. After we reviewed the related literature, we gave a high level overview of our developed system and described in detail the functions of each major component in our system. We further explained our three schemes to enable semantic keyword search over encrypted data in cloud computing. The three schemes that we have developed in this project are: “Synonym-Based Keyword Search (SBKS)”, “Wikipedia-Based Keyword Search (WBKS)”, and “Wikipedia-Based Synonym Keyword Search (WBSKS)”. SBKS used the Synonym Set Construction (SSC) process to expand the keyword with synonyms of the keyword to ensure words with similar meanings will be covered by the searching order to capture user’s true intention of the search. WBKS adopted the Wikipedia Similarity Matching technique to create a reference point with a set of Wikipedia articles called Wikipedia Key Set (WKS) between the uploaded data files and the search keyword. WBSKS took advantages of WBKS and SBKS to further increase the keyword intersections between the uploaded data files and the search keyword.

We have implemented our developed schemes and conducted various kinds of test to collect experimental data for comparison and analysis. We have also implemented the “Wildcard-based Fuzzy Set Construction (WFSC)” proposed in [3] with predefine edit distance 1 (WFSC-ED1) and 2 (WFSC-ED2) as our benchmark. With careful analyses over the collected test data, we have illustrated our developed schemes perform better than WFSC in terms of storage requirements, performance, and search quality while preserving the data security and privacy of the uploaded data. In terms of storage requirements, our developed schemes show an average of 95% reduction in index size if compared to WFSC-ED2. In terms of performance, our developed schemes have illustrated an average of 95% reduction in time used to search the index if compared to WFSC-ED2. In terms of security, we have exhibited our developed schemes is preserving the data security and privacy of the uploaded data if the shared key and the WKS arrangement remain hidden through careful analysis over the index entries and trapdoors created by our developed schemes. In terms of search quality, we compared SBKS with WFSC and the comparison showed 100% of the results returned by SBKS were related to the search keyword while only 51.72% returned by WFSC-ED1 and only 7.93% returned by WFSC-ED2 related to the search keyword. We have demonstrated the search quality of WBKS and WBSKS were high with 100% hit rate to return the target data file when searching with an error-free keyword. We also illustrated how WBSKS can

handle typo keyword to certain extend depending on the dictionaries implemented with the scheme. With all these improvements showed in our analyses, we can conclude that our developed schemes are more efficient and more practical than WFSC.

During our test, we have noticed some limitations in our developed schemes that can be further improved in the future. One problem we experienced in WBKS and WBSKS is an indexing problem when the keywords between the data file and the WKS were not intersecting. Any keywords contained in the data files but not contained in any of the Wikipedia articles in WKS will not be indexed by the index file so any attempt to search using these keywords will fail. This problem is reducing the usability of WBKS and WBSKS. One suggestion to resolve this problem would be implementing a data-oriented Wikipedia articles selection scheme in order to select different Wikipedia articles for WKS based on the content of the uploading file. The challenge of this data-oriented Wikipedia articles selection scheme is to select a set of articles from all the Wikipedia articles within a short period of time. The articles selected by this scheme should include all of the keywords in the data file and share a common topic with the data file. Another good improvement for future work would be to take advantage of SSC process to expand keywords in the uploading data file so the synonyms of the keyword will be considered when constructing the Wikipedia index. In this project, we have actually attempted to use our implemented SSC to expand each keyword into the modified keyword set and use it to construct the Wikipedia index in order to increase the keyword intersections between the data file and WKS. However, our attempts show degradation in accuracy due to the modified keyword has destroyed the term frequency of the keywords. We may need some strategies to expand the keyword with synonyms in a way that will not alter the term frequency of the keyword too much. These two improvements will be some good candidates for future works.

References

[1] Khorshed, M.T., Ali, A.S., Wasimi, S.A. “Monitoring Insiders Activities in Cloud Computing Using Rule Based Learning”, Security and Privacy in Computing Communications (TrustCom), 2011 IEEE 10th International Conference on, 2011, pp 757-764.

[2] Koletka, R., Hutchison, A. “An Architecture for Secure Searchable Cloud Storage,” Information Security South Africa (ISSA), 2011, pp 1-7.

[3] Li, J., Wang, Q., Wang, C., Cao, N., Ren, K., Lou, W. “Fuzzy Keyword Search over Encrypted Data in Cloud Computing,” in INFOCOM, 2010 Proceedings IEEE, pp 1-5.

[4] Liu, C., Zhu, L., Li, L., Tan, Y. “Fuzzy keyword search on encrypted cloud storage data with small index,” Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on, 2011, pp 269-273.

[5] Pak, A. “Using Wikipedia to Improve Precision of Contextual Advertising,” Human Language Technology, Challenges for Computer Science and Liguistics, Lecture Notes in Computer Science, 2011, Vol 6562/2011, pp 533-543.

[6] Ribeiro-Neto, B., Cristo, M., Golgher, P.B., Moura, E.S. “Impedance Coupling in Content-Targeted Advertising.” Proc. SIGIR 05, Salvador, Brazil, August 15–19, pp 496–503., ACM Press, NewYork.

[7] Rocha, F., Correia, M. “Lucy in the Sky without Diamonds: Stealing Confidential Data in the Cloud,” Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on, 2011, pp 129-134.

[8] Salem, M., Stolfo, S. “Modeling user search-behavior for masquerade detection,” Recent Advances in Intrusion Detection, in Proceedings of the 14th International Symposium on, Heidelberg: Springer, 2011, pp 1–20.

[9] Stolfo, S., Salem, M., Keromytis, A. “Fog Computing: Mitigating Insider Data Theft Attacks in the Cloud”, Security and Privacy Workshops (SPW-2012), 2012 IEEE Symposium on, 2012, pp 125-128.

[10] Voris, J., Boggs, N., Stolfo, S. “Lost in Translation: Improving Decoy Documents via Automated Translation”, Security and Privacy Workshops (SPW-2012), 2012 IEEE Symposium on, pp 129-133.

[11] Wu, Z., Xu, G., Zhang, Y., Dolog, P. “An Improved Contextual Advertisting Matching Approach based on Wikipedia Knowledge,” Special Focus on Engineering Knowledge and Semantic Systems, The Computer Journal 2012, pp 277-292.

Appendix A: Test Data

Table 11: Test data used to generate Figure 5

Actual Index Size (KB)					
Files (Keywords)	10(5004)	20(9067)	30(10732)	40(11649)	50(13137)
WFSC-ED=1	6538	12878	15686	17575	21258
WFSC-ED=2	51117	110429	133078	148121	433944
SBKS	5302	9905	11923	13383	16084
WBKS/WBSKS	298	596	894	1192	1490

Table 12: Test data used to generate Figure 6

Number of Index Entries					
Files (Keywords)	10(5004)	20(9067)	30(10732)	40(11649)	50(13137)
WFSC-ED=1	76688	142260	169354	184536	210524
WFSC-ED=2	603320	1185989	1412736	1541958	2206898
SBKS	59414	101409	117253	125617	148975
WBKS/WBSKS	10	20	30	40	50

Table 13: Test data used to generate Figure 9

Index Build Time (millisecond)					
Files (Keywords)	10(5004)	20(9067)	30(10732)	40(11649)	50(13137)
WFSC-ED=1	1761	2616	2929	3058	3240
WFSC-ED=2	2848	5969	6760	7312	18966
SBKS	33997	59259	67716	74172	110227
WBKS/WBSKS	140684	299362	387588	461296	552645

Table 14: Test data used to generate Figure 10

Average Search Time (millisecond)					
Files (Keywords)	10(5004)	20(9067)	30(10732)	40(11649)	50(13137)
WFSC-ED=1	1200	2112	2513	2826	3208.00
WFSC-ED=2	39453	76697	93556	102553	151683.00
SBKS	1215	2119	2485	2524	2913.00
WBKS	3980	3984	4045	4026	4187.00
WBSKS	4181	4653	4549	4155	4452.00

Table 15: Test data used to generate Figure 11

Average Time Used to Pre-Compute WKS (Millisecond)						
Number of Wiki Articles	600	1200	1800	2400	3000	3600
Average Time Used	30383	58155	85466	115023	138459	170624

Table 16: Test data used to generate Figure 7

Size of the WKS (KB)						
Number of Wiki Articles	600	1200	1800	2400	3000	3600
Original Articles	14547	26844	38269	50418	63052	76847
WKS	7850	14487	20800	27471	34428	41918
Reduction Rate	46.04%	46.03%	45.65%	45.51%	45.40%	45.45%

Table 17: Test data used to generate Figure 8

Size of the Index (KB) with different Wikipedia Key Set				
Number of Wikipedia Articles	1000	2000	3000	3807
WBKS/WBSKS	394	784	1175	1409

Table 18: Test data used to generate Figure 13

Classification of the Search Result			
Techniques	WBFKS-ED=1	WBFKS-ED=2	SBKS
Type1	51.72%	7.43%	10.71%
Type2	0.00%	0.50%	9.29%
Type3	0.00%	0.00%	78.57%
Type4	48.28%	92.08%	0.00%

Table 19: Test data used to generate Figure 12

Average number of distinct search result returned	
Schemes	Average number of distinct result
WFSC-ED=1	3
WFSC-ED=2	20
SBKS	14

Table 20: Test data used to generate Figure 14

WBKS and WBSKS target article hit rate			
Schemes	Regular	Typo	Related
WBKS	100%	0%	80%
WBSKS	100%	60%	80%

The End