Fuzzy Keyword Search on encrypted data

A Project Report

submitted by

SWETHA PRIYA EDDULA

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

May 2017

THESIS CERTIFICATE

This is to certify that the thesis entitled **Fuzzy Keyword Search on encrypted data**, submitted by **Swetha Priya Eddula**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology**, is a bonafide record of the work carried out by her under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Chester Rebeiro
Project Guide
Professor
Dept. of Computer Science and Engineering
IIT-Madras, 600 036

Place: Chennai

Date:

ACKNOWLEDGEMENTS

I would like to to express my sincere thanks to my guide **Prof. Chester Rebeiro** for his immediate acceptance of the project idea I proposed and guiding me whenever stuck. He let me work on the my ideas and gave me the freedom of what and how to work on, at my own pace.

I would like to thank Shaih https://github.com/shaih that has implemented homomorphic functions and made it available on github for the use. It is one of very few homomorphic libraries that are available online. This work would have been a lot more tedious if not for Shaih's done work.

ABSTRACT

KEYWORDS: Cloud Computing; Fuzzy Keyword Search; Homomorphic functions; Edit distance; Cloud Security

With the advent of cloud computing, huge amount of data is being outsourced and with each day. Using the cloud has numerous advantages like Scalability, Cost reduction, reliability, manageability to name a few. Because of the advantages that it offers more and more companies small and large are moving to the cloud by every passing day. However security, limited control, downtime stand as challenges to cloud computing.

With more and more sensitive data being outsourced to the cloud,Security becomes a pressing issue that needs to be addressed. Cyber crime is also on the rise making data security the top most concern.

A number of encryption techniques are currently in use. The client using the server encrypts the data before outsourcing. Data retrieval now becomes a matter of concern. The traditional techniques of keyword search on encrypted data can get the files of the exact keyword match. However, practically the search word and keyword are not exactly the same. Differences in format, spelling mismatch due to errors or locational language differences are very common. Fuzzy Keyword search cant be achieved by the traditional methods.

The naive and easy approach is to correct the search word to various possible search terms and encrypt all of them and then send to the cloud. This method requires the client to generate a set of search terms from the search word. This set is usually too large to compute. Numerous papers have suggested the use of wildcard to drastically reduce the large number of search terms. Another approach is the use of homomorphic functions.

Using the homomorphic functions approach doesn't require the generation of corrected search term set generation by the client. It's more elegant way to search for the fuzzy keyword search and the client doesn't have to the pre-computation and lot of memory is also saved.

In this paper we suggest a homomorphic encryption method that enables us to do partial fuzzy keyword search, i.e, submission with edit distance 1. This is achieved using the HELib,an implementation of homomorphic functions by https://github.com/shaih. Encryption method and search algorithm have been suggested to make the fuzzy search possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

When the user of client's software searches for files, the search word is not the exact match of the keywords provided by the client to the cloud service. For instance, **P.O.Box** can be searched for as *P.O Box , PO Box*. **boquet** can be mispelt as *boque*. Such mismatches are very common. This is why fuzzy keyword search is needed.

1.1 Outline of the Work

An elegant solution to the problem by using homomorphic functions has been proposed. In this approach, generation of the fuzzy search terms is done using homomorphic functions. Hence doesn't require the client service to do the generation before querying the cloud. The idea is to develop homomorphic functions for substitution, deletion, insertion, swap, so that they can be generated even from the encrypted keyword using homomorphic functions. A matching algorithm is proposed to match the generated fuzzy words to the indexed client files.

In this work homomorphic functions for substitution with edit distance of 1 are implemented. This is done with the help of the HELib thats avalable on the github.

1.2 Major Contribution

- Built a homomorphic function that takes the encrypted word,position of the letter from the beginning, letter to be substituted(wildcard) as input and outputs the encrypted word with substitution of wildcard at that location.

- Suggested a matching technique to match the encrypted key words to the indices of the files

1.3 Organization of thesis

Chapter 2 defines the problem statement and briefly explains the client-cloud-user set-up that is assumed in this work. While Chapter 3 discusses the related work done in the field.The proposed homomorphic functions, matching algorithm are discussed in detail in Chapter 4. The future scope and the summary of the entire work can be seen in Chapter 5.

CHAPTER 2

PROBLEM DEFINITION

The basic architectural assumptions that are made will be described before going furthur. Client is the company that is using the cloud to provide its services to the company's users. Before giving the data to the cloud, the client indexes each file with a set of keywords. Every file now has a set of keywords attached to it. After this is done, all the files including the set of keywords re encrypted and sent to the cloud. Whenever a user wants to retrieve a file, the search word/set of search words is encrypted and sent to the cloud. The cloud with the help of the encrypted set of search terms, tries to retrieve the files by matching the search words with the encrypted keywords of the files.The matched files are given to the user.

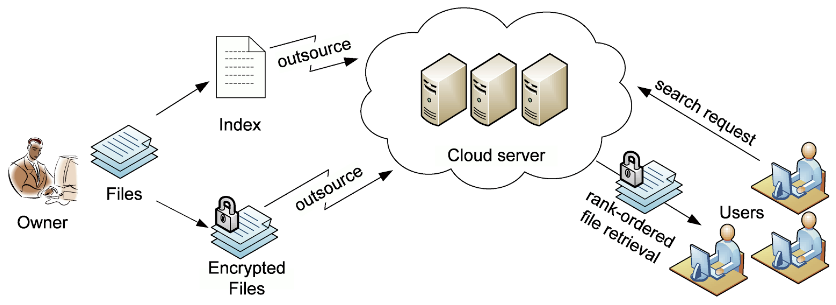

Figure 2.1: Architecture for the fuzzy keyword search

The above figure depicts the explained architecture clearly.The cloud also takes care of the access permissions the user has. The cloud only retrieves the files that the user has permission to access to. However in the current work, we are considering that all the users have access to all the files of the company.

2.1 Problem Description

Say user1 has queried the cloud using the search word **plsce**. Here the user has mispelt the word *place* as *plsce* by typing s(adjacent to a on keyboard) instead of a.

Now the keyword can be encrypted as such and sent to the cloud. Or the changes can be made it and then encrypted and sent to the cloud. We use the former method. We encrypt the wrongly spelled search term as it is and send it to the cloud and let the cloud take care of the corrections using homomorphic functions. We are considering corrections done to edit-distance 1 alone. The current work only does substitution. Insertion,Deletion Swapping and higher edit-distances can be further work that can be done.

2.1.1 Problem Statement

Define homomrphic function **Substitute()** such that they take the encrypted search word, index of the letter to be substituted from the left as input and outputs the corrected word. **Example**

$$Substitute(enc(plsce, 3) = enc(pl * ce)$$

CHAPTER 3

RELATED WORK

3.1 Traditional approach

The traditional method of searching was simply encryption followed by exact match. It could only fetch files that has exact match of keyword and search word. For example, lets say user queries using the search word **flower**. The word flower will be encrypted and sent to the cloud for file retrieval. The cloud then searches the encrypted keywords of the files to find if any of them matches *enc(flower)*. If the matche is found, the corresponding files are retrieved else none returned. The files with the tag **flowers** will not be retrieved because **enc(flowers)** is not the same as **flower**

3.2 Use of wildcard

Initial methods for fuzzy keyword search stated that if the search word is spelled right, it is encrypted and sent to the cloud. If it is wrong, all the meaningful corrections of word will be generated with edit-distance less a certain value(say 2), these set of corrected search words, is either shown to the user to choose from and the selected word is used for querying, or the entire set of corrected search words are encrypted and sent to the cloud to fetch the files.

There are drawbacks to this method. The search word might not be a meaningful English word, but a name of a person or some slang. So considering it incorrect isn't right. Also, the user needs to do the extra work of choosing the right word from the list of suggestions. Suggesting only meaningful words is also not a right idea because a word meaningless to the dictionary doesn't necessarily mean its meaningless.

Apart from the above logical errors there is also high memory utilisation, computation. For the word **plsce**, edit distance one, we can generate 2^{11} search words which is a very huge number, practically not possible to deal with. Also this number only increases with increasing length of the search word.

In order to avoid this huge number of search word being generated, wildcard technique is introduced. wildcard is like the joker in the game of cards that can take any value. wildcard is represented by the symbol *

Usage

pl*ce = place,plbce,plcce,pldce,plece, ..., plxce,plyce,plzce

* can take any value. By using wild card the number of search words generated drastically reduces to 11. The following are the set of search words generated.

*plsce, *lsce, p*lsce, p*sce, pl*sce, pl*ce, pls*ce, pls*e, plsc*e, plsc*, plsce*

Note that all these operations are done before they are encrypted and sent to the cloud. The method of using homomorphic functions, does this correction on the encrypted data it self, if the exact match isn't found, the cloud then uses these functions to find the closer word matches. It doesn't require to send a signal of no match and wait for the set of corrected terms. Overhead can be reduced.

CHAPTER 4

Algorithm

Let us consider the alphabetset { a,b,c }. Before doing the actual encryption the search word is modified a little. **Example**

consider the word **cab**. a-1, b-2, c-3 ,*-4/5/6/7/8/9 **cab** - 312 a,b,c are replaced by 1,2,3 respectively and the digits 4-9 represent wildcard.

After this is done, the modified search word is encrypted and sent to the cloud. In this case, enc(312) is sent to the cloud.

4.1 Homomorphic function for substitution

We need to achieve the function Sub(enc(word),n). Note that we assume that the length of the word is known. In this case it is 3. Let l denote the length of the word

The substitution function is defined as follows

$$sub(enc(word), n, l) = enc(word) + enc(10^{l-n})$$

Note that, the + symbol in the above equation represents homomorphic addition. For all the homomorphic functions, HELib is used.

Applying the sub() to the current example we get,

$$sub(enc(312), 2, 3) = enc(312) + enc(4 * (10^{3-2}))$$

$$= enc(312) + enc(40)$$

$$= enc(352)$$

We see that the function sub on application on the encrypted word cab, for substitution at 2, gave us enc(352) i.e C*B as expected.

Now, we need to match the generated output to the encrypted keywords.

Lets define a function match(a,b) which outputs 0 if it is exact match else it returns value other ¿0.

$$match(a, b) = (a - b) * (11..1(lones) - (10^{l-n}))$$

The - in the above equation denotes homomorphic subtraction and the * denotes homomorphic bitwise multiplication. The match function should output zero for all the encrypted keywords of the form 3*2 and not zero for all the other keywords.

4.2 Verification

Table 4.1: Verification of algorithm

enc(Key)	**match(enc(352),enc(Key))**	**enc(Key)**	**match(enc(352),enc(Key))**
enc(1)	NZ	enc(312)	0
enc(2)	NZ	enc(322)	0
enc(3)	NZ	enc(332)	0
enc(11)	NZ	enc(122)	NZ
enc(12)	NZ	enc(321)	NZ

The above table gives the output of match() as applied on keywords. We can see that we get zero's only for 3*2 kind of keywords and its non-zero(NZ) for all the other keywords. The table is not exhaustive.

Calculation of match function example:

$$match(enc(352), enc(312)) = enc(352) - enc(312) * (enc(111 - 10^{3-2}))$$

$$= enc(040) * (enc(101))$$

$$= enc(0)$$

4.3 Proof of Correctness

$$(enc(a) - enc(b)) * enc(11..1(ntimes) - 10^{l-n}) = 0$$

$$enc(a - b) = enc(kx(10^{l-n}))$$

(k is wildcard)

$$a - b = kx(10^{l-n})$$

$$b = a - kx(10^{l-n})$$

We can see that we get zero only for those b values, which replace the nth letter of a. Hence it is right.

4.4 Implementation

The implementation of the bitwise and, addition, subtraction is done using the HELib. The details of library are mentioned in Appendix A.

CHAPTER 5

CONCLUSION AND FUTURE WORKS

5.1 Summary

Importance and growth of cloud computing is discussed. Its advantages and downsides are mentioned. Security stands as the biggest concern.

The importance of fuzzy keyword search is explained. Work done so far in this area is discussed with illustrations. The drwbacks of the methods are explained. An attempt to develop an elegant way for fuzzy keyword search using homomorphic functions is made. Algorithm for partial fuzzy keyword search(substitution) is proposed and proved to be true. It is implemented using the homomorphic library HELib.

5.2 Future Work

Developing algorithms for deletion, insertion, swapping is the immediate possible work that can be done. Then trying to generate a common algorithm that can do all of them together will be a really challenging problem to solve.

Building a homomorphic library with tailored functions for the implementation of these algorithms will prove to be highly handy because there are very few implentations available.

APPENDIX A

How to Use : HELib - An implementation of homomorphic functions

A.1 Installation

To build HElib, you will need to have GMP and NTL libraries installed.

Many distributions come with GMP pre-installed. If not, you can install GMP as follows.

1. Download GMP from http://www.gmplib.org 2. uncompress and cd into the directory gmp-XXX 3. On the command line: ./configure make sudo make install 4. This should install GMP into /usr/local

Once GMP is installed, you can install NTL as follows: (NOTE: you will needs NTL v9.4.0 or higher)

1. Download NTL from http://www.shoup.net/ntl/download.html 2. uncompress and cd into the directory ntl-XXX/src 3. On the command line: ./configure NTL$_GMP_LIP = onmakesudomakeinstall4.ThisshouldinstallNTLinto/usr/local$

A.2 Usage

On the command line in the HElib src directory:

make

will compile and build the library fhe.a. Once built, run

make check

which will compile and runs a series of test programs.

If you want to build your own program based on HElib, the easiest way to do it is to write the program in a file called myprog.cpp and then run

make myprog$_x$

which will compile myprog.cpp and link in fhe.a and all required support libraries, and create the executable myprog$_x$.

REFERENCES

1. ”Fuzzy Keyword Search over encrypted data” by Jin Li, Qian Wang, Cong wang

2. ”Implementation of Fuzzy Keyword Search over Encrypted Data in Cloud Computing” by Narendra Shekokar, Dr.Kunjita Sampat, Chandni Chandawalla, Jahnavi Shah

3. ”Secure Cloud Computing through Homomorphic Encryption” by Maha TEBAA, Said EL HAJII