A NOVEL DISTRIBUTED SECURE AND PRIVACY PRESERVING SEARCH METHODOLOGY FOR ENCRYPTED BIG DATA

A THESIS

submitted by

LIJA MOHAN

for the award of the degree

of

DOCTOR OF PHILOSOPHY

DIVISION OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF ENGINEERING

COCHIN UNIVERSITY OF SCIENCE & TECHNOLOGY

JULY 2018

CERTIFICATE

This is to certify that the thesis entitled "**A Novel Distributed Secure and Privacy Preserving Search Methodology for Encrypted Big Data**" submitted by Ms. Lija Mohan to the Cochin University of Science & Technology, Kochi for the award of the degree of Doctor of Philosophy is a bonafide record of research work carried out by her under my supervision and guidance at the Division of Computer Science and Engineering, School of Engineering, Cochin University of Science and Technology. The content of this thesis, in full or in parts, have not been submitted to any other University or Institute for the award of any degree or diploma.

I further certify that the corrections and modifications suggested by the audience during the pre-synopsis seminar and recommended by the Doctoral Committee of Ms. Lija Mohan are incorporated in the thesis.

Dr. Sudheep Elayidom M,

Research Supervisor,

Professor,

Division of Computer Science & Engineering,

School of Engineering,

CUSAT, 682022.

Place: Kalamassery

Date:

DECLARATION

I hereby declare that the work presented in the thesis titled “A Novel Distributed Secure and Privacy Preserving Search Methodology for Encrypted Big Data” is based on the original research work carried out by me under the supervision and guidance of Dr. Sudheep Elayidom M, Professor, School of Engineering, for the award of the degree of Doctor of Philosophy with Cochin University of Science and Technology. I further declare that the contents of this thesis in full or in parts have not been submitted for the award of any degree, diploma, associateship, or any other title or recognition from any other University/Institution.

LIJA MOHAN

Reg No: 4702

Place: Kalamassery

Date:

Dedicated to all those who spend sleepless nights to make their dreams come true.

ABSTRACT

Big Data is evolving as a ‘gold mine’ for data analysts. Flexible & scalable algorithms and architectures are needed, to excavate the hidden treasures from this ‘gold mine’. Cloud architectures satisfy the requirements very well and have been found inevitable for Big Data storage and analysis. However, the outsourcing and resource sharing features of cloud deployment raise some security challenges to the confidentiality of data and privacy of users. Security challenges along with the pressing demand for adopting Big Data technologies together motivate the development of strong encryption algorithms, for the implementation of secure Big Data information retrieval.

Encrypting the data makes it difficult to retrieve the most matching documents with respect to the query keywords. The Server side document ranking based on Searchable Symmetric Encryption (SSE) and Order Preserving Encryption (OPE) schemes reduce data privacy. Therefore, the objective of my research is to develop a secure and privacy preserving data storage and retrieval scheme that supports multiple keyword searches, ranked information retrieval and scalability.

Well known techniques in information retrieval like the vector space model and TF-IDF, are utilized for similarity matching. Existing Fully Homomorphic Encryption scheme is modified to ensure scalable and privacy preserving user searches, for our encrypted information retrieval scenario. The proposed Modified Homomorphic Encryption Scheme (MHE) calculates similarity scores at the server side, by applying homomorphic operations on encrypted TF-IDF values. The queries issued by users are also encrypted using the MHE, to ensure privacy.

To eliminate any data leakages including statistical leakages or term distribution, a Dual Round Encrypted Information Retrieval (DREIR) scheme is proposed, which efficiently utilizes the processing power of the cloud server to compute the similarity scores, leaving the decryption and ranking to the client side. The stages of the DRIER scheme are accelerated by proposing a Map Reduce implementation powered by Hadoop. This ensures scalability and faster execution for Big Data. The performance and accuracy of the method is evaluated using AWS Hadoop Cluster and found to be scalable, efficient and practical. The proposed MHE scheme is tested for correctness and security. The thesis also covers how the proposed technique can be applied to security critical applications like Email servers and logging.

ACKNOWLEDGEMENT

The work presented in this thesis would not have been possible, without my close association with many people. I take this opportunity to extend my sincere gratitude and appreciation to all those who made this PhD thesis possible.

I am deeply indebted to my research supervisor and mentor, Dr. Sudheep Elayidom M, for his continuous support, guidance, ideas, time, patience, systematic review, motivation, and knowledge in Big Data and Information Retrieval, things that stimulated the success of my research. I am very much honored to be his first PhD student. I thank him from the bottom of my heart for having shown me this brighter way of research.

My special word of thanks to Dr. Sheena Mathew for being my Doctoral Committee member as well as a motivator and dependable person, with whom I can share anything. She is such an affectionate and understandable person.

A special person who gave valuable insights to the research is Padmashri Dr. S.K. Pal. He came to evaluate the DST Inspire Senior Research Fellowship. He gave a very positive feedback about the problem we are addressing, and also gave suggestions to develop it to applications that are useful in real world.

I thank all the teaching and non teaching staff at CUSAT for making my life here joyful. I sincerely thank all the teachers, who taught me from childhood as they have sown the seeds of knowledge in me. I fondly remember the friendship I had with the research scholars in our Division. Some of the names that are worth mentioning are Cerene and Sheikha. I also thank all the scholars of School of Engineering, for building a positive and constructive research culture here.

I would like to thank all the anonymous reviewers for giving critical suggestions, to improve this research. Also, my research life would not have been so comfortable without the fellowship and funding I availed from DST INSPIRE and Amazon Web Services.

It is again, with deepest love and affection that I remember my parents and mother in law. I learnt to aspire to a career in research from my parents in my childhood, and later from my husband. The warmest of thanks to my husband for supporting me in each and every way, believing in me permanently and inspiring me in all dimensions of life.

I dedicate my research to you all. I reached this last stage, only because of the support I received from each one of you. Thank you from the heart of my heart.

LIJA MOHAN

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

NOTATIONS

F	File
W	word
n	number of files
m	number of words
K	number of files to be retrieved
t	number of words in the query
λ	security parameter
ρ	bit length of noise
X	number of mappers
<<	very much less than
>>	very much greater than
D	Number of documents
{}	set
<k,v>	Key value pair
||x||	bit length of x

ABBREVIATIONS

BQ	Binary Query
EQ	Encrypted Query
ES	Encrypted Score
SI	Secure Index
SSE	Secure Searchable Index
FHE	Fully Homomorphic Encryption
MHE	Modified Homomorphic Encryption
FID	File Identifier
TopList	contains the file id of top k similar files
PK	Public Key
SK	Secret Key
OPE	Order Preserving Encryption
SSE	Searchable Symmetric Encryption
TF-IDF	Term Frequency - Inverse Document Frequency
AWS	Amazon Web Service
PPE	Property Preserving Encryption
ORAM	Oblivious Random Access Model
PEKS	Public Key Encryption with Keyword Search
GCD	Greatest Common Divisor
HElib	Homomorphic Encryption Library
DRIER	Dual Round Encrypted Information Retrieval

TRC2	Thomson Reuters Text Research Collection
HDFS	Hadoop Dostributed File System
BaMS	Balanced Multi-FileInputSplit
DNS	Domain Name Service
DDOS	Distributed Denial of Service
FNPKMN	Files stored in HDFS to the memory required by Namenode.
HAR	Hadoop Archive
SQ	Sequencer

Chapter 1
Introduction

“We can evade reality but we cannot evade the consequences of evading reality."

–Ayn Rand

This chapter introduces the need for ensuring secure and privacy preserving searches while outsourcing large amount of data to third party servers.

1.1 Introduction

Today, data is evolving at a very huge rate. World Wide Web, social media, organizational data, health care data, sensor data etc are all sources of data. To eliminate the difficulty in storage and retrieval of the data, people started outsourcing their data to service providers. This will enable easy and hassle free storage. But, outsourcing of sensitive data to third parties lead to severe security violations and affect the privacy of the user. In this thesis, we will define a novel and comprehensive approach to store and retrieve large amount of sensitive data without compromising the privacy of the user. The remaining sections of the chapter detail the importance and motivation of my research.

1.2 Introduction to Big Data

Large amount of data characterized by high volume, velocity, value, veracity and variety is termed as Big Data. Big Data can be in the form of texts, audio, video, images etc. According to a study conducted by the International Data Corporation (IDC) [66], the amount of data in the digital world will grow by 35 trillion gigabytes (1 gigabyte equivalent to 40 (four-drawer) file cabinets of text), between 2015 and 2025. That is greater than the number of stars in the whole universe!

1.3 Challenges associated with Big Data

Big Data storage, transfer and sharing should address a lot of challenges. As Big Data contains Petabytes to Xetabytes of data, storage itself has to be outsourced to some third party system. With this, the availability, security and integrity of the data are adversely affected. Hardware fault tolerance, end point validation etc are other factors to be taken care of. National Institute of Standards and Technology, NIST [1] has identified and classified the top 10 challenges in Big Data (Figure 1.1). Security and Privacy [76] is one of the major challenges that have been identified. Secure computations in distributed programming environment, security restriction for storing non-relational data, privacy preserving data analytics, data security by applying cryptographic techniques, granular access control etc are some of the other open problems.

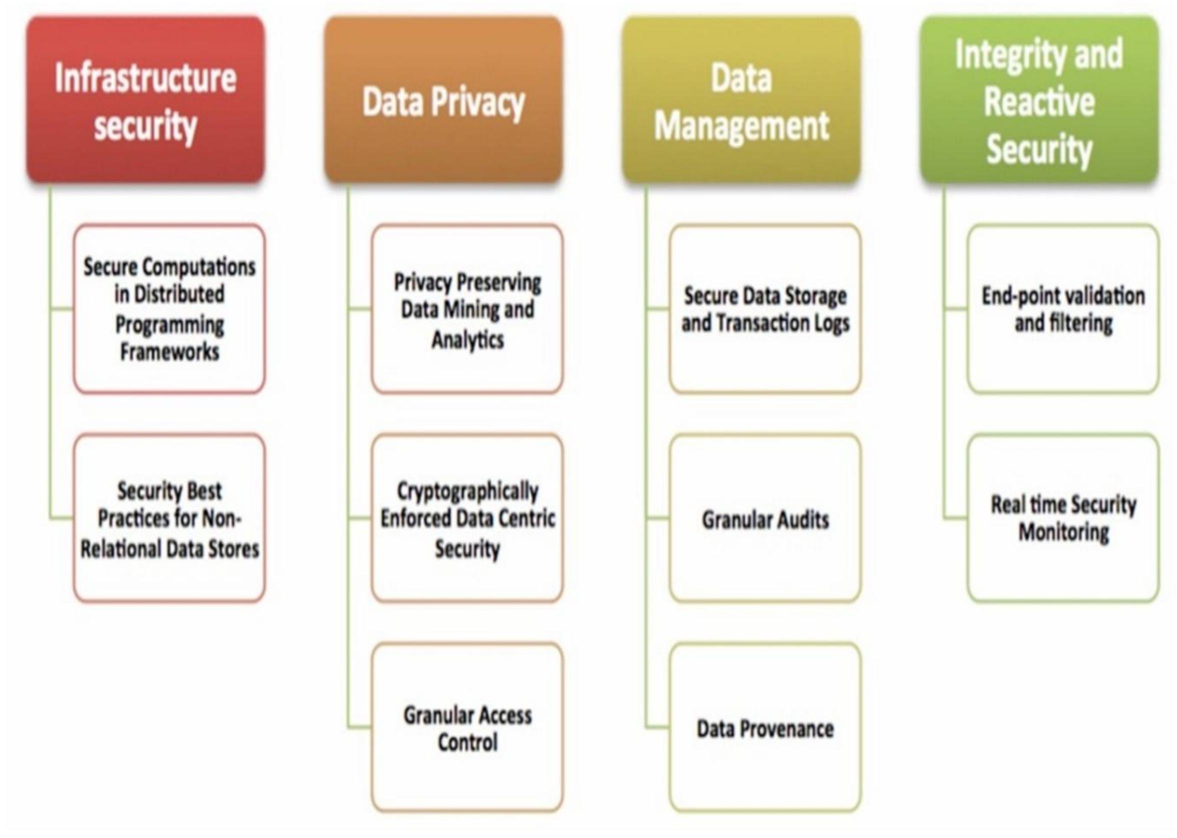

Infrastructure security
Secure Computations in Distributed Programming Frameworks
Security Best Practices for Non-Relational Data Stores
Data Privacy
Privacy Preserving Data Mining and Analytics
Cryptographically Enforced Data Centric Security
Granular Access Control
Data Management
Secure Data Storage and Transaction Logs
Granular Audits
Data Provenance
Integrity and Reactive Security
End-point validation and filtering
Real time Security Monitoring

Figure 1.1: Top 10 Big Data Challenges identified by NIST

1.4 Need for Big Data Security

To eliminate the hurdles associated with storing Big Data, organizations and people are moving to third party storage like Cloud [69]. Different service providers like Google, Amazon, Azure etc are capable of providing the needed resources for storage and processing. These providers follow a ‘pay as you go’ pricing strategy which makes the customers pay for the resources and bandwidth they consume. This architecture will allow scalability and easy maintenance. The downside of depending on a third party to store the data is security violation. The data may contain many sensitive information like personal details, documents of national importance, login details etc. Due to the distributed nature of storage, data owners may not be knowing where actually their data is getting stored and who all have access to their data. Also, every user wishes to preserve their privacy. They wouldn’t want any other person to know what they are searching for and what data they are accessing. That is, the user searches should be private. Since these conditions are not met in third party storage, a part of the population seems reluctant to adopt the cloud benefits. Also, data with sensitive nature are not stored in the cloud.

A researcher at security firm UpGuard [67] reportedly discovered a repository containing the names, addresses, account details and account PINs of 14 million Verizon customers in the U.S. Their AWS S3 bucket is owned and run by Nice Systems, a third-party vendor based in Israel that Verizon uses to handle its back-office and call center operations. The personal data of millions of Verizon customers was exposed because of a misconfigured Amazon Web Services S3 bucket leak.

On a Cloud Computing platform [75], the key security challenges to be addressed are shown in figure 1.2.

a. **Infrastructure Security:** Data is stored at third party machines. Intrusions, attacks, memory corruptions, physical damage etc may lead to loss of user's data.
b. **Data Management:** Data should be stored and replicated in different machines so as to enable easy recovery, high availability and fault tolerance.
c. **Data Privacy:** Even though users access data from third party machines, the service providers should not be able to see the queries, files or data issued and accessed by the users.
d. **Integrity and Reactive Security:** There should be provision for data owners to make sure that the uploaded data is present in the server as such and no modifications or access has been done by unauthorized entities. Attacks happening on data collection devices and programs should be monitored real time.

1.5 Challenges in ensuring Big Data Security

a. Data is too huge to store and process within a single system. Hence, there arises the need for a distributed storage. Distributed storage will impose a lot of additional security violations like storage theft, disk corruption, storage network intrusion, etc.
b. Distributed systems are difficult to maintain and program. The cost and effort associated with it is very high.

c. To eliminate the need for maintaining our own distributed clusters, the help of a service provider can be taken. But again, at the cost of a higher risk. Access control, location details, integrity, etc of the uploaded data will be unknown to the owner of the data.
d. Encryption can add security to the data. But, the operations that can be applied to encrypted data will be very much limited. For example, if the text is stored in encrypted form, we cannot perform a normal keyword search [74]. If the numerical data is encrypted and stored, we cannot perform normal addition or multiplication over it.
e. Privacy of the user is another concern. Service providers should not be able to monitor the queries issued and documents accessed by them.
f. Security and Privacy preserving technique should not add implementation complexity to the system and should allow the data analytics options.

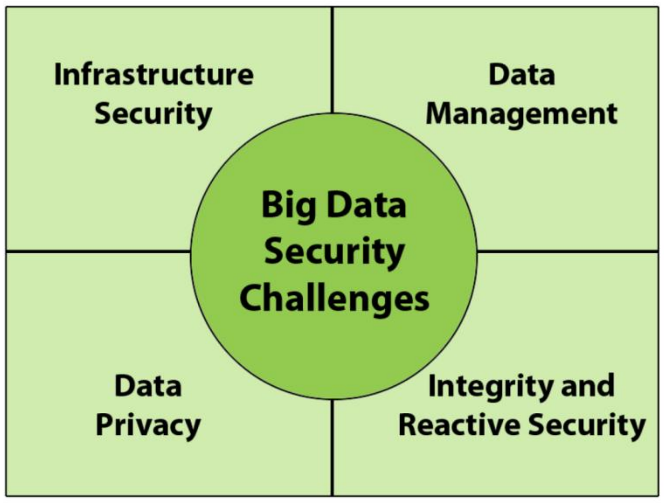


Figure 1.2: Cloud Computing Security Challenges

1.6 Challenges with secure and efficient mining of Big Data

Cloud Computing is now a paradigm shift to store, process and analyze a huge amount of data. It comes with benefits like low upfront setup & running cost, easy maintenance and efficient execution. However, genuine concerns are being put forward regarding the security and privacy associated with sensitive data. These fears are being further strengthened by the new security challenges posed by media [2] like the ‘PRISM’, ‘Breach of JPMorgan Chase’, ‘NSA Scandal’ etc.

The data stored in the cloud is always prone to attack mainly due to the anonymous, virtual, shared, replicated, multi-node storage and execution. Unauthorized access or manipulation leads to data theft or data loss. NIST [3] has identified ‘Data Security and Privacy’ as the major bottleneck to adopt the Cloud platform.

A sound encryption algorithm with efficient management of keys, is the only solution to the security issues put forth by cloud. Encryption is a ‘mathematical guardian wall’ built around your sensitive data. Even if your data is encrypted, compromising the key will lead to attacks on data. Hence, key management [2] also needs to be made efficient through techniques like split-key management or secret-key-sharing.

The retrieval of relevant information from the encrypted data [73] stored in the Cloud is the next issue to be solved. For e.g., consider a scenario where the company uploads day-to-day system logs to the cloud after encryption. If some problem occurs to the system, the system administrator will have to go through the activities that happened in that system. But, since the logs are encrypted, the only option available would be to download the entire logs, decrypt it at the client side and search for the logs of this particular system. In terms of the execution time as well as the network bandwidth consumption, this method is quite inefficient. In short, some other mechanism is needed to search within an encrypted domain, based on the keywords issued by the user. The problem of encrypted data searching [4] was first proposed by Song et.al, in the year 2000, but the problem became more significant with the evolution of cloud computing and distributed processing.

1.7 Problem Statement

Big Data is being outsourced and stored in an encrypted form, in a cloud system. The users issuing the keywords should retrieve the relevant files ranked according to the order of significance from this encrypted form, within a fraction of a second. The data should be securely stored in such a way so as to avoid any statistical leakage. To summarise, the objective of this research is to develop a multi-keyword ranked information retrieval, without revealing the statistical pattern on a Big Data domain.

Problem Statement: ***To design, implement and analyze a secure and privacy preserving multi-keyword ranked search on an encrypted big data domain.***

1.7.1 System Model

The proposed system consists of a model with three actors, the Data Owner who uploads the documents, the Data User, who searches for particular documents by issuing keywords and a Cloud Server who actually stores the data and does the computations to find matching documents. Authorization between data owner and user is assumed to be well established. An encrypted index is uploaded along with encrypted files for finding the similarity with the queried keywords. Figure 1.3 illustrates the system model.

1.7.2 Threat Model

Since, the cloud servers are third party systems, we assume them to be “honest but curious”. Apart from ensuring the security of the file contents, the index, search keywords and search pattern should also be hidden from the Cloud to preserve the privacy of each user.

1.7.3 Design Goals

a. Statistical Leakage

Even though, the vector space and the uploaded documents are in encrypted form, the cloud system can guess some document contents by analysing the search patterns and by predicting the term distribution, inter-distribution etc. Hence, the access patterns and search patterns should be hidden from a cloud provider. This can be explained in the context of encrypted log storage of a company on the cloud system, which was discussed in Section 4.6. Imagine the proxy logs of a company are stored in cloud in encrypted form. Since, it contains the domain details, the most frequently occurring word will be ‘www’, ‘allow’, etc. If the attacker cracks the encrypted form of ‘www’, subsequently he can crack ‘com’. Thus proceeding, entire file can be cracked.

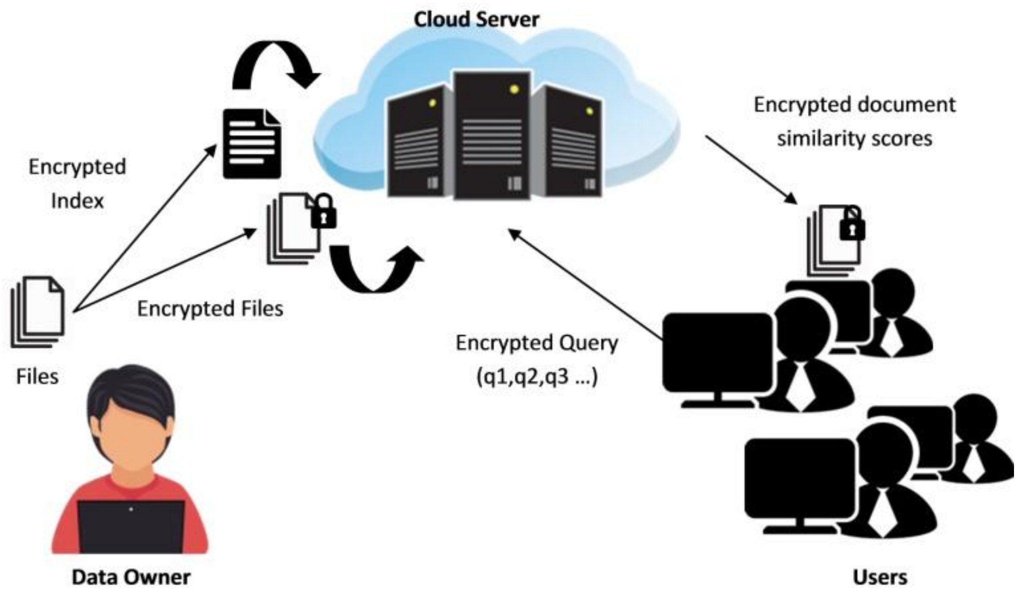

Cloud Server
Encrypted Index
Encrypted document similarity scores
Files
Encrypted Files
Encrypted Query (q1,q2,q3 ...)
Data Owner
Users

Figure 1.3: System Model

b. K-similarity Relevance

Based on the statistical measure on how frequently two or more terms co-occur in different documents, information leakage can exist. To rectify this, all keywords should be encrypted before applying the search function.

c. Scheme Robustness

If the ranking of documents is performed at the Cloud Server, then the keywords should be made available to the server. This may lead to information leakage. So as to overcome this, the ranking process is delegated to the client itself. This may increase the processing time and affect the speed of the ranking function, but will ensure better privacy to the user.

1.8 Motivation

Numerous researches were conducted on encrypted data searching techniques and a lot of solutions with Searchable Symmetric Encryption (SSE) [5, 70] were proposed, but most of them supported only boolean keyword search, which retrieves the document based on the presence or absence of a keyword. These schemes [4,6] that supported similarity of documents, were able to handle only one keyword. Even though there exists some work [7, 71] on multi-keyword ranked retrieval, they were not able to reduce the tradeoff between the security and efficiency of their implementations. Homomorphic encryption schemes proposed by Craig Gentry in 2009 [8-9] lead to further researches on encrypted data searching. But the method was not easy to adopt because of the difficulty in practical implementation. Here our principle objective is to utilise and extend the concepts of Homomorphic encryption [72] for an information retrieval scenario based on similarity matching. The scheme should also support multiple keyword searches.

In order to bring down the data leakage to a minimum, it is better to delegate all the encryption and decryption operations to the client side. But today as data is evolving at a tremendous rate, an efficient mechanism needs to be used to do this processing. Hence, distributed processing using Hadoop architecture has been proposed for faster and more secure information retrieval.

1.9 Real-time use-cases where encrypted data searching in Big Data becomes inevitable

1. Consider an organization maintaining their mail server in a Cloud System. To implement secure data sharing, the emails are stored in an encrypted form in the Cloud. Later if one user needs to retrieve the emails matching the query “Secret Mission X”, the system should support encrypted data searching.

2. A company having thousands of computers need to store the logs for a long period. Logs will help in auditing, troubleshooting, disaster recovery etc. But, maintaining the logs of the entire machines for a longer period, will consume a large part of the storage. To overcome this, logs can be stored in Cloud systems. But, raw storage of logs affects security as login or other critical details become

accessible to the attackers. Hence, logs should be encrypted before outsourcing. Then, how can the logs corresponding to a particular machine IP be retrieved, when something happens to that machine? The system should support encrypted data search.

3. In the healthcare domain, to store the health records in cloud system without compromising the security and privacy of the patients.

1.10 Contribution of the Thesis

The contribution of this thesis can be summarised as follows:

1. The thesis describes the existing problems associated with Big Data storage like security & privacy issues, and statistical data leakage and went through the existing methods to deal with encrypted data searching.

2. Modified Homomorphic Encryption (MHE) scheme is proposed, which is a variation of the existing homomorphic encryption scheme & can be practically applied to the information retrieval domain. The scheme retrieves relevant documents based on the similarity score, using generalised TF-IDF.

3. The proposed MHE scheme is tested for correctness, security and privacy.

4. A Map Reduce implementation powered by Hadoop is proposed for the MHE based information retrieval. A performance analysis done on the proposed scheme demonstrates higher efficiency, scalability and linear execution time for large amount of data.

1.11 Organisation of the Thesis

The thesis is organized as follows. Chapter 2 discusses the work that has already been done to retrieve data from encrypted domain. Chapter 3 defines the problem with preliminaries needed to understand the problem in deep. Chapter 4 gives a background knowledge needed to understand Homomorphic Encryption and its variants. Chapter 5 details the design of the proposed MHE scheme whereas Chapter 6 discusses the phases in development and implementation of the MHE scheme for Ranked Information Retrieval. Implementation of MHE using Map Reduce Programming model is explained in Chapter 7. Applications of the proposed MHE based information retrieval are handled by Chapter 8. Finally, Chapter 9 concludes the thesis and mentions the future scope of the research. Additional works done which supported the thesis indirectly to improve the design and implementation are included as Appendices. Appendix 1 lists the articles published as part of the research. Appendix 2 details how the information retrieval techniques used in our thesis can effectively applied in the context of predicting the winner of an election. Appendix 3 provides the CPP code used for implementing the MHE scheme.

Summary

This chapter outlines the security and privacy challenges associated with storing Big Data. Third party storage is the most viable solution but not considering the security factor. Encryption can be used to secure the outsourced data but at a cost of minimum computability on the data. Research aims to investigate on efficient and secure mechanisms that helps to store and retrieve the large amount of data in a third party system like cloud. The scheme should support multiple keyword search and ranked information retrieval. Also, the privacy of the user is another factor to be considered. Service providers should not be able to see the query terms or search results which compromise the privacy of the users.

Chapter 2

Literature Review

"Encryption enables protecting fundamental rights such as freedom of expression and the protection of personal data and ensures safe online commerce."

- European Commission, Sep 2017.

This chapter discusses the existing well known techniques available in literature that deals with encrypted data searching. The techniques are compared based on their complexity, security, key size needed etc.

2.1 Introduction

To implement solutions for Big Data storage as well as for problems involving severe computations, cloud computing is the most appropriate method. But use of the public cloud has resulted in a lot of security and privacy issues. Several reports [10-16] reveal about the security breaches and data theft that took place in real world scenarios.

‘Encryption of data’ [77] seems to be a first hand solution to ensure secrecy and privacy. But encryption limits the computations that can be performed on data, like retrieving a particular file containing a specific keyword or extracting features from an image etc. Always, there exists a trade-off between security and usability [90-92]. But the solution here is to apply the security mechanism in such a way that it will not limit the functionality as well.

2.2 Search on encrypted data

Encrypted data searching [93, 95, 96] techniques need to be developed to overcome many of the security and privacy issues associated with the cloud. The problem has got much significance because:

- Big Data storage is practical and easy with cloud architectures.
- To ensure secrecy, encryption can be applied, but users need to search, sort, compare different data stored in this encrypted format.
- We cannot completely trust the service providers.

This chapter analyses different encryption schemes suitable for implementing secure and privacy preserving applications in the cloud. Each technique differs in practicality, security, functionality and flexibility.

2.3 Review of Existing Solutions to Enable Encrypted Data Searching

Basically we identified six different ways to search on encrypted data, each based on one of the following cryptographic primitives:

2.3.1 Property Preserving Encryption
2.3.2 Functional Encryption
2.3.3 Fully-Homomorphic Encryption
2.3.4 Searchable Symmetric Encryption
2.3.5 Oblivious RAMs

2.3.1 Property Preserving Encryption (PPE)

PPE scheme [17] encrypt the text in such a way that it leaks certain properties of the underlying data. Different PPE schemes are proposed, based on the property that is leaked. The basic one is 'Deterministic Encryption' [18] in which one message always generates same cipher text after encryption. Thus, by comparing the cipher text, one can determine whether the messages are same. These types of encryptions are hence applicable to problems where similarity is compared.

For e.g., if 'm_1' encrypts to 'c_1' and 'm_2' encrypts to 'c_2', then by comparing the value of $c_1$and c_2 we can determine whether m_1 is equal to m_2.

Order Preserving Encryption (OPE) [19-23], Orthogonality Preserving Encryption etc are some variations of the Property Preserving Encryption. Bellare et. al. [21] proposed a method where PPE scheme can be efficiently applied on securing databases.

The High-Level Idea: Assume we have both a Deterministic encryption scheme E^D and a standard Randomized encryption scheme E^R. Then we can create an encrypted database EDB as follows.

For each record D_i in the database $(D_1 \ldots D_n)$, the user computes deterministic encryptions of each keyword, D_i. If each record, D_i has m keywords $(w_{i1}, w_{12}, \ldots\ldots, w_{im})$, the *EDB* then simply consists of n tuples.

$$r_i = (d_{i1}, d_{i2} \ldots d_{im}, ptr(c_i))$$

where $d_{ij}=E^D{}_{k2}(w_{i,j})$, $c_i=E^R{}_{k1}(D_i)$, and $ptr(c_i)$ is a pointer to cipher text C_i.

The user will finally send the encrypted database $EDB(r_1 \ldots. r_n)$ to the third party server along with the randomized encryptions of the records $(c_1 \ldots. c_n)$.

Consider a scenario where a user needs to search for a particular query term, w, find deterministic encryption of w and send this to the server.

Therefore, the token to be searched is $d_w = E^D_{K2}(w)$.

Every server that holds partitions of data will search for d_w by simple string comparison and if a match is found, by following the corresponding pointer, the corresponding record can be retrieved.

Theoretically, for all $1 <= i <= n$ and $1 <= j <= m$, server tests if $d_w = d_{ij}$ and if they are equal, it follows $ptr(c_i)$ to return c_i.

Computational Complexity: Search complexity is O(nm), where *'m'* is the number of documents. i.e., linear complexity. But data structures like Binary search trees can improve the speed.

Security of PPE: Since encryption on m1 always generates same cipher text, security is limited since this can lead to some statistical leakages.

2.3.2 Functional & Identity-Based Encryption

The concept behind Functional Encryption was first proposed by Sahai and Waters in a conference and later formalized and proved to be practical by Boneh, Sahai, Waters and by O'Neill [24]. Identity Based Encryption, Attribute Encryption, Predicate Encryption etc can be considered as variations of Functional Encryption. The working of Identity encryption can be explained by a simple real world application. For instance, Alice wants to send some secret message to Bob. Alice knows that Bob works at Google.

According to Functional Encryption, Google will initialize the security system by generating a pair of master keys *(msk,mpk)*, where one is a secret key and the other is public. Google then distributes *mpk* together with a valid certificate to its authorized employees.

To encrypt a message '*m'*, Alice will collect Google's Master Public key *mpk*, and apply the encryption algorithm on '*m'* using *mpk* and Bob's public identity, '*bob@google.com'*.

c= E(mpk,'bob@google.com',m)

For Bob, to decrypt the message *'c'*, Bob generates his secret key using Google's master key and his own id.

sk=KeyGen(msk,'bob@google.com')

Bob recovers the message by applying the decryption algorithm.

m=Dec(sk,c).

The advantage of this method is its simplicity. Without revealing any public key of Bob, Alice can send encrypted messages to him or any person in the organization, knowing only the public key of that organization.

In case of attribute based encryption, some attributes approved by the organization will be utilized for encryption. For e.g., consider a hospital domain. Alice needs to upload a file which can be viewed by a person if he is a 'doctor specialized in oncology with masters degree'. Hence the attributes can be 'doctor', 'MD', 'Oncology' etc.

Computational Complexity: Complexity is *O(nm)* as the algorithms has to try to decrypt each cipher text in the encrypted domain. But always m<<n, hence the time complexity needed will always be more, compared to PPE.

Security: This approach substantially ensures security, since neither statistical leakages nor brute force attacks exist in the system.

2.3.3 Fully Homomorphic Encryption

A cryptosystem that supports both addition and multiplication operations on encrypted data is called fully homomorphic encryption (FHE) and is far more powerful compared to the existing searchable encryption schemes. Homomorphic encryption schemes process data in its encrypted form itself. No decryption is needed. Thus, these types of applications are best suited for third party computations like cloud computing. Encrypted searching does not reveal any information to external agents.

Homomorphism with respect to addition or multiplication has been made possible since the development of RSA and paillier encryption [25]. They are called partial homomorphic systems. The concept of Fully Homomorphic encryption which made possible additions and multiplications over the encrypted data was first proposed by Gentry [8].

Craig Gentry developed lattice based cryptosystem to achieve fully homomorphic property and he was successful in evaluating arbitrary depth circuits. The scheme was also bootstrappable, which means as the circuit grows, the noise rises and ultimately the circuit will get capable of decrypting its own encrypted data i.e. the circuit gains self referential property. Even though the scheme had a strong mathematical base and correctness, it was not possible to implement the scheme for practical applications because the computational complexity was very high. Even a high performance system will consume more time in implementing the scheme. Cipher text size and computational time increased as the security level increased and the scheme is assumed to have a complexity 2^K, where k is the security level. Hence, in 2012, Gentry along with Vaikundanathan [14], proposed a variation of the original scheme using the property of ideal lattices over integers.

Idea of fully homomorphic encryption with a simple application context is as follows. Consider a 'Privacy preserving search on Google'. The owner uploads files in an encrypted format to Google server. When a query comes, Google will evaluate a certain function F on the encrypted data and return that result to the user. The user now decrypts that result to obtain the original files. Construction of F depends on the application context.

Let us illustrate the fully homomorphic symmetric encryption scheme with an example:

Let the shared secret key, p be an odd number 101. The domain consist of bits {0,1}. To encrypt m = 1; Choose a random value, q =9 and small prime number r =5.

Encryption (m) = $c = m + 2r + pq$ = 11 + 909 = 920

Here cipher text will always be close to a multiple of p.

Therefore, m ≈ LSB of distance to nearest multiple of p.

Decryption is m = $(c \% p) \% 2$ = 11 % 2 = 1.

Proof:

The proof for the correctness of the above mentioned scheme is:

$$m = c - p * [c/p] \% 2 = c - [c/p] \% 2 = LSB(c) - LSB([c/p])$$

If it is a Fully Homomorphic Public Encryption scheme then:

Secret key is an odd p as before. Public key is many "encryptions of 0" i.e.

$$x_i = q_i p + 2r_i$$

$$Enc_{pk}(m) = subset\text{-}sum(x_i\text{'s})+m+2r$$

$$Dec_{sk}(c) = (c \% p) \% 2$$

Computational Complexity: Comparing Fully-homomorphic encryption using integers and ideal lattices, the latter method has exponential complexity which is not at all tolerable. The Integer based method is assumed to have complexity λ^5. Table 2.1 describes the complexity details.

2.3.4 Searchable Encryption Schemes

The Searchable Encryption Scheme [79-83] allows users to search for a particular keyword from an encrypted domain. Authorized users possessing the requisite keys are able to create the trapdoors to retrieve information from an encrypted data. Trapdoor generation can be done based on a symmetric or asymmetric encryption scheme.

i. Asymmetric Searchable Encryption (ASE) Scheme: ASE scheme is best suited for storing data generated by one user and used by other users. Here, the public key of the user can be used for encrypting the data, and secret key of the intended recipient is used for decryption. Asymmetric encryption scheme like RSA can be utilised for encryption and decryption. Since, the implementation of ASE scheme requires elliptic curve, pairing, etc, the scheme is inefficient compared to the symmetric ASE version.

Table 2.1: Complexity Comparison

Dimension	**KeyGen**	**PK size**	**Re-Crypt**
512 200,000-bit integers	2.4 sec	17 MB	6 sec
2048 800,000-bit integers	40 sec	70 MB	31 sec
8192 3,200,000-bit integers	8 min	285 MB	3 min
32728 13,000,000-bit integers	2Hrs	2.3 GB	30 min

ii. Symmetric Searchable Encryption (SSE) Scheme: SSE schemes are more practical due to their simplicity in operations like hashing or block ciphering [88]. In functionality, it is preferred to store the data which is uploaded and accessed by the same set of users. Symmetric encryption schemes like AES can be used for the implementation.

Let D={D_1, D_2,…. D_N} denotes a set of documents.

Let W = {W_1, W_2, …W_L} be a set of keywords.

Let D(w) denote the set of documents which contain a keyword w ∈ W.

If X is a string, then |X| denotes the bit length of X. If X is a set, then |X| denotes the cardinality of X. Basically, any SSE scheme consists of five polynomial time algorithms

SSE = (Gen, Enc, Trapdoor, Search, Dec) such that

– K ← Gen(1^k): is a probabilistic algorithm which generates a key K, where k is a security parameter.

– (I, C) ← Enc(K, D, W): is a probabilistic encryption algorithm which outputs an encrypted index I and C = {C_1, ⋯ , C_N }, where C_i is a cipher text of D_i.

– t(w) ← Trapdoor(K, w): is a deterministic algorithm which outputs a trapdoor t(w) for a keyword w.

– (C(w), Tag) ← Search(I, C, t(w)): is a deterministic search algorithm, where

C(w) = {C_i | C_i is a cipher text of D_i ∈ D(w)}

Complexity: All the basic functions are executed in polynomial time. Apart from the basic encryption, SSE utilizes pseudo random functions and permutations which make the encryption scheme more complex. Additional storage space is required to store the encrypted index.

Security: Even though SSE schemes help to achieve only controlled searching, the scheme supports provable security. Server cannot learn anything from the queries issued. But, same word search may result in the same query which leads to search pattern leakage**.**

2.3.5 Oblivious RAM

Oblivious RAM concept was first proposed by Goldreich and Ostrovsky [25, 68] as a method to implement software protection on third party servers. But at that time it seemed irrelevant because cloud computing or third party computing were not at all in practice. But now, the work has gained so much application related to cloud storage.

An ORAM scheme basically consists of 3 stages Setup, Read and Write.

- *Setup*: inputs are
    - security parameter 1^K
    - *RAM* (memory array) of *N* items.
    - Outputs: Secret key *K* and an oblivious memory *ORAM*.
- *Read*: A two-party algorithm that runs between client and server. The client runs the Read function with a secret key *K* and an index *i* as input while the server runs the Read Function with an oblivious memory *ORAM* as input. At the end of the execution, the client receives *RAM[i]* while the server receives $\mathcal{E}$, i.e., null.

$$Read((K , i) ,ORAM) = (RAM[i] , \mathcal{E}).$$

- *Write* : Two party protocol executed between the client and a server. The client runs the Write function with a key *K*, an index *i* and a value *v* as input and the server runs the Write function with an oblivious memory *ORAM* as input. At the end of the protocol, the client receives nothing (again denoted as Ɛ) and the server receives an updated oblivious memory *ORAM'* such that the i^{th} location now holds the value v. This can be represented as,

$$Write ((K , i , v) , ORAM) = (\mathcal{E} , ORAM').$$

Oblivious RAM via FHE: In this section we will see how ORAM is related to FHE. Actually an implementation of ORAM is made possible by applying FHE concept. If FHE scheme can be represented with the following functions (Gen, Enc, Eval, Dec), then ORAM can be constructed as:

- *Setup*(1^k,*RAM*): Obtain a key for the *ORAM* scheme by computing $K=FHE.Gen(1^k)$ and encrypt *RAM* as $c = FHE.Enc_k(RAM)$. This cipher text *c* is kept as the oblivious memory ORAM.

    - *Read((K , i) ,ORAM):* Client encrypts the index '*i*' to generate $c_i=FHE.Enc_k(i)$ and sends c_i to the server. Server then computes

$$c' = FHE.Eval(f , ORAM , c_i)$$

C' is decrypted by client to recover RAM[i].

- Write ((K , i, v), ORAM) : Value generated apart from cipher text is

$c_v = FHE.Enc_k(v)$ and sends c and c_v to server so that the server can now compute $c' = FHE.Eval(g, ORAM, c_i, c_v)$ *where 'g' is a random number.*

Security of ORAM: ORAM is constructed such that server is unable to derive any information about RAM [94]. *Read* and *Writ*e functions do not leak information about the index and values either.

Computational Complexity: Since FHE has to be implemented in Read and Write phase, ORAM is the slowest of all techniques mentioned above.

Among the techniques that are discussed above, PPE, Functional encryption using attributes or IDs and SSE are the most commonly used ones. Public Key Encryption with Keyword Search (PEKS) [34, 78] proposed by Boneh *et al* was a preliminary work done on encrypted data searching which searches for a single keyword in the encrypted domain. Here, if Alice needs to secure her mails, she will store it in an encrypted form in the mail server and to retrieve a mail containing the keyword 'K', she will deliver a 'gateway key' to the server which enables the server to check for the presence of that keyword alone without revealing any other information. Confidentiality-Preserving Rank-Ordered Search [35] proposed by Swaminathan et.al. was another breakthrough in similarity based searching. Here, an inverted index is maintained to retrieve the documents based on the similarity to the keywords used in searching. But, the index is not encrypted.

A comparison of the different schemes based on the complexity of information retrieval is presented in Table 2. Search complexity implies the complexity of the algorithm represented in Big O notation. Search type indicated the way of searching. Linear implies a sequential search. A search scheme applied on the initially prepared index is termed as 'Pre-processed index'. In the case of SSE and Rank Ordered search, searching requires a recalculation of scores based on the scores present in index, since the index is stored in encrypted form.

Based on the ability to search, a comparison is provided in Table 3.Here, ‘Exact Match’, ‘Sub-Match’ [85-87], ’Case Insensitivity’, ‘Regex’ [84], ‘Proximity’ [89] and ‘Stemming’ implies, whether the scheme supports ‘word to word match’, ‘substring matching’, ‘matching irrespective of case of the letters’, ‘matching based on regular expression’, ‘matching based on difference of two or three letters’, and ‘matching of base words’.

Table 2.2: Critical Comparison of searching schemes

Scheme	Search Complexity	Search Type	Insert Requires Recalculation?
PPE	O(n)	Linear	No
Functional Encryption	O(d)	Pre-processed Index	No
SSE	O(1)	Pre-processed Index	Yes
PEKS [34]	O(n)	Linear	No
Rank Ordered [35]	O(d)	Pre-processed Index	Yes

Table 2.3: Summary of major search schemes and their ability to perform certain search options

Scheme	**Exact Match**	**Sub Match**	**Case Insensitivity**	**Regex**	**Proximity**	**Stemming**
Practical Technique	Yes	No	No	No	Yes	No
Secure Indexes	Yes	Maybe	Maybe	No	No	Maybe
SSE	Yes	Maybe	Maybe	No	No	Maybe
PEKS	Yes	Maybe	Maybe	No	No	Maybe
Rank Ordered	No	No	Yes	No	No	Yes

Table 2.4: Summary of different encryption schemes

Scheme	**Summary**
PPE	Fast search, but at the expense of information leakage.
Functional Encryption	Easy implementation, Secure but slow search time
FHE	Secure but application dependant, We should choose a homomorphic function based on the application context in which it is implemented.

SSE	Scheme is provably secure but leaks the search pattern.
ORAM	Most secure solution which hides even the access pattern. But, the implementation complexity is very high.

Summary

Cloud computing is gaining much interest due to the huge amount of data generated and the need for computations to be performed on these data. Security and privacy is the only factor that hinders the usability of cloud. Users of data do not trust a third party agent like cloud to store their sensitive data. The solution is encryption. But encryption limits the computability and searchability of data. To overcome such limitations, we can choose encryptions that properly match each application, reducing the tradeoff between security and usability. This chapter surveys different encryption schemes available in literature and compare them based on factors like security, complexity etc. From the study, my research adopted a Fully Homomorphic Encryption based technique to implement secure and privacy preserving ranked retrieval.

Chapter 3

Preliminaries and Background for Ranked Information Retrieval

"A human is not a device that reliably reports a gold standard judgment of relevance of a document to a query."

— Henrich Schutz, Introduction to Information Retrieval

This chapter explains the theory and existing techniques used for information retrieval scenario.

3.1 Introduction

This chapter discusses the basic concepts adopted from Information Retrieval domain that served as the backbone of our secure and privacy preserving information retrieval technique. The well-known techniques for information retrieval, like vector space model [26] and TF-IDF [27], are utilized for retrieving the relevant documents. The search similarity index thus generated is encrypted using homomorphic encryption [28] scheme and encrypted functions are applied on it to retrieve the similar document indices. This list is then sent to the client side and the ranking is done there by decrypting the obtained indices and sorting them based on their similarity score.

3.2 Components of an Information Retrieval System

The basic web information retrieval system has the following functions.

1. The system browses the document collection and fetches documents - Crawling

2. The system builds an index of the documents – Indexing

3. User gives the query - Searching

4. The system retrieves documents that are relevant to the query from the index and displays that to the user - Ranking

5. User may give relevance feedback to the search engine - Relevance Feedback.

The goal of any information retrieval system is to satisfy user’s information need. Unfortunately, characterization of user information need is not simple. Users often do not know clearly about the information need. Query is only a vague and incomplete description of the information need. Query operations like query expansion, stop word removal etc are usually done on the query.

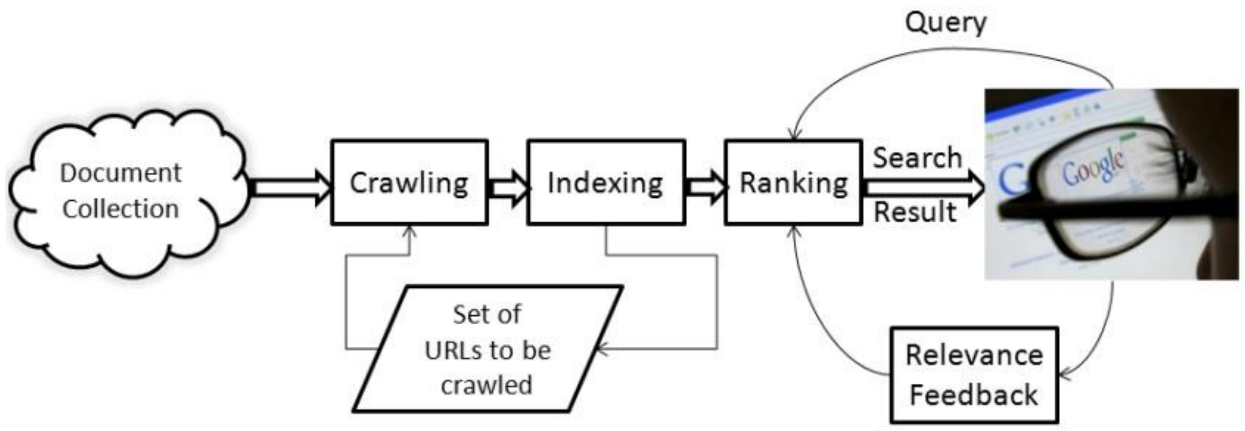


Figure 3.1 Basic Information Retrieval System

3.3 Information Retrieval System – Formal Definition

Theoretical models of IR show many different ways in which the IR problem can be formulated and solved. Formally, the IR model can be defined as a 4-tuple **[D, Q, F, R(q_i , d_j)]** where

D is document collection. In most of the modeling approaches (Boolean, Vector or probabilistic) each document is modeled as a bag of index terms where index terms are assumed to be independent of each other. This way the semantics of the document is lost.

Q is the query collection. The queries fired by the user belong to this set. It is also modeled as a bag of index terms in most of the cases.

F is the framework for modeling document representations, queries and their relationship.

R(q_i , d_j) is a ranking function which associates a score (real number) with the pair (q_i , d_j) where $q_i \in Q$ and $d_j \in D$. Given the query (q_i) the documents are ranked according to this score.

3.4 Classical Model of Information Retrieval

This section discusses three classical models of IR; namely, Boolean, Vector Space and Probabilistic Model.

a. Boolean Model

Boolean Model is one of the oldest and simplest models of Information Retrieval. It is based on set theory and Boolean algebra.(Baeza-Yates and Ribeiro-Neto, 1999) In this model, each document is taken as a bag of index terms. Index terms are simply words or phrases from the document that are important to establish the meaning of the document. The query is a Boolean algebra expression using connectives like $\wedge$, $\vee$, $\neg$ etc. The documents retrieved are the documents that completely match the given query. Partial matches are not retrieved. Also, the retrieved set of documents is not ordered.

Advantages:

• It is simple, efficient and easy to implement.

• It was one of the earliest retrieval methods to be implemented. It remained the primary retrieval model for at least three decades.

• It is very precise in nature. The user exactly gets what is specified.

• Boolean model is still widely used in small scale searches like searching emails, files from local hard drives or in a mid-sized library.

Disadvantages:

• In Boolean model, the retrieval strategy is based on binary criteria. So, partial matches are not retrieved. Only those documents that exactly match the query are retrieved. Hence, to effectively retrieve from a large set of documents users must have a good domain knowledge to form good queries.

• The retrieved documents are not ranked.

• Given a large set of documents, say, at web scale, the Boolean model either retrieves too many documents or very few documents.

• The reason of the above is: users usually do not form complex queries. Either they use very few (often a single) term fetching a tremendously large list of unordered documents. Else, they use a large set of terms joined by AND. This fetches very few documents.

• The model does not use term weights. Even if a term occurs once in a document or several times, it is treated the same.

b. Vector Space Model

The main problem with Boolean model is its inability to fetch partial matches and the absence of any scoring procedure to rank the retrieved documents. This problem was addressed in the vector based model of information retrieval.

In this model documents are represented as a vector of index terms.

$$d_j = \{w_{1j}, w_{2j}, \ldots, w_{tj}\}$$

where t is the total number of index terms in the collection of documents. Each $w_{ij} > 0$ if and only if the term i is present in document d_j . Unlike Boolean model, we do not consider only presence or absence of terms. So in vector model, these term weights are not binary. Like documents, queries are also represented as vectors in a similar way. The similarity between the query vector and document vector is a measure of relevance of the document and used as a ranking score. The similarity between document vector and query vector is usually calculated as the cosine similarity (equation 3.1) between them. If the similarity is greater than a predefined threshold, the document is retrieved.

If $\mathbf{d}_2$ and $\mathbf{q}$ are tf-idf vectors, then

$$\cos\theta = \frac{\mathbf{d_2} \cdot \mathbf{q}}{\|\mathbf{d_2}\| \, \|\mathbf{q}\|} \qquad \ldots\ldots\ldots\ldots (3.1)$$

Advantages:

• The cosine similarity measure returns value in the range 0 to 1. Hence, partial matching is possible.

• Ranking of the retrieved results according to the cosine similarity score is possible.

Disadvantages:

• Index terms are considered to be mutually independent. Thus, this model does not capture the semantics of the query or the document.

• It cannot denote the “clear logic view” like Boolean model.

Despite its simplicity, the vector based model works well with general collections. It is widely used in practical systems.

c. Probabilistic Model

In probabilistic model we try to capture the information retrieval process from a probabilistic framework. The basic idea is to retrieve the documents according to the probability of the document being relevant. The basic question the system needs to ask before deciding whether to retrieve a document or not is: ‘What is the probability of this document being relevant given this query’. Here it is assumed that the relevance of a document to a query is independent of other documents in collection. Probabilistic ranking principle:

If a reference retrieval system’s response to each request is a ranking of the documents in the collection in order of decreasing probability of relevance to the user who submitted the request, where the probabilities are estimated as accurately as possible on the basis of whatever data have been made available to the system for this purpose, the overall effectiveness of the system to its user will be the best that is obtainable on the basis of that data.

Advantage:

Higher accuracy compared to other models

Disadvantage:

Not suitable for large scale data mining.

3.5 Indexing of the documents

Text indexing will really improve the retrieval efficiency of documents. Search engines retrieving a subset of documents that exactly match the keywords issued by the user within a fraction of the second would not have been possible without indexing. Instead of searching through the entire documents, these engines will traverse through the index that is created at the time of document upload. Documents can be indexed with the meta-data or the full text they contains. Indexing is essential for storage and retrieval of unstructured data like emails, research reports, news contents, etc.

Indexing involves several stages as given below:

Tokenization: Extracts words from text.

Eg : The President is invited for the party. -> [the, president, is, invited, for, the, party]

Stop Word Elimination: Eliminate frequently occurring words like prepositions, conjunctions, articles, etc that do not have any disambiguation power.

Eg: [The, President, Is, invited, for, the, party] -> [president, invited, party]

Stemming: Obtain the root form of words.

Eg: [President, invited, party] -> [president, invite, party]

Inverted List Creation: Words matched to the list containing documents they contain.

Eg:

Doc 1: [president, invite, party]

Doc2: [president, address, people]

Inverted List:

President	Doc1, Doc2
Invite	Doc1

Party	Doc1
Address	Doc2
People	Doc2

3.6 Ranked Information Retrieval of the Documents

User searching for the documents will issue the queries without knowing the details of the words in the index. They might issue multiple keywords and a large number of matching files will be retrieved. To make the search results useful for the user, the results should be ranked based on the similarity with the query issued.

Given a document collection D = {d_1, d_2, … d_t} and a query, Q={q_1, q_2, … q_m}, the ranked information retrieval algorithm should calculate a score, S_i= similarity(Q, D_i) based on the similarity between query set and each document d_i in D. Files are then displayed in the descending order of S_i.

Score can be calculated based on Boolean model and Vector based approach. In the Boolean model we will only search for the presence or absence of the keywords. Hence, partial matches are not considered. To implement a better ranking scheme, vector space model is adopted in our approach. Here, instead of just boolean values, each term is associated with a term weight. Term weight defines the strength of relation between a word and a document. Method below illustrates the steps in constructing Vector Space Model.

a. TF-IDF Calculation

Term Frequency – Inverse Document Frequency is a statistical measure used to evaluate the importance of a word in a document, or a corpus. Term Frequency implies the cardinality of occurrence of each word in a document and Inverse Document Frequency implies the importance of a word in the entire corpus.

$$TF_{i,j} = \frac{Nij}{\sum Nkj} \qquad \text{……………..} \ (3.2)$$

Where TF_{ij}implies the term frequency of *an* ith word in jth document, N_{ij} implies the frequency of occurrence of ith word in jth document and $\sum N_{kj}$ implies the total number of words in the jth document. Since we are dealing with Big Data, we utilise a normalised TF value for further evaluations.

$$TF_{nij} = \frac{TFij}{\max(TF)} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(3.3)}$$

Where TF_{nij} implies the normalised TF value for the ith word in the jth document and max(TF) implies the maximum value for TF obtained for any word in the document collection.

$$IDF_i = 1 + log\frac{|D|}{|Fi|} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \text{(3.4)}$$

where |D| implies total number of documents in the corpus and $|F_i|$ implies total number of occurrence of terms in the corpus.

b. Vector Space Model

Vector Space model [29] represents text documents in rows and columns, where the rows are distinct words, and the columns are documents in the corpus and each cell represents the degree to which each word belongs to a document. TF-IDF is used as the metric to represent the degree of relevance of words in a document. This model represents documents and words as a vector.

$$\text{Document collection, } D_t = (d_1, d_2, d_3, \ldots, d_t)$$

$$\text{Word Collection, } W_k = (w_1, w_2, w_3, \ldots, w_k)$$

If D_t is arranged in columns and W_k in rows, each cell, C_{tk} represents the similarity score.

When a query comes with x words, $Q_x = (w_1, w_2 \ldots w_x)$, the similarity of the document is identified by equation 3.5.

$$\textit{Similarity Score, } S_t = \sum_{i=1}^{x} Cit * Bi \quad \ldots\ldots \text{(3.5)}$$

Here, B_i has a value 0/1, depending on whether the word is present in the query list or not.

After obtaining the similarity score for ‘t’ documents, they are ranked in order to find the most similar documents.

c. *Generalised Vector Space Model*

For lengthy documents, the vector space model might not be a good information retrieval technique. More number of words and fewer documents will result in lengthening of the index, without improving scalar product values. Hence, the similarity score obtained will not be accurate. To rectify this, the generalised vector space model [30] can be utilised. Here, term to term correlation is measured rather than the pair wise orthogonality as that of vector space. Hence, the similarity function now becomes;

$$Similarity\_Score,\ S_t = \frac{\sum_{j=1}^{n}\sum_{i=1}^{n} Wik * Wjq * Ti.Tj}{\sqrt{\sum_{i=1}^{n} w_{i,k}^2} * \sqrt{\sum_{i=1}^{n} w_{i,q}^2}} \ \ldots\ldots \quad (3.6)$$

where t_i and t_j are term correlation vectors of n-dimensional space. Term correlation vectors can be implemented using term occurrence frequency matrix [31] or semantic correlations [32].

Algorithm 3.1 summarises the steps for implementing a vector space model.

Algorithm 3.1: Ranked Retrieval Using Vector Space Implementation

```
Input: Vector Space arranged in $t$ columns for documents and $k$ words as rows
Output: Sorted List of Documents based on score.
Steps:
For each document $d_i$ in the document collection
    for every query term $q_j$ in Q do
        retrieve the $TF\text{-}IDF_{ij}$ for $q_j$ from the vector space
        Calculate $score(d_i) = score(d_i) + TF\text{-}IDF_{ij}$
    End
End
Sort documents according to their score
Return the sorted list of documents.
```

Summary

This chapter summarizes the basic concepts in information retrieval discussing the existing methods like Boolean, Vector Space and Probabilistic models. The chapter also outlines the methods to help implementing the IR model like indexing, vector space construction, score calculation, etc to retrieve files in ranked order of similarity with multiple query terms. My research adopted vector space model based information retrieval technique because they are proved to be the best in literature and widely used by search engines, mail servers etc.

Chapter 4

Preliminaries and Background on Homomorphic Encryption

"Alice, the owner of a jewelry store wants her employees to assemble precious materials into finished jewelry, but she is worried about theft. She addresses the problem by constructing glove boxes for which only she has the key."

- Craig Gentry.

Provides a formal background on homomorphic properties and explains different encryption algorithms that support homomorphism.

4.1 Introduction

Homomorphic encryption is a form of encryption which allows specific types of computations to be carried out on cipher texts, and generates an encrypted result which, when decrypted, matches the result of operations performed on the plain texts. This is a desirable feature in modern communication system architectures. RSA [103] is the first public-key encryption scheme with a homomorphic property. However, for security, RSA has to pad a message with random bits before encryption [97-98], to achieve semantic security. The padding results in RSA losing the homomorphic property. To avoid padding messages, many public-key encryption schemes with various homomorphic properties have been proposed in the last three decades. In this chapter, basic homomorphic encryption techniques are discussed. It begins with a formal definition of homomorphic encryption, followed by some well-known homomorphic encryption schemes.

4.2 Homomorphic Encryption- Definition

In abstract algebra, a homomorphism is a structure-preserving map between two algebraic structures, such as groups.

A group is a set G, together with an operation $\circ$, (called the group law of G) that combines any two elements a & b, to form another element, denoted as $a \circ b$. To qualify as a group, the set and operation (G, $\circ$), must satisfy four requirements known as the group axioms:

- **Closure**: For all a, b in G, the result of the operation, $a \circ b$, is also in G.
- **Associativity**: For all a,b, and c in G, $(a \circ b) \circ c = a \circ (b \circ c)$
- **Identity element**: There exists an element e in G, such that for every element a in G, the equality $e \circ a = a \circ e = a$ holds. Such an element is unique, and is called the identity element.
- **Inverse element**: For each a in G, there exists an element b in G such that $a \circ b = b \circ a = e$, where e is the identity element. The identity element of a group G is often written as 1.

The result of an operation may depend on the order of the operands. In other words, the result of combining element a with element b, need not yield the same result as combining element b with element a; the equation $a \circ b = b \circ a$ may not always be true. This equation always holds in the group of integers under addition, because a+b = b+a, for any two integers (commutativity of addition). Groups for which the commutativity equation $a \circ b = b \circ a$ always holds, are called abelian groups.

Given two groups (G, $\Diamond$) and (H,$\circ$), a group homomorphism from (G, $\Diamond$) to (H,$\circ$) is a function f : G->H, such that for all g and g' in G,

$$F(g \Diamond g') = f(g) \circ f(g') \text{ …….. } 4.1$$

Let (P,C,K,E,D) be an encryption scheme, where P,C are the plain text and cipher text spaces, K is the key space, and E,D are the encryption and decryption algorithms. Assume that the plain texts form a group (P,$\Diamond$) and the cipher texts form a group (C, $\circ$), then the encryption algorithm E is a map from the group P to the group C, i.e., E_k : P -> C, where k ε K is either a secret key (in a secret key cryptosystem) or a public key (in a public-key cryptosystem [105]).

For all a and b in P and k in K, if $E_k(a) \circ E_k(b) = E_k(a \Diamond b)$, the encryption scheme is homomorphic.

In an unpadded RSA [36], assume that the public key pk=(n,e), the plaintexts form a group (P,*), and the cipher texts form a group (C,*), where * is the modular multiplication. For any two plain texts m1* m2 in P, it holds that

$$\begin{aligned} E(m1,pk)*e(m2,pk) &= m_1^e * m_2^e \ (mod\ n) \\ &= (m_1*m_2)^e \ (mod\ n) \\ &= E(m1*m2,pk) \end{aligned}$$

Therefore, the unpadded RSA has the homomorphic property. Unfortunately, the unpadded RSA is insecure.

4.3 Fully Homomorphic Encryption

Homomorphic encryption is a very useful tool with a number of attractive applications. However, the applications are limited by the fact that only one operation is possible (usually addition or multiplication in the plain text space) so as to manipulate the plain text by using only the cipher text. What would really be useful is to be able to utilize both addition and multiplication simultaneously. This would permit more manipulation of the plain text by modifying the cipher text. In fact, this would allow one without the secret key to compute any efficiently computable function on the plain text, when given only the cipher text. Fully homomorphic encryption (FHE) techniques allow one to evaluate both addition and multiplication of plaintext, while remaining encrypted. The concept of FHE was introduced by Rivest [14] under the name privacy homomorphism. The problem of constructing a scheme with these properties remained unsolved until 2009, when Gentry [6] presented his breakthrough result. His scheme allows arbitrary computation on the cipher texts, and it yields the correct result when decrypted.

4.4 Fully Homomorphic Encryption - Definition

Fully homomorphic encryption can be considered as ring homomorphism. In mathematics, a ring is a set R equipped with two operations + and x, satisfying the following eight axioms called the ring axioms. R is an abelian group under addition, meaning:

1. (a + b) + c = a + (b + c) for all a,b,c in R (+ is associative).
2. There is an element 0 in R such that a + 0 = 0 + a = a (0 is the additive identity).
3. For each a in R there exists -a in R such that a + (-a) = (-a) + a = 0 (-a is the additive inverse of a).
4. a+b = b+a for all a and b in R (C is commutative).

R is a monoid under multiplication, meaning:

5. (a .b).c = a.(b.c) for all a,b,c in R (. is associative).
6. There is an element 1 in R such that a.1=a, and 1.a=a (1 is the multiplicative identity).

Multiplication distributes over addition:

7. a.(b+c) = (a.b)+(a.c) for all a, b, c in R (left distributivity).
8. (b+c).a = (b.a) + (b.c) for all a,b,c in R (right distributivity).

A ring homomorphism is a function between two rings which respect the structure. More explicitly, if R and S are two rings, then a ring homomorphism is a function **f: R -> S** such that **f (a+b) = f(a) + f(b)** and **f(a.b) = f(a) . f(b)** for all a and b in R.

Let us see an example of ring homomorphism. Consider the function f: Z_2 -> Z_2 given by $f(x) = x^2$ where x = 0 or 1.

First,

$$f(x+y) = (x+y)^2 = x^2 + 2xy + y^2 = x^2+y^2 = f(x) + f(y)$$

where 2xy = 0 because 2 times anything is 0 in Z_2.

Next,

$$f(x.y) = (xy)^2 = x^2 * y^2 = f(x) * f(y)$$

The second equality follows from the fact that Z_2 is commutative. Thus, f is a ring homomorphism.

Let (P,C,K,E, D) be a encryption scheme, where P,C are the plain text and cipher text spaces, K is the key space, and E,D are the encryption and decryption algorithms. Assume that the plain texts form a ring (P, $\oplus_p$,$\otimes_p$) and the cipher texts form a ring (C, $\oplus_c$, $\otimes_c$); then the encryption algorithm E is a map from the ring P to the ring C, i.e., E_k : P -> C, where k ε K is either a secret key (in the secret key cryptosystem) or a public key (in the public-key cryptosystem).

For all a and b in P and k in K, if

$$E_k(a) \oplus_c E_k(b) = E_k(a \oplus_p b)$$

$$E_k(a) \otimes_c E_k(b) = E_k(a \otimes_p b)$$

Then the encryption scheme is **fully homomorphic.**

4.5 Overview of Fully Homomorphic Encryption

Craig Gentry [8, 37], using lattice-based cryptography, showed the first fully homomorphic encryption scheme as announced by IBM on 25 June 2009. His scheme supports evaluations of arbitrary depth circuits. His construction starts from a somewhat homomorphic encryption scheme using ideal lattices, that is limited to evaluating low-degree polynomials over encrypted data. It is limited because each cipher text is noisy in some sense, and this noise grows as one adds and multiplies cipher texts, until ultimately the noise makes the resulting cipher text indecipherable. He then shows how to modify this scheme to make it bootstrappable—in particular, he shows that by modifying the somewhat homomorphic scheme slightly, it can actually evaluate its own decryption circuit, a self-referential property. Finally, he showed that any bootstrappable somewhat homomorphic encryption scheme can be converted into a fully homomorphic encryption, through a recursive self-embedding.

In the particular case of Gentry's ideal-lattice-based somewhat homomorphic scheme, this bootstrapping procedure effectively "refreshes" the cipher text by reducing its associated noise so that it can be used thereafter in more additions and multiplications, without resulting in an indecipherable cipher text. Gentry based the security of his scheme on the assumed hardness of two problems: certain worst-case problems over ideal lattices and the sparse (or low-weight) subset sum problem.

Regarding performance, the cipher texts in Gentry's scheme remain compact so far as their lengths do not depend at all on the complexity of the function that is evaluated over the encrypted data. The computational time only depends linearly on the number of operations performed. However, the scheme is impractical for many applications, because the cipher text size and computation time increase sharply as one increases the security level. To obtain 2^k security against known attacks, the computation time and cipher text size are high-degree polynomials in k. Stehle and Steinfeld [38] reduced the dependence on k substantially. They presented optimizations that permit the computation to be only quasi-$k^{3.5}$ per Boolean gate of the function being evaluated.

In 2009, Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan [9] presented a second fully homomorphic encryption scheme, which used many of the tools of Gentry's construction, but did not require ideal lattices. They showed that the somewhat homomorphic component of Gentry's ideal lattice based scheme can be replaced with a very simple somewhat homomorphic scheme that uses integers. The scheme was conceptually simpler than Gentry's ideal lattice scheme, but had similar properties when it came to homomorphic operations and efficiency.

In 2010, Nigel P. Smart and Frederik Vercauteren [39] presented a fully homomorphic encryption scheme with smaller key and cipher text sizes. The Smart– Vercauteren scheme followed the fully homomorphic construction based on ideal lattices given by Gentry. It also produced a fully homomorphic scheme from a somewhat homomorphic scheme. For the somewhat homomorphic scheme, the public and the private keys consisted of two large integers (one of which was shared by both the public and the private keys), and the cipher text consisted of one large integer. The Smart–Vercauteren scheme had smaller cipher text and reduced key size than Gentry's scheme, which was based on ideal lattices. Moreover, the scheme also allowed efficient fully homomorphic encryption over any field of characteristic two. However, the major problem with this scheme was that the key generation method was very slow. Hence, this scheme was still not fully practical.

At the rump session of Eurocrypt 2011, Craig Gentry and Shai Halevi [40] presented a working implementation of fully homomorphic encryption (i.e., the entire bootstrapping procedure) together with performance numbers.

Recently, Coron, Naccache, and Tibouchi [41] proposed a technique that would reduce the public-key size of the van Dijk et al. scheme to 600 KB. In April 2013 the HElib [42] was released, via GitHub, to the open source community which implements the Brakerski-Gentry-Vaikuntanathan (BGV) homomorphic encryption scheme [43], along with many optimizations, to make homomorphic evaluation runs faster.

4.6 Homomorphic Encryption Scheme over Integers

Although interesting from a theoretical standpoint, the lattice-based construction is difficult to describe. But its integer-based version is easy to implement. That is, we can embed an ideal into an integer ring, and if the parameters are set correctly, the scheme can be considered secure (against known attacks).

4.6.1 Secret Key Somewhat Homomorphic Encryption

Let us begin with the description of the secret key integer-based somewhat homomorphic encryption scheme [9]. The scheme is surprisingly simple, and we can construct very complex functionality from it.

Key Generation KeyGen: The secret key is an odd integer, chosen from some interval $p \in [2^{\eta-1} - 2^{\eta}]$.

Encryption **Encrypt(pk,m)**: To encrypt a bit $m \in \{0,1\}$, set the cipher text as an integer whose residue mod p, has the same parity as the plain text. Namely, set

$$C = pq + 2r + m$$

Where, the integers q, r are chosen at random in some other prescribed intervals, such as 2r is smaller than p/2 in absolute value.

Decryption **Decrypt(p,c)**: Given a cipher text c and the secret key p, output

$$M = (c \pmod p) \pmod 2$$

The decryption equation holds because

$$(c \pmod p) \pmod 2 = (pq + 2r + m \pmod p) \pmod 2$$

$$= 2r + m \pmod 2 \quad = m$$

Fully Homomorphic Property: Given two cipher text $c_1 = pq_1 + 2r_1 + m_1$ and $c_2 = pq_2 + 2r_2 + m_2$, we have

$$c_1 + c_2 = pq_1 + 2r_1 + m_1 + pq_2 + 2r_2 + m_2$$

$$= p(q_1 + q_2) + 2(r_1 + r_2) + m_1 + m_2$$

$$c_1 * c_2 = (pq_1 + 2r_1 + m_1) * (pq_2 + 2r_2 + m_2)$$

$$= (pq_1q_2 + 2q_1r_2 + 2q_2r_1 + m_1q_2 + m_2q_1)p + 2(2r_1r_2 + m_1r_2 + m_2r_1) + m_1m_2$$

when

$$r1 + r2 < p/2 \quad \ldots\ldots\ldots\ldots\ldots\ldots (4.1)$$

$$2r_1r_2 + m_1r_2 + m_2r_1 < p/2 \quad \ldots\ldots\ldots\ldots\ldots\ldots (4.2)$$

we have

$$(c_1 + c_2 \text{ (mod p)}) \text{ (mod 2)} = m_1 + m_2$$

$$(c_1 * c_2 \text{ (mod p)}) \text{ (mod 2)} = m_1 * m_2$$

Therefore, this scheme has the fully homomorphic property.

However, when we use the fully homomorphic property to evaluate a Boolean function $f(x_1,x_2,..,x_n)$ where x_i ε {0,1}, given ci , the encryption of x_i , for i = 1,2, … , n, it is noticed in Eqs. (4.1) and (4.2) that

$$r_1 + r_2 \geq \max(r_1,r_2)$$

$$2r_1r_2 + m_1r_2 + m_2r_1 \geq \max(r_1,r_2)$$

That is, the size of the noise component r in the resulting cipher text is increasing with the number of the additions and multiplications in the Boolean function. Once

$$r1 + r2 > p/2 \text{ and}$$

$$2r_1r_2 + m_1r_2 + m_2r_1 > p/2,$$

the decryption of $f(c_1,c_2,..,c_n)$ may not be $f(x_1,x_2,..,x_n)$. Therefore, this scheme can be only used to evaluate low-degree Boolean functions over encrypted data. This is why this scheme is called the somewhat homomorphic encryption scheme.

If we choose $r \approx 2^n$, $p \approx 2^{n2}$, and $q \approx 2^{n5}$, the somewhat encryption scheme can compute polynomials of degree ≈ n, before the noise grows too large.

Security: The security of this scheme can be reduced to the hardness of the approximate integer greatest common divisor (approximate GCD) problem [44]. As an example, we explain this in the more specific and familiar case of greatest common divisors. If we are given two integers a and b, we can clearly find their GCD d, say, in polynomial time. If d is in some sense large, then it may be possible to incur some additive error on either of the inputs a and b, or both, and still recover this GCD. This is what we refer to as an approximate common divisor problem. Of course if there is too much error incurred on the inputs, the algorithm may well not be able to discern the GCD d we had initially, over some other approximate divisors d (e.g., they may all leave residues of similar magnitude when dividing a and b). In this sense, the problem is similar to those found in error correcting codes.

Continuing this error correcting code analogy, we can state the problem from the standpoint of the design of the decoding algorithm, i.e., we wish to create an algorithm which is given two inputs a0 and b0 and bounds X, Y, and M for which one is assured that $d|(a_0 + x_0)$ and $d|(b_0 + y_0)$ for some $d>M$ and x_0, y_0 satisfying $|x_0| \leq X$, $|y_0| \leq Y$. The output of the algorithm should be the common divisor d, or all of the possible ones, if more than one exist.

Howgrave-Graham analyzed the (approximate GCD) problem in [44]. The problem is believed to be a hard problem in lattice theory. With a judicious choice of parameters, the secret key somewhat homomorphic encryption scheme is even more secure.

4.6.2 Public Key Somewhat Homomorphic Encryption

The secret key somewhat homomorphic encryption needs the secret key p to encrypt a message. Now we describe a public-key somewhat homomorphic encryption scheme [9] that allows encryption without the knowledge of the secret p.

Parameters: The scheme has many parameters, controlling the number of integers in the public key and the bit-length of the various integers. Specifically, we use the following four parameters (all polynomial in the security parameter λ):

γ, which is the bit-length of the integers in the public key

η, which is the bit-length of the secret key (which is the hidden approximate GCD of all the public-key integers).

ρ, which is the bit-length of the noise (i.e., the distance between the public-key elements and the nearest multiples of the secret key).

τ, which is the number of integers in the public key.

These parameters must be set under some constraints [9]. A convenient parameter set to keep in mind is $\rho = \lambda$, $\rho' = 2\lambda$, $\eta = O(\lambda^2)$, $\eta = O(\lambda^5)$ and $\tau = \gamma + \lambda$. The setting results in a scheme with complexity $O(\lambda^{10})$.

Key Generation KeyGen(λ): Choose a random η-bit odd integer p as the private key. Using the private key, generate the public key as

$$x_i = pq_i + r_i$$

Where, $q_i \; \varepsilon \; \mathbb{Z} \cap [0,2^\gamma/p)$ and $r_i \; \varepsilon \; \mathbb{Z} \cap (-2^\rho,2^\rho)$ e chosen randomly, for i = 0, 1, …, τ. Relabel so that x_0 is the largest. Restart unless x_0 is odd and x_0(mod p) is even. The public key is

$$pk = <x_0, x_1, \ldots., x_\tau>$$

Encryption **Encrypt(pk, m)**: Given m ε {0,1} and the public key pk, choose a random subset $S \subseteq \{1,2,..., \tau\}$ and a random integer r ε $(2^{-\rho'},2^{\rho'})$, and output

$$c = (m + 2r + 2\sum_{i \varepsilon S} x_i) \pmod{x_0}$$

Decryption **Decrypt(sk,c)**: Given the cipher text c and the private key p, output

$$m = (c \pmod p) \bmod 2$$

Here, c(mod p) = c – p.[c/p] where '[]' denotes rounding to the nearest integer.

Security: Like the secret key homomorphic encryption scheme, the security of the public-key somewhat homomorphic encryption scheme is also based on the approximate-GCD problem. Consider the approximate-GCD instance $\{x_0, x_1, \ldots, x_t\}$, where $x_i = pq_i + r_i$. Known attacks on the approximate-GCD problem for two numbers include brute forcing the reminders, continued fractions, and Howgrave-Graham's approximate GCD algorithm [44].

A simple brute-force attack is to try to guess r_1 and r_2 and verify the guess with a GCD computation. Specifically, for $r_1', r_2' \varepsilon (2^{-\rho'}, 2^{\rho'})$, set

$$x_1' = x_1 - r_1',\ x_2' = x_2 - r_2',\ p' = \mathrm{GCD}(x_1', x_2').$$

If p' has η bits, then output p' is a possible solution. The solution p will definitely be found by this technique, and for the parameter choices, where ρ is much smaller than η, the solution is likely to be unique. The running time of the attack is approximately $2^{2\rho}$.

Attacks for arbitrarily large values of t include lattice-based algorithms for simultaneous Diophantine approximate [45], Nguyen and Stern's orthogonal lattice [47], and extensions of Coppersmith's method to multivariate polynomials [46].

4.7 Fully Homomorphic Encryption Scheme

In this section, we describe the construction of a fully homomorphic encryption scheme given by van Dijk [9]. It is built on the somewhat homomorphic encryption scheme described in the last section and squashing the decryption circuit.

4.7.1 Squashed Encryption

Let κ, θ, Ω be three more parameters which are functions of λ. We set $\kappa = \gamma\, \eta / \rho'$, $\theta = \lambda$, and $\Omega = \omega(\kappa.\log \lambda)$. For a secret key $sk^* = p$ and public key pk^* from the original somewhat homomorphic encryption scheme, we add to the public key, a set $y=\{ y_1, y_2, \ldots y_\Omega \}$ of rational numbers in [0,2) with κ bits of precision, such that there is a sparse subset $S \subset \{1,2,\ldots, \Omega\}$ of size θ with

$$\sum_{i \varepsilon S} y_i \approx 1/p (\text{mod } 2)$$

Now the secret key is replaced by the indicator vector of the subset S. The encryption scheme is modified by van Dijk [9] as follows.

Key Generation KeyGen(λ): Generate sk* = p and pk* as before. Set $x_p = [2^k/p]$, choose a random Ω bit vector $(s_1, s_2, \dots, s_\Omega)$ with hamming weight θ and let $S = \{i : s_i = 1\}$.

Choose at random ui $\varepsilon\ \mathbb{Z} \cap [0, 2^{\kappa+1})$, $i = 1,2, \dots , \Omega$ subject to the condition that

$$\sum_{i \varepsilon S} u_i = x_p \ (\text{mod } 2^{\kappa+1})$$

Set $y_i = u_i/2\kappa$ and $y = \{y_1, y_2, \dots y_\Omega\}$. Hence, each y_i is a positive number smaller than 2, with κ bits of precision after the binary point. Also we have

$$\sum_{i \varepsilon S} y_i \ (\text{mod } 2) = (1/p) - \Delta_p$$

For some $|\Delta_p| < 2^{-\kappa}$ because

$$\begin{aligned}\sum_{i \varepsilon S} y_i &= \sum_{i \varepsilon S} u_i/2^\kappa \\ &= (x_p + \alpha\, 2^{\kappa+1}) / 2^\kappa \\ &= x_p/ 2^\kappa + \alpha.2 \\ &= [2^\kappa / p]/ 2^\kappa + \alpha.2 \\ &= (1/p - \Delta/2^\kappa) + \alpha.2 \\ &= 1/p - \Delta_p (\text{mod } 2)\end{aligned}$$

Where, $\Delta < 1$

Output the secret key sk = S and the public key {pk, y}.

Encryption Encrypt(pk,c*): Given a cipher text c^*, for $i \varepsilon \{1,2,\dots, \Omega\}$,set

$$z_i = c^* .y_i \ (\text{mod } 2)$$

Keeping only $n = [\log \theta]+3$ bits of precision after the binary point for each z_i.

Output both c* and z = $\{z_1,z_2,\ldots z_\Omega\}$

Decryption Decrypt(sk,c*,z): Given the ciphertext c*, z and the private key p, output

$$m = (c^* - [\textstyle\sum_{i \varepsilon S} z_i])(\text{mod } 2)$$

4.7.2 Bootstrappable Encryption

It constructs homomorphic encryption for circuits of any depth from the ‘somewhat homomorphic encryption’, which is capable of evaluating just a little more than its own decryption circuit.

Augmented Decryption Circuit: Let £ be an encryption scheme, where decryption is implemented by a circuit that depends only on the security parameter. For a given value of the security parameter λ, the set of augmented decryption circuits consist of two circuits, both of which take a secret key and two cipher texts as input.

- The circuit decrypts both cipher text and adds the resulting plain text bits mod 2.
- The circuit decrypts both cipher text and multiplies the resulting plain text bits mod 2.

Bootstrappable Encryption: Let £ be a homomorphic encryption scheme. We say that £ is bootstrappable, if its augmented decryption circuits are permitted circuits for every value of the security parameter λ.

The squashed encryption scheme is bootstrappable. During evaluation, every time we have a cipher text that grows beyond 2^γ , we reduce its first modulo x_γ’, then modulo $x_{\gamma-1}$’and so on all the way down to x_0’, at which point, we again have a cipher text of bit-length no more than γ.

Recall that a single operation at most, doubles the bit-length of the cipher text. Hence, after any one operation, the cipher text cannot be larger than 2 x_γ’ and therefore the sequence of modular reductions involves only small multiples of the x_i’ , which means that, it only adds a small amount of noise. It is not clear to what extent adding these larger integers to the public key influences the security of the scheme. Fully homomorphic encryption (FHE) allows a worker to perform implicit additions and multiplications on plain text values, while exclusively manipulating encrypted data. The fully homomorphic scheme proceeds in several steps. First, one constructs a somewhat homomorphic encryption scheme, which only supports a limited number of multiplications: cipher texts contain some noise that becomes larger with successive homomorphic multiplications, and only cipher texts whose noise size remains below a certain threshold can be decrypted correctly. The second step is to squash the decryption procedure associated with an arbitrary cipher text, so that it can be expressed as a low-degree polynomial in the secret key bits. Then the key idea called bootstrapping evolves, homomorphically evaluating this decryption polynomial on encryptions of the secret key bits, resulting in a different cipher text associated with the same plain text, but with possibly reduced noise. This refreshed cipher text can then be used in subsequent homomorphic operations. By repeatedly refreshing cipher texts, the number of homomorphic operations become unlimited, resulting in a fully homomorphic encryption scheme.

Summary

This chapter discussed the basics of Homomorphic encryption, fully homomorphic encryption with symmetric and asymmetric keys, along with well known techniques currently used by researchers to solve the problem of encrypted data searching. Homomorphic encryption schemes are extensively used in Electronic voting, private searching, etc. The primary open problem is to improve the efficiency of the existing schemes based on the application context, to the extent that it is possible while preserving the hardness of the approximate-GCD problem.

Chapter 5

Modified Homomorphic Encryption (MHE) Scheme

"There's a lot of engineering work to be done. But until now we've thought this might not be possible. Now we know it is."

- Ronald Linn Rivest.

Modified Homomorphic Encryption (MHE) scheme is a fully homomorphic encryption algorithm that enables operations to be applied on a cipher text and obtain the result without decrypting the data.

5.1 Introduction

Homomorphic Encryption scheme enables operations to be applied on an encrypted domain with results obtaining similar to operations applied on raw data. The advantage is that, there is no need of decrypting the operands to perform the operation and hence the operation can be applied on a third party system without revealing any sensitive information to them. However, original FHE scheme proposed by Gentry employs ideal lattices over polynomial ring and it is proven to be too expensive for practical applications. Gentry's FHE scheme can be modified to suit each application scenario thus reducing the implementation cost. This chapter discusses the Modified Homomorphic Encryption scheme that my research proposed to suit the ranked information retrieval scenario.

5.2 Need for modifying Gentry's FHE Scheme

According to Gentry's public key FHE scheme [12], the encryption and decryption formulae are $c = pq + 2r + 2\sum x_i$ and $m = (c \bmod p) \bmod 2$, which can be applied on messages, $m= \{0,1\}$. Since the algorithm takes only 0 and 1 as inputs, every number is converted to binary before it is applied to the operation, and the encryption and decryption algorithm is applied over each bit. This solution is ideal when the value of the message is unpredictable. But, if the value of the message to be secured is predefined to be within a range, then the space and time complexity of the algorithm can be improved by redefining the algorithm, thus eliminating the need for conversion to binary and encrypting bit by bit.

For the ranked information retrieval scenario, my research utilizes vector space model where each cell is filled with TF-IDF values. Hence, here only multiplication and summation operation is needed to be applied on TF-IDF to find the similarity score of query to documents. Before encrypting the TF-IDF, normalization is applied on the values which always ensure that the resulting values range between 0 and 10. Thus the proposed MHE scheme needs to handle only integer multiplication and summation operation to support the encrypted similarity calculation.

5.3 Modified Homomorphic Encryption (MHE) Scheme

Let m and c denotes the plaintext and cipher text message respectively.

Gentry's FHE scheme can be defined as follows:

Encryption: $c = pq + 2r + m$

Decryption: $m = (c \bmod p) \bmod 2$

Here, p denotes the secret key, q denotes the multiplying parameter and r denotes the noise to protect from brute force attack. '$pq+r$' serves as the public key.

This encryption scheme encrypts each bit to $||p|| + ||q||$ bits where the notation $||x||$ refers to bit length of x. If this scheme is applied as such to encrypt the TF-IDF values, then each bit will be transformed to very large values which make the storage, transfers and computations costly, especially when the index to be encrypted is huge.

To reduce the communication overhead, the FHE scheme is modified to reduce the resultant cipher text size. The Modified Homomorphic Encryption Scheme (MHE) is defined as:

Symmetric Encryption: $c = pq + sr + m$, where $s = 2^{2||m||}$, $p >> r$ and $r >> s$ ensures correct decryption. As the size of the cipher text doubles after multiplication, select the noise parameter 'r' greater than or at least equal to $2^{2||m||}$.

Asymmetric Encryption: $c = pq + sr + s\sum x_i$, where s depends on the maximum value possible for message, m.

For both the schemes, decryption becomes $m = (c \bmod p) \bmod s$

5.4 Advantage of the MHE Scheme

Apart from eliminating the need for separate bit by bit encryption, this scheme reduces the number of bits in cipher text by approximately $1/||m||$, where $||m||$ implies the number of bits in the maximum possible value of m. In Gentry's scheme, the number of bits will be nearly $n.(||p||+||q||)$ bits (n = number of bits in message) whereas in ours, the number of bits will remain approximately $||p||+||q||$. For example, if the score has a value of 2^{100}, then size of the cipher text will be $100*(||p||+||q||)$ for Gentry's FHE scheme whereas it will be simply $(||p||+||q||)$ for the modified MHE.

5.4.1 Formal definition of Modified Homomorphic Encryption (MHE)

Let m ranges from 1 to n, then set $s = 2^{2||n||}$. The Private key, p and noise parameter, r are chosen such that, $p >> r >> s$. To be specific, choose private key, $p >> s . \sum r_i$ + noise and $p >> s^2. \prod^2_1 (\sum r_i)$ to preserve the correctness of decryption. Public Key, $PK = <pk_0, pk_1 \ldots pk_\delta>$ is a set of δ elements where pk_0 is the largest. Each element, $pk_i = pq_i + r_i$ where $q_i \varepsilon Z \cap [0, 2^\gamma/\rho)$ and $r_i \varepsilon Z \cap [2^{-\rho}, 2^\rho]$ are chosen randomly for $i=0,1,\ldots \delta$. Here, $\gamma = O(\lambda^5)$, $\rho = \lambda$ and $\delta = \lambda+\gamma$, for a security parameter λ. Refer Section 5.4.3 to see the numeric examples of MHE scheme.

Encryption: Encrypt(PK, m):

Given $m \varepsilon Z_n$ and the public key PK, choose a random subset $X \subseteq \{1, 2, .. \delta\}$ and a random integer, r and output $\mathbf{c = m + s.r + s.\sum_{i \varepsilon X} pk_i}$.

Decryption: Decrypt(SK, c):

Given the private key/secret key, ‘p’ and the cipher text, ‘c’ then output is, **m = (c mod p) mod s.**

5.4.2 Proof of Correctness for MHE Algorithm

i. Proof of correctness for decryption

$\mathbf{c_1} = m_1 + sr + s.\sum_{i\varepsilon X} pk_i$

$= m_1 + sr + s.\sum_{i\varepsilon X} (pq_i+r_i)$

$= m_1 + sr + s\,p.\sum_{i\varepsilon X} (q_i) + s.\sum_{i\varepsilon X} (r_i)$

$\mathbf{m_1} = (c_1 \bmod p) \bmod s$

$= ((m_1 + sr + s\,p.\sum_{i\varepsilon X} (q_i) + s.\sum_{i\varepsilon X} (r_i)) \bmod p) \bmod s$

$= (m_1 + sr + s.\sum_{i\varepsilon X} (r_i)) \bmod s$ since $s.(r + \sum_{i\varepsilon S} (r_i)) < p$

$= m_1$

ii. Proof of correctness for homomorphic addition

$\mathbf{c_1 + c_2} = (m_1 + sr_1 + s\,p.\sum_{i\varepsilon X} (q_i) + s.\sum_{i\varepsilon X} (r_i)) \mathbf{+} (m_2 + sr_2 + s\,p.\sum_{i\varepsilon X} (q_i) + s.\sum_{i\varepsilon X} (r_i))$

$= m_1 + m_2 + s(r_1 + z_2) + sp(.\sum_{i\varepsilon x} (q_i) + \sum_{j\varepsilon x} (q_j)) + s.(\sum_{i\varepsilon S} (r_i) + \sum_{j\varepsilon x} (r_j))$

$\mathbf{m_1 + m_2} = ((c_1 + c_2) \bmod p) \bmod s$

$= ((m_1 + m_2 + s(r_1 + z_2) + sp(.\sum_{i\varepsilon x} (q_i) + \sum_{j\varepsilon x} (q_j)) + s.(\sum_{i\varepsilon S} (r_i) + \sum_{j\varepsilon x} (r_j))))\bmod p) \bmod s$, since $s.(noise + \sum_{i\varepsilon x} (r_i)) < p$

$= (m_1 + m_2 + s(r_1 + r_2) + x.(\sum_{i\varepsilon S} (r_i) + \sum_{j\varepsilon S} (r_j)) \bmod s$

$= m_1 + m_2$

iii. Proof of correctness for homomorphic multiplication

$\mathbf{c_1 * c_2} = (m_1 + sr_1 + s\,p.\sum_{i\varepsilon X} (q_i) + s.\sum_{i\varepsilon X} (r_i)) \mathbf{*} (m_2 + sr_2 + s\,p.\sum_{i\varepsilon X} (q_i) + s.\sum_{i\varepsilon X} (r_i))$

= $(m_1 ((m_2 + sr_2 + sp .\sum_{j\varepsilon x} (q_j)+s.\sum_{j\varepsilon S} (r_j)) + sr_1 ((m_2 + sr_2 + sp .\sum_{j\varepsilon x} (q_j)+x.\sum_{j\varepsilon x} (r_j)))+sp.\sum_{i\varepsilon x} (q_i). (m_2 + sr_2 + sp.\sum_{j\varepsilon x} (q_j)+s.\sum_{j\varepsilon x} (r_j) + s.\sum_{i\varepsilon x} (r_i) ((m_2 + sr_2 + sp .\sum_{j\varepsilon x} (q_j)+s.\sum_{j\varepsilon x} (r_j)))$

$\mathbf{m_1 * m_2}$ = $((c_1 * c_2) \bmod p) \bmod s$

= $(m_1 ((m_2 + sr_2 + sp .\sum_{j\varepsilon x} (q_j)+s.\sum_{j\varepsilon S} (r_j)) + sr_1 ((m_2 + sr_2 + sp .\sum_{j\varepsilon x} (q_j)+x.\sum_{j\varepsilon x} (r_j)))+ sp.\sum_{i\varepsilon x} (q_i). (m_2 + sr_2 + sp.\sum_{j\varepsilon x} (q_j)+s.\sum_{j\varepsilon x} (r_j) + s.\sum_{i\varepsilon x} (r_i) ((m_2 + sr_2 + sp .\sum_{j\varepsilon x} (q_j)+s.\sum_{j\varepsilon x} (r_j))) \bmod p) \bmod s$ (since $p > s^2. \prod^2{}_1 (\sum r_i)$)

= $((m_1 ((m_2 + sr_2 + s.\sum_{j\varepsilon S} (r_j)) + sz_1 ((m_2 + sr_2 + s.\sum_{j\varepsilon S} (r_j))) + s.\sum_{i\varepsilon x} (r_i) ((m_2 + sr_2 + s.\sum_{j\varepsilon x} (r_j)))) \bmod s$

= m_1*m_2

5.4.3 Numerical Example for MHE

Let the secret key be p = 19760800000 (P>>r>>x) . Based on p, we construct the public key as follows.

$x= 2^{2.4} = 2^8 = 256$

Q = [q0,q1,q2,q3] = [36,27,34,6]

R = [r0,r1,r2,r3] = [8,5,4,2]

$Xi = pq_i + r_i$

X0 = 19760800000 * 36 + 8 = 711388800008

X1 = 19760800000 * 27 + 5 = 533541600005

X2 = 19760800000 * 34 + 4 = 671867200004

X3 = 19760800000 * 6 + 2 = 118564800002

X=[x0,x1,x2,x3] = [711388800008, 533541600005, 671867200004, 118564800002]

Let M1=5 r1=500 I = [1,3]

C1 = 5+256*(500)+256*(533541600005 + 118564800002)

= 5+256*(500)+256(652106400007) = 5+ 128000 + 166939238401792

= 166939238529797

M1 = 166939238529797 % 19760800000 % 256 =5

As expected, the cipher text is decrypted correctly.

Another example:

M2=11 r2=600 i=[0,1]

C2 = 11 + 256*(600)+256*(1244930400013) = 11+ 153600+ 318702182403328

= 318702182556939

M2 = 156939 %256 = 11

C1*c2 = 53203899673839748484600611383 % 19760800000 % 256 = 55 (=11*5)

C1 + C2 = 485641421086736 % 19760800000 = 286736 % 256 = 16 (=11+5)

Proves, the algorithm satisfies homomorphic addition and multiplication.

Let M3 = 12 r_m = 550 i= [3,4]

C3 = 12 + 140800 + 256*790432000006 = 202350592001536 + 140812 = 202350592142348

Above example shows that MHE is not order preserving. Here, E(11) > E(12)

M3 = 202350592142348 % 19760800000 % 256 = 12

C2 * C3 = 64489575357455298640503152772 %19760800000 = 2579152772 %256 = 132

C2 +C3 = 521052774699287 % 19760800000 % 256 = 23

5.5 Security of the MHE scheme used for securing the index

The proposed MHE scheme is secure and can be explained based on the approximate GCD problem. Consider the approximate-GCD instance $\{x_0, x_1, \ldots x_t\}$ where $x_i = pq_i + r_i$. Known attacks on the approximate-GCD problem for two numbers include brute forcing the reminders, continued fractions, and Howgrave-Graham's approximate-GCD algorithm [14]. Given two integers, we can compute their greatest common divisor efficiently using Euclid's algorithm. Howgrave-Graham [28] formulated and gave an algorithm to solve an approximate version of this problem, asking the question "What if instead of exact multiples of some common divisor, we only know approximations?" In the simplest case, we are given one exact multiple $N = pq_0$ and one near multiple $a_1 = pq_1 + r_1$, and the goal is to learn p, or at least p gcd(q0, q1). Our scheme is based on the hardness of approximate integer common divisors problems introduced by Howgrave-Graham. In the general version of this problem (approximate GCD), the goal is to recover a secret number p (typically a large prime number), given polynomially many near-multiples $x_0, \ldots, x_m$ of p, that is, each integer x_i is of the hidden form $x_i = pq_i + r_i$ where each q_i is a very large integer and each r_i is a very small integer.

A simple brute-force attack is to try to guess r_1 and r_2 and verify the guess with a GCD computation. Specifically, for r_1', r_2' ε $(2^{-\rho}, 2^{\rho})$, set $x_1' = x_1 - r_1'$, $x_2' = x_2 - r_2'$, $p' = GCD(x_1', x_2')$.

If p' has η bits, output p' as a possible solution. The solution p will definitely be found by this technique, and for the parameter choices, where ρ is much smaller than η, the solution is likely to be unique. The running time of the attack is approximately $2^{2\rho}$. For terms, t>2 the complexity still increases to $O(t^3 2^{2\rho})$ which is too much impractical to implement real time.

Attacks for arbitrarily large values of t include lattice-based algorithms for simultaneous Diophantine approximate [15], Nguyen and Stern's orthogonal lattice [16], and extensions of Coppersmith's method to multivariate polynomials [17].

In the continued fractions attack, a sequence of integer pairs is obtained (y_i, z_i) such that $|(x_1/x_2)-(y_i/z_i)| < 1/z_i^2$. Since q_1/q_2 is a good approximation of x_1/x_2, i.e.,$(x_1/x_2 - q_1/q_2) \approx 0$

, (q_1,q_2) probably occurs in the sequence. If so, p can be recovered by p = [x_1/q_1]. The ($x_1/x_2 - q_1/q_2$) in our scheme, however, is not small enough to be recovered by this attack. Specifically, q^{22}

$$\frac{x1}{x2} - \frac{q1}{q2} = \frac{(q2.r1 - q1.r2)}{q2(pq2 + r2)} \approx \frac{q2.r1 - q1.r2}{p} . \frac{1}{q2.q2}$$

Since, $\frac{q2.r1 - q1.r2}{p} >> 1$, according to the parameter selection in our scheme, the pair (q1,q2) cannot be obtained. Therefore, the continued fraction attack cannot break into our system.

Howgrave-Graham gives a lattice attack on the multi-element approximate-gcd problem. In this attack, when t = 2, the relevant lattice may contain exponential vectors unrelated to the approximate-gcd solution, so that lattice reduction turns out to be in vain. For arbitrary t > 2, the time needed to guarantee a 2^{η} approximation is roughly $(2^{\eta})^{\gamma/2}$, resulting the overall computing complexity is $\Omega(2^{\lambda})$, which is difficult to crack. In conclusion, the MHE scheme guarantees sufficient security.

5.6 Implementation of MHE

HElib [17] is an open-source library which implements the homomorphic encryption scheme with some optimizations such as cipher text packing techniques (SIMD) [28] and optimizations in [14]. There are many useful functions in the library besides the evaluation of the AND gate and the XOR gate, including some initialization functions, and some helper classes like Encrypted Array which provides us with easy encryption and manipulation to the cipher text slots. The library is written in C++ and uses the MTL Mathematical Library. The code that we used for creating a vector and perform the addition and multiplication has been included in the appendix.

There are many parameters in HElib interface, most of which are used to compute the integer m. The library provides a function FindM() which can determine a proper m according to the input parameters. In our experiments, we set the security level λ = 80 (that implies the breaking time of the encryption scheme is roughly 2^{80}) which is a reasonable value. We test our implementation, which is single-threaded, on a PC with a Intel Core i4 CPU at 3.60GHz and 8GB RAM.

5.7 Results and Discussion

To compare the performance of MHE with FHE we randomly generated some vectors of mxn dimension and multiplied it with a 1xn array of values. (The test is designed to match our application scenario, because in our application multiplication and summation is needed to calculate the similarity score.) Execution time incurred to perform the matrix multiplication is as shown below. For this experiment, we set m=257.

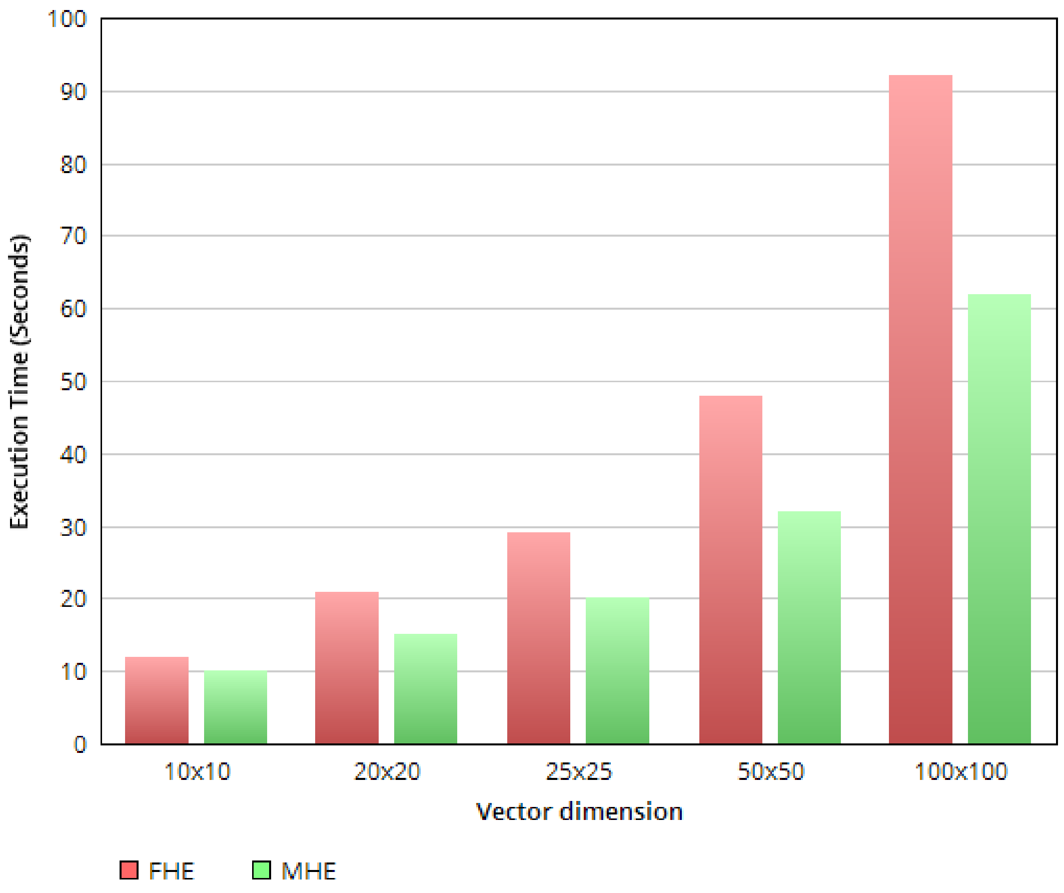


Figure 5.1: Comparison of Execution time

The execution time needed for addition and multiplication for the mentioned parameters is given below. For smaller values of ‘m’ the Modified Homomorphic Encryption shows better execution performance than FHE. Hence, in our application scenario we will normalize the TF-IDF such that it always falls within a smaller interval. For this analysis we considered a 100x100 matrix.

Apart from the execution time, the MHE scheme outperforms the FHE in terms of cipher text space consumed.

For different vector dimensions and maximum value m of 257, the cipher text space consumed is plotted below. The space consumed is less for MHE as the operation is not done on a bit by bit. Instead the numeral is considered as such for the operation which reduces the storage space.

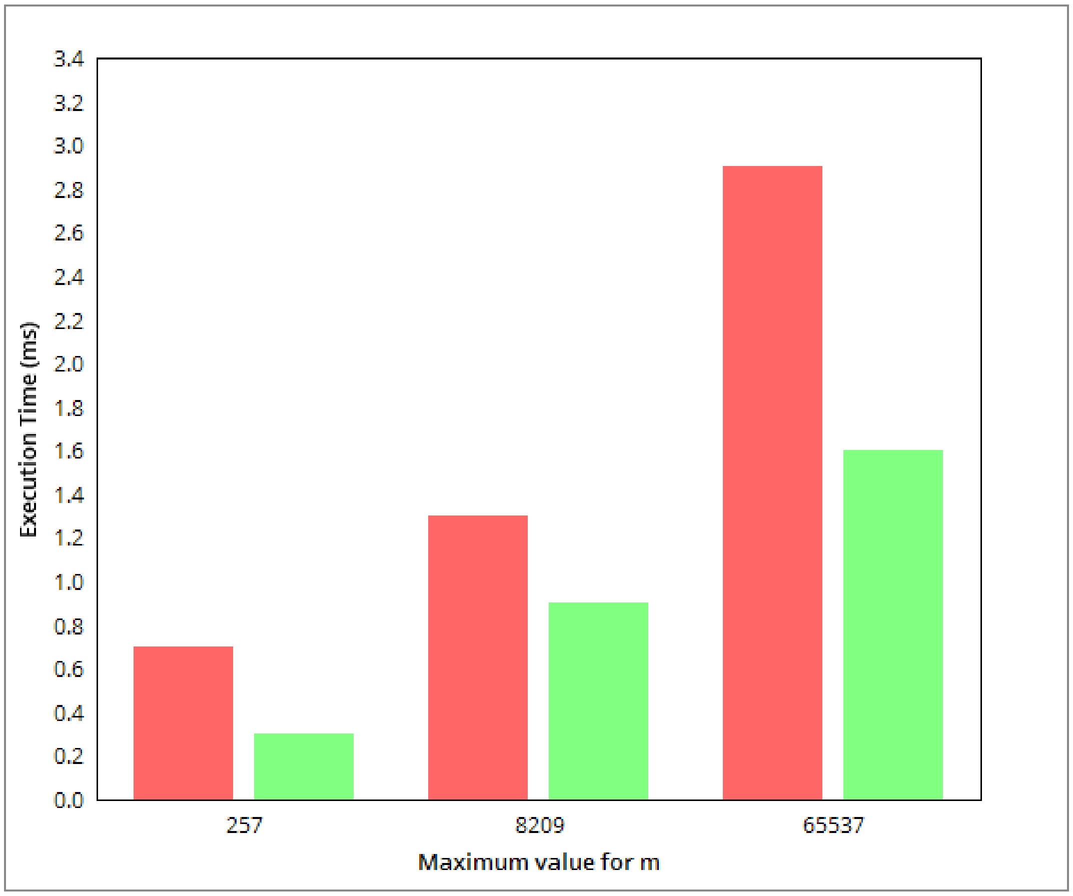


Figure 5.2: Execution Time for Addition

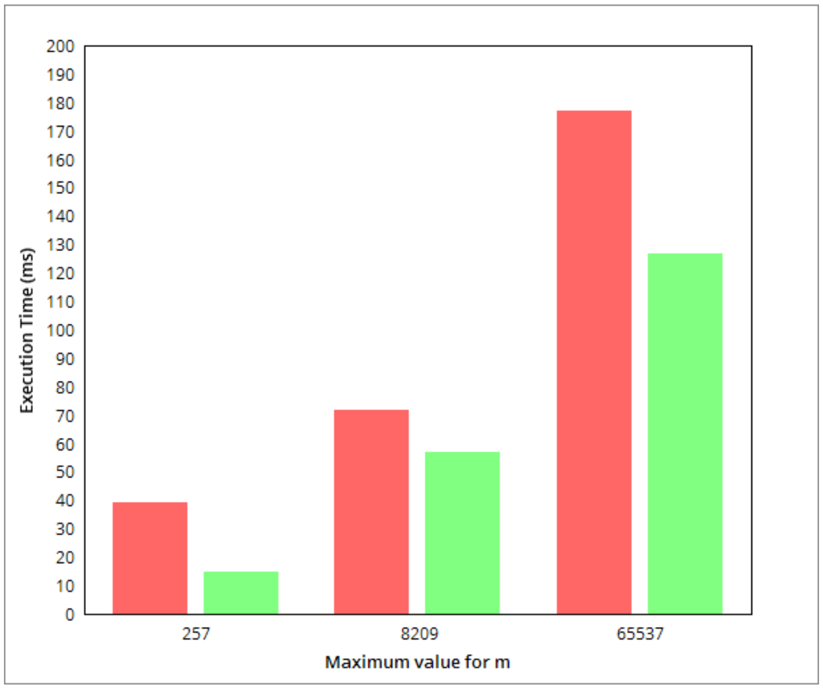


Figure 5.3: Execution Time for Multiplication

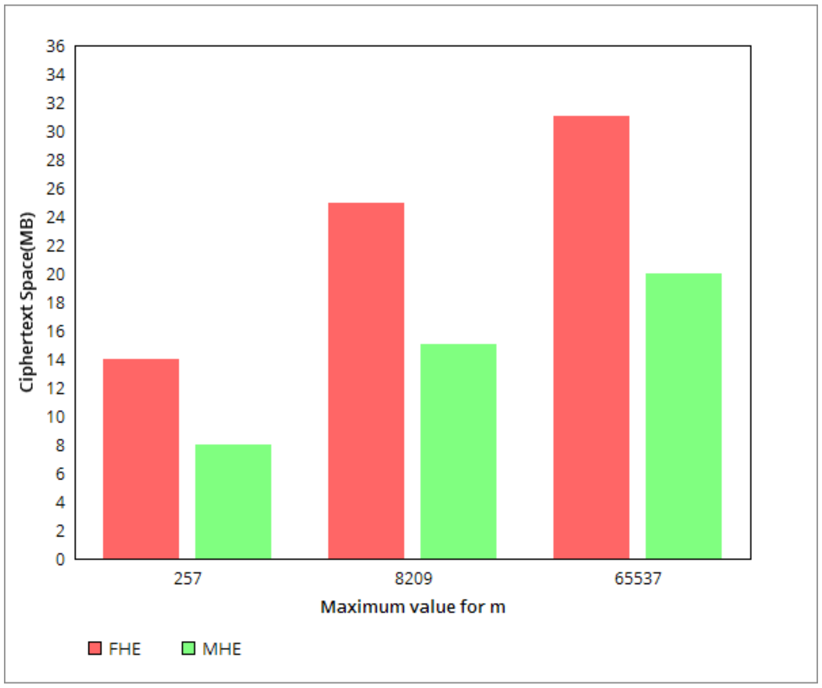


Figure 5.4: Comparison on Cipher text Space

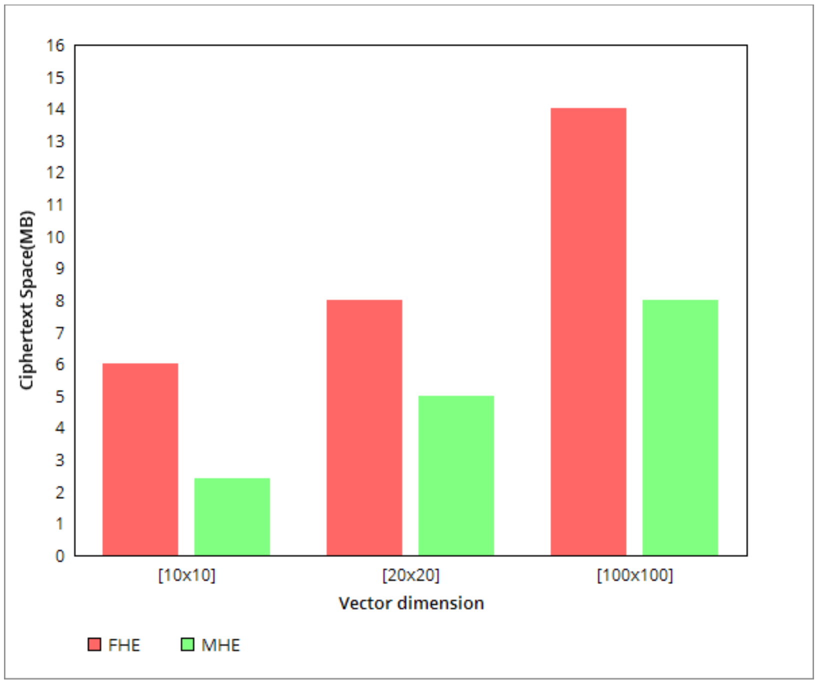


Figure 5.5: Comparison on cipher text space

Summary

This chapter discussed in detail the Modified Homomorphic Encryption proposed that makes it suitable for information retrieval based on similarity score. The proposed method is proven to be secure and it performs well compared to the base scheme especially in terms of the cipher text size consumed, which is very much significant when dealing with large amount of data. Analysis done on the scheme proves that it is secure and optimized and can be applied in privacy preserving ranked information retrieval.

CHAPTER 6

Dual Round Encrypted Information Retrieval (DRIER) Framework

"There are two types of encryption: one that will prevent your sister from reading your diary and one that will prevent your government."

- Bruce Shneier.

The DRIER framework describes how to apply the proposed Modified Homomorphic Encryption scheme for secure and privacy preserving keyword searches from outsourced encrypted data.

6.1 Introduction

This chapter describes how we can apply the MHE algorithm we proposed, for efficient ranked information retrieval from outsourced data. Well known techniques in information retrieval, like vector space and Term Frequency-Inverse Document Frequency are utilized for relevance matching. The searching scheme supports multiple keyword searches and ranking, based on query keywords. DRIER framework establishes two rounds of communication between the user and the cloud, to retrieve the matching documents, in a secure and privacy preserving way.

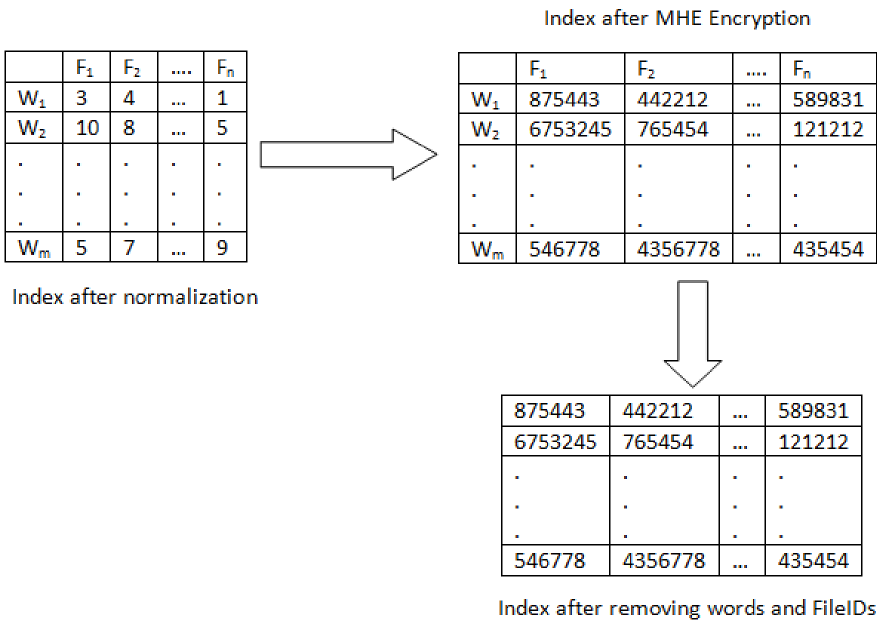
6.2 Searchable Index Generation

To store the files in a secure manner and to enable searching from the encrypted domain, we need to maintain a searchable index which is protected by encryption. Initially, we create the vector space model, which is a two dimensional representation of the words and files in the documents. Then, each cell is filled with the TF-IDF values as described in chapter 3. We apply Min-Max normalization technique to rescale the values to a new range, 1 to 10. Now, we encrypt each cell with our MHE scheme and remove the words and file IDs from the index. We maintain the list of words and list of files as keys, and distribute these keys to privileged users. Only the encrypted index is uploaded to the cloud.

Actual files are encrypted using any symmetric algorithm and stored in the cloud separately. AES is used to encrypt the actual files. Figure 6.1 illustrates the secure index generation stages.

6.3 Information Retrieval using MHE encrypted index

When a user needs to search for a particular file, he will issue a query $Q= \{q_1,q_2,\ldots q_t\}$, having some keywords q_i. If the index is present in plain text format, the score of each file will be calculated by adding the TF-IDF$_{ij}$ for all j and for all i, for which w_i is present in Q. After obtaining the sum of TF-IDF of all files, we will sort the files based on the value of scores obtained. Those files having higher scores are ranked relevant to the query.


Index after MHE Encryption
Index after normalization
Index after removing words and FileIDs

	F_1	F_2	....	F_n
W_1	3	4	...	1
W_2	10	8	...	5
. . .	. . .	. . .	. . .	. . .
W_m	5	7	...	9

	F_1	F_2	....	F_n
W_1	875443	442212	...	589831
W_2	6753245	765454	...	121212
. . .	. . .	. . .	. . .	. . .
W_m	546778	4356778	...	435454

875443	442212	...	589831
6753245	765454	...	121212
. . .	. . .	. . .	. . .
546778	4356778	...	435454

Figure 6.1 Encrypted Index Generation

Algorithm 6.1: Searchable Index Generation

Input: Files to be securely stored F = $\{F_1,F_2,\ldots F_n\}$

Output: Secure Searchable Index Generated using MHE

Steps:

1. Extract each word from the text.

2. Apply Stemming to the words.

3. Eliminate stop words like a, an, the, is etc. and form the final wordlist, W = $\{w_1,w_2,w_3,\ldots w_m\}$

4. Find TF-IDF values of each word in each file.

5. Form the vector space model W X F, with cell values filled with TF-IDF. i.e. for all values of i,j put W_iF_j = TF-IDF_{ij} where $1 \leq i \leq m$ and $1 \leq j \leq n$.

6. Apply Min-Max normalization technique to rescale the values to the range 1 to 10.

6. For all i, j replace TF-$IDF_{i,j}$ with MHE(TF-$IDF_{i,j}$)

8. Remove the words and file IDs from the Index to form the secure *mxn* index, SI.

Considering the secure index, SI is in an encrypted form and we do not have words present in the index to match with the query terms; hence we need to modify the basic information retrieval scheme. Also, the file names will not be present in the encrypted index. If we can generate a pattern, P of 0s and 1s of length m, such that 1 indicates the presence of the word, w_i in Q and 0 indicates the absence of w_i in Q, and if this pattern is generated based on the exact word order followed in the index SI, we need to just multiply each bit P_i with the TF-$IDF_{i,}$ and find the sum of scores for each file. Figure 6.2 illustrates the modified scheme for score calculation.

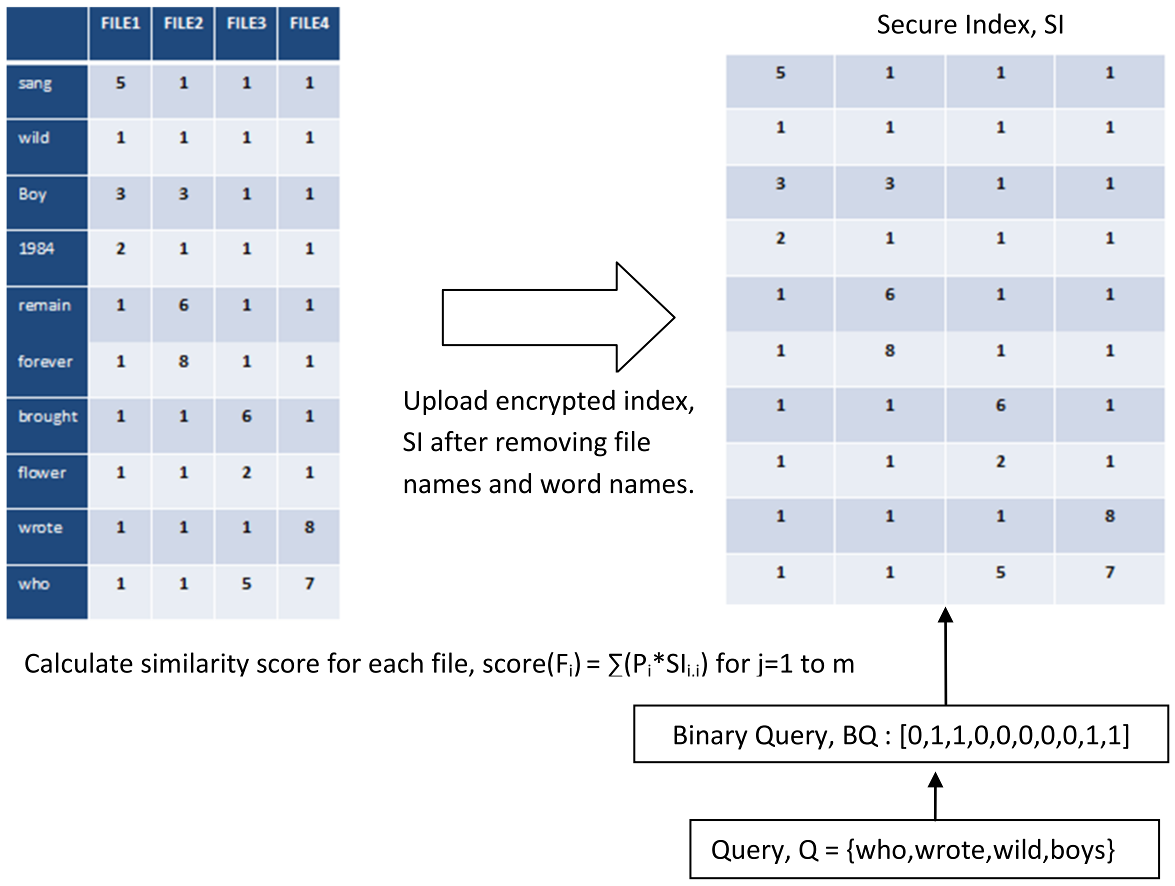
Let the query be Q= $\{q_1,q_2,\ldots q_t\}$. Generate a binary pattern BQuery, based on the presence or absence of words in the query. BQuery will be of length m. The procedure is illustrated in figure 6.2. Here, to find the similarity score, only addition and multiplication operation is needed. Hence, the idea is to encrypt the values in Secure Index and bits in binary query, with the modified encryption scheme that we proposed. After obtaining the BQuery, each bit is encrypted using the MHE algorithm to form the encrypted query, EQuery. Since our MHE scheme is fully homomorphic, the score calculated on the encrypted data will be equal to the raw data after decryption. This encrypted query, acts as the trapdoor for score calculation.

The list containing encrypted scores of files, ES, is then sent to the user. At the user side, scores are decrypted to identify the files having maximum scores. The user can select how many files are to be retrieved from the server, based on his requirement.

If the number of files is very high, the ranking of the scores S_n to obtain the top K files having highest match, can be found out by following the algorithm below.

Algorithm 6.2: Trapdoor Generation

```
Input:   Query keywords, Q={ $q_1,q_2,\ldots q_t$}
         Word Order, W = {$w_1,w_2,w_3,\ldots w_m$}
Output: Encrypted Query, EQuery
For i in 1 to m,
         $BQuery_i$ = If $w_i$ is present in Q, the value is 1 or else it is 0.
         $EQuery_i$ = MHE($BQuery_i$)
Return EQuery
```



	FILE1	FILE2	FILE3	FILE4
sang	5	1	1	1
wild	1	1	1	1
Boy	3	3	1	1
1984	2	1	1	1
remain	1	6	1	1
forever	1	8	1	1
brought	1	1	6	1
flower	1	1	2	1
wrote	1	1	1	8
who	1	1	5	7

Secure Index, SI

5	1	1	1
1	1	1	1
3	3	1	1
2	1	1	1
1	6	1	1
1	8	1	1
1	1	6	1
1	1	2	1
1	1	1	8
1	1	5	7

Upload encrypted index, SI after removing file names and word names.
Calculate similarity score for each file, score(F_i) = $\sum(P_i * SI_{i,i})$ for j=1 to m
Binary Query, BQ : [0,1,1,0,0,0,0,0,1,1]
Query, Q = {who,wrote,wild,boys}

Figure 6.2: Similarity Score Calculation from Secure Index.

Algorithm 6.3: Score Calculation with Secure Index

```
Input: Encrypted Query, EQ={ $eq_1$,$eq_2$,...$eq_t$}
        Secure Index, SI[m][n]
Output: Encrypted Scores, ES[n]
For i = 1 to n
        For j = 1 to m
                ES[i] = ES[i] + ( SI[j][i] * EQ[j] )
        End For
End For
Return ES
```

Algorithm 6.4: Top-K Similar Document Select Algorithm (S_n,K)

```
Input:
S_n : list containing scores of each file S_n=((fid_1,s_1),(fid_2,s_2),….,(fid_n,s_n))
K : number of files to be retrieved.
Output:
TopList_K = Top K-Relevant Files
    1.  Initialize: TopList_K = NULL
    2.  For each item Ɛ S_n
    3.          If length(TopList_i) < K
    4.                  Add item  to TopList in ascending order of the score
    5.          Else
    6.                  If(item['score'] > TopList_0['score'])
    7.                          Replace TopList_0 with item
    8.                          Sort first K elements in TopList_k in ascending order.
    9.                  Else
    10.                         Discard the item
    11.                 End IF
    12.         End IF
    13. End For
    14. Return TopList_k
```

Algorithm 6.4, Top-K Similar Document Select Algorithm, takes the decrypted scores of each file (S_n) as the input, and finds the top k matching files in minimal execution time. Instead of sorting the entire list which follows a complexity of $O(n^2)$, the proposed algorithm reduces the execution time to O(nk). Here, the worst case needs to sort only k elements. By, implementing max-heap concept, the complexity can still be reduced to O(logn). Also, the next chapter illustrates how we can reduce this complexity to O(k) by applying Map Reduce programming model to rank the list.

6.4 Dual Round Encrypted Information Retrieval (DRIER) framework to retrieve the similar documents

After obtaining the top K matching documents, the user will send the list containing top k file identifiers $FID_k = \{i_1, i_2, \ldots i_k\}$ to the cloud. The cloud will send the encrypted file corresponding to the FID. At the user side, the files are decrypted to retrieve the required contents. If the required files are not found, user can issue the next set of file IDs and continue the process till the required files are found. The Dual Round Information Retrieval Scheme to find the matching documents is illustrated in figure 6.3.

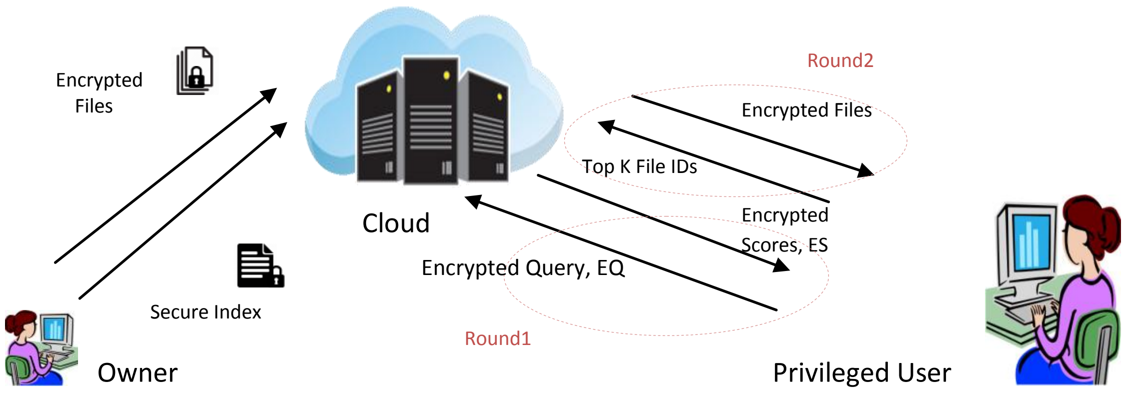


Figure 6.3 Dual Round Secure Information Retrieval

Dual Round Secure Information Retrieval Framework can be summarized as follows:

1. Setup(λ): Based on the security parameter λ, the data owner generates the public key PK, and the secret key SK. The public key PK is then delivered to authorized users.

2. IndexBuild(DocCollection,PK): Documents are arranged in vector space model after applying IR techniques like stemming and stop word elimination. The index is homomorphically encrypted to generate a secure index, SI with height m and width n, using the public key, PK. Then, SI is uploaded to cloud server along with other encrypted files. Files are encrypted using AES and index is encrypted using MHE.

3. TrapdoorGenerate(Query,PK): The query keywords obtained from user Q_t, are arranged into a Boolean Query vector form BQ_t, where $Q_j = 1$ if w_j is present in Q_t or else 0. BQ_t is then homomorphically encrypted using PK to form the trap door EQ_t. This is then sent to the cloud server.

4. ScoreCalculate(T_q,I_e): Encrypted score, 'es' of each document is calculated using equation 1. Resulting vector will be $SS_d = (es_1, es_2,…., es_d)$

5. Rank(SS_n,SK,k): Encrypted Scores are decrypted at client machine using secret key, SK and retrieve the actual scores, $S_d=((fid_1,s_1),(fid_2,s_2),….,(fid_n,s_n))$. Sort the scores to find the top *k* similar documents matching with the query.

6. Retrieve Top Matching Files: The top-K ranked document ids are sent to the cloud server and it returns the encrypted documents to the clients, which can then be decrypted by authorized users.

6.5 Security Analysis of the DRIER Framework using MHE

Security of the MHE scheme has already been analyzed in chapter 5. Here, the security of the Dual Round framework for Information Retrieval is analyzed.

6.5.1 Security against Brute Force Attack

Apart from the security of the homomorphic encryption, the proposed scheme utilizes word order and document order as the keys for correct retrieval. M words and N documents can be arranged in M! X N! ways, and as M or N increases, the complexity of brute force increases. Hence, we can say that as the data to be protected increases, the security of the proposed scheme increases.

6.5.2 Security against Statistical Data Leakage

The most common attack that happens in text encryption is the statistical data based brute forcing. For example, 'the' is the most frequently occurring word in English documents. Based on this prior information on frequency distribution of words, critical brute forcing can be initiated. For example, if we identified one word as 'human', the next possible word that follows can be 'resource'. To summarize, there exists basically two types of statistical leakages: term distribution and inter-distribution. Term distribution refers to the frequency distribution of terms within a document, and inter-distribution refers to a file's frequency distribution for each term.

Term distribution and inter-distribution are analyzed by attackers in two ways. In the direct method, prior information available on statistical distribution is utilized as such. But, such attacks may not be fully effective, as most of the frequent words are stop words, and we eliminate such words during indexing. Also, TF-IDF value taken for indexing instead of just frequency will eliminate the statistical distribution of the terms on documents.

Indirect way of attack will utilize the previous access and search pattern of users, to identify the term distribution. Order preserving encryption schemes will encrypt to similar result, for words having similar occurrences. But, since the MHE scheme does not preserve any order, the scheme encrypts the same value to different results at each encryption, because the result depends on the public key subset that we select during the encryption.

Term distribution of some frequently occurring words are analyzed and found that the proposed scheme does not preserve the frequency as such, before and after encryption. Figure 6.4 illustrates this.

Search Pattern and Access patterns are hidden from the attacker as well. Each time, the user searches for a set of keywords, the query is encrypted such that the cloud cannot assume what the user has searched for, and what is present in the file. Also, if two different queries contain the same words, different encryptions at different time will yield different encrypted results, as the encryption depends on the subset of the public key that was chosen. Formally, same keyword k, is requested in two different queries Q1 and Q2, and each of this will be encrypted to two cipher texts C1 and C2, by taking different subsets of the public key PK. Also, the index stored in the cloud does not contain the words or file names, thus preventing any guessing attacks about the content of the files.

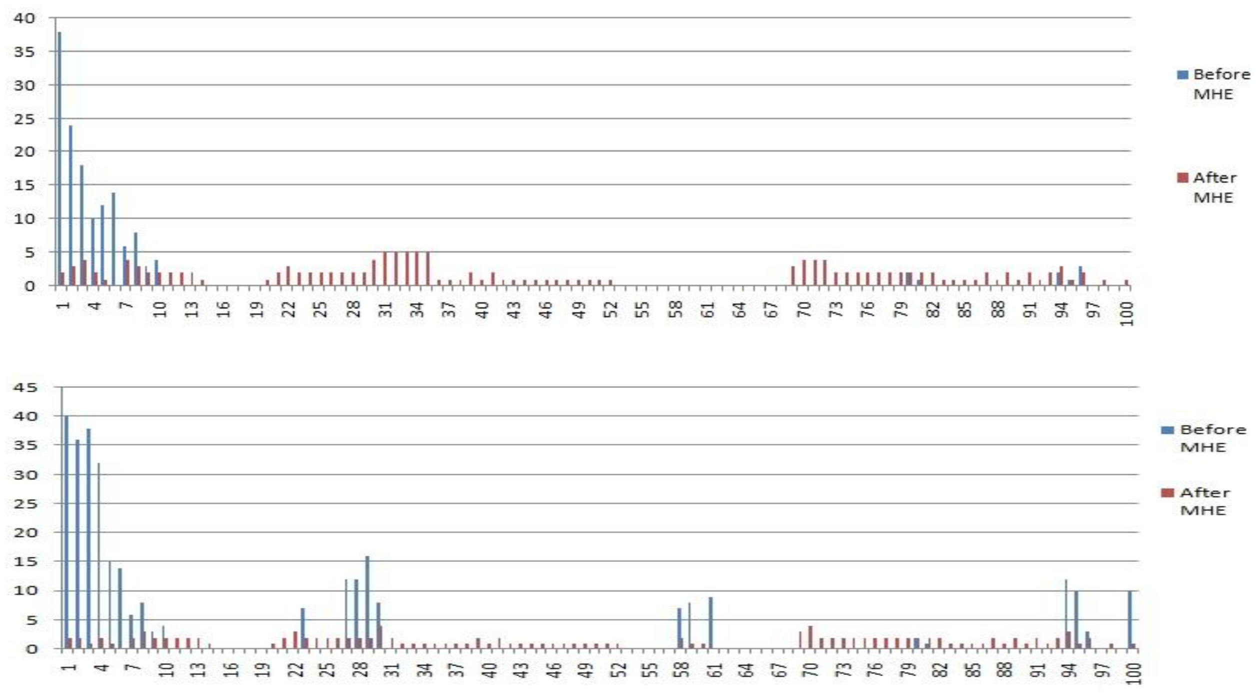


Figure 6.4: Distribution of similarity relevance of 142 terms with (a) “data”, and (b) “resources” before and after MHE in the Newsgroups data set.

6.6 Performance Analysis of the DRIER Framework

This section analyzes the performance of the proposed DRIER scheme, taking into account the execution time as well as the communication complexity.

6.6.1 Experimental Setup and Dataset used

The experiment is conducted with two machines acting as client and server. The user machine is setup on a laptop with Windows 7 installed, having Core 2 Duo CPU and speed 2 GHz. The server machine is installed on an AWS Ubuntu instance with Xeon E5620 CPU and speed 2.4 GHz. The client machine is the user’s system that creates the index, encrypts and uploads the files. The server machine is considered as the cloud server that stores the index, calculates the scores and sends request files to the user. The DRIER framework is developed in Java and utilizes HELib libraries (C++) to perform homomorphic operations.

The dataset used for testing is Thomson Reuters Text Research Collection (TRC2). It contains 1,800,370 stories, which occurred between the period 01-01-2008 00:00:03 and

28-02-2009 23:54:14. The size of the dataset is 2.8GB. TRC2 is a single long file with date, headlines and stories, stored in comma separated form. To match our testing requirement, we split this large file into multiple files, where each file is named with a date in ddmmyyy.txt format and the content of that file is the headlines and stories on that particular day. Thus 419 files have been generated, where each file size ranges from 8MB to 16MB.

6.6.2 Performance Analysis

The performance of the proposed scheme is dependent on different stages of the framework like index generation, trapdoor generation, score calculation, decryption of score at client side, and score ranking.

i. *Index Generation Time*

Initially, we need to run the setup(λ) algorithm to derive the public and secret keys needed for encryption and decryption. To reduce the tradeoff between security and efficiency, the value of λ is fixed as 128. The secret key will be a value between $[2^{\eta-1},2^{\eta}]$. The set of public keys, PK = $\{k_0,k_1,k_2, \dots , k_t\}$ ε $(-2^{\rho},2^{\rho})$, where ρ indicates bit length of noise. Thus, the complexity of this stage will be $O(\lambda^{\eta\rho})$ which is a constant, as λ is constant.

The index building stage involves tf-idf calculation and homomorphic encryption. To reduce the execution time of index building of large data, we implement a distributed map reduce parallel programming model that reduces the complexity to linear and it will be explained in chapter 7. Also, tokenization, stemming and stop word elimination is done, to reduce the volume of keywords to be indexed. To update the documents, re-iteration of the entire index building stage is needed and to avoid such a scenario, we store idf values separately. Hence, only the updated part of the file needs to be re-evaluated to find tf-idf of the corresponding words. Encryption can be implemented in O(dw) time, where d is the number of documents and w is the number of words. Time needed to generate the index is same for all methods, whereas the time needed to encrypt the index using traditional SSE, FHE and proposed MHE is as given in Figure 6.5. Here, the index size is always very much less in size compared to original file, because there is lot of repeated words in the original text.

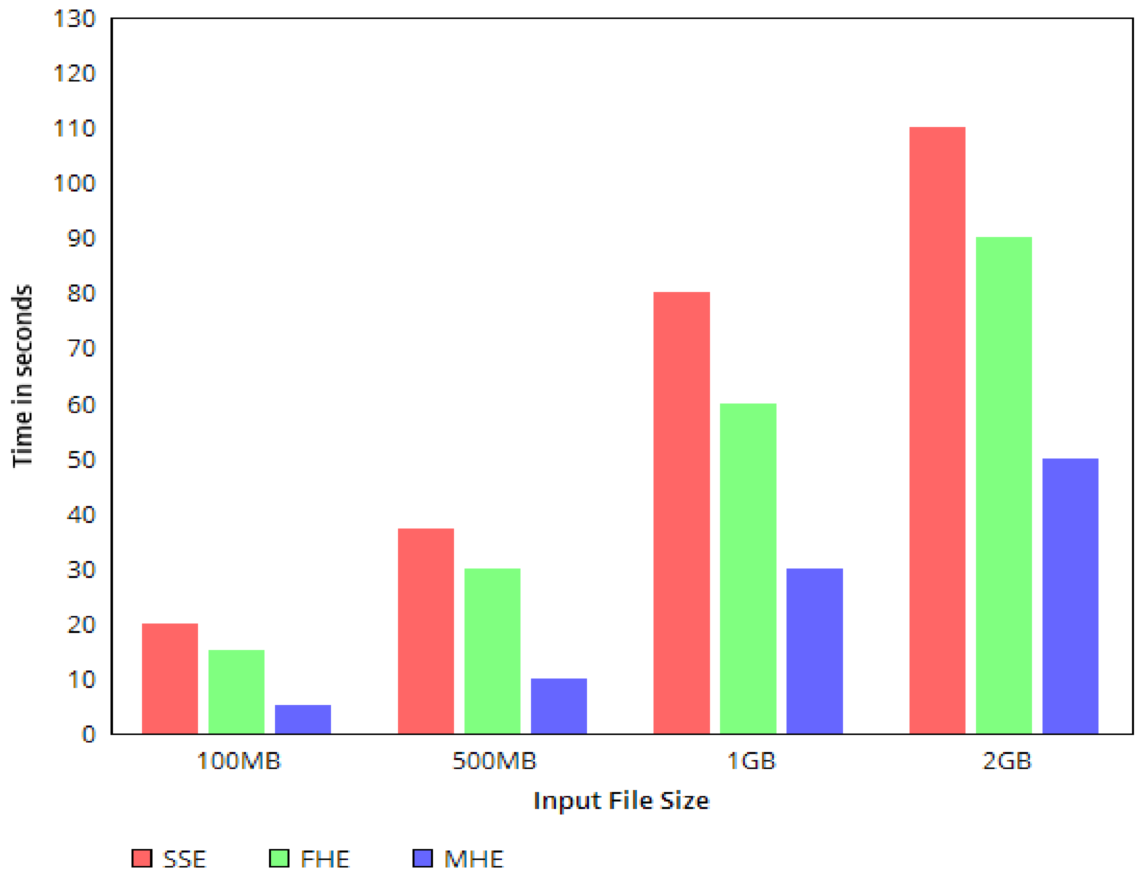


Figure 6.5 Encrypted Index Generation Time

ii. *Trapdoor Generation Time*

The retrieval phase includes different stages like Trapdoor generation, Score Calculation and Sorting & shuffling, to identify the top-k documents. The complexity of our proposed scheme is highly dependent on the retrieval phase, as this has to be repeated each time a user posts a query. Hence, we parallelize the most time-consuming retrieval phase i.e., sorting and shuffling of top-k results. The concept is described in Chapter 7.

Trapdoor Generation involves the binary conversion of a posted query and the homomorphic encryption of each bit. If the query contains n keywords, then the complexity will be O(n). Figure 6.6 illustrates the time needed to compute the Trapdoor by employing homomorphic encryption, as well as traditional searchable symmetric encryption (SSE), by varying the number of total distinct words in the document collection and the number of terms in a query. It is well observed that, the execution time is approximately half for our proposed modified homomorphic encryption scheme (MHE).

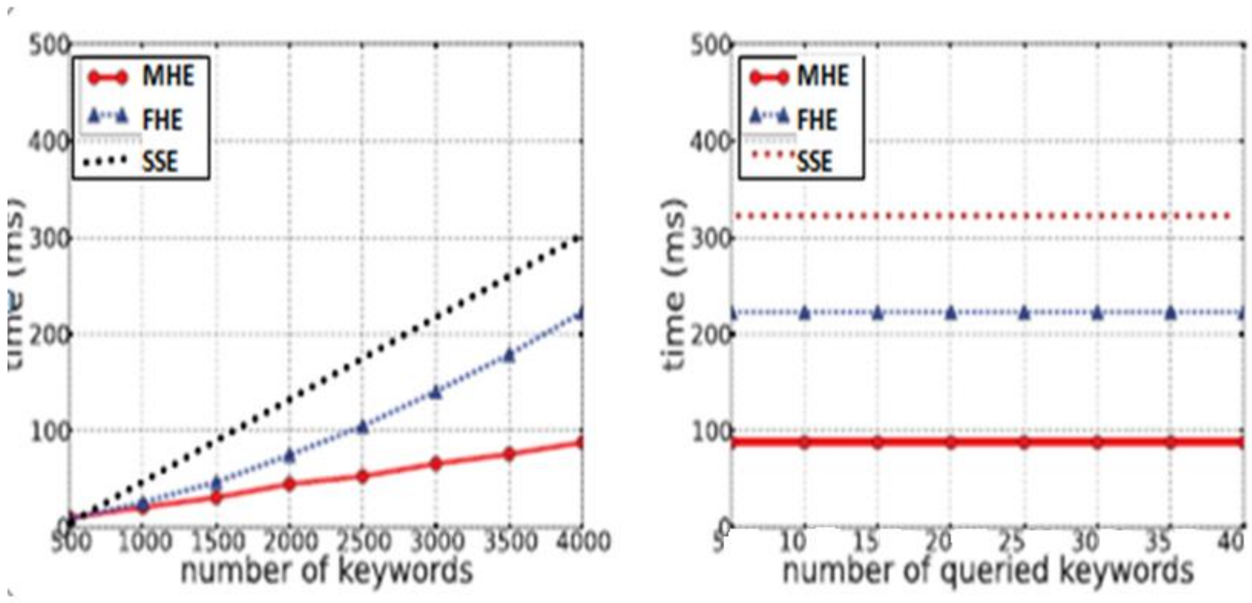


Figure 6.6: a) Trapdoor Generation Time by varying the total number of words in the document collection b) Execution Time by varying the total number of terms in the query.

iii. *Score Calculation Time*

To calculate the encrypted similarity score, the inner product has to be performed. This calls for w multiplications and d additions, where w is the number of words and d is the number of documents which lead to a complexity of O(wd). Here, the execution time varies with a variation in the number of query terms and documents. The comparison is illustrated in figure 6.7. It is well observed that, for the MHE scheme, the execution time is increasing almost linearly, whereas for SSE, it is an exponential increase.

iv. *Decryption of Score at client side*

Decryption of scores to obtain the similarity score is done at the client side and the number of terms to be decrypted depends on the total number of documents in the collection. Hence, the complexity will be utmost O(d). If there are too many documents, then distributed parallel processing can be employed to decrypt the terms. Figure 6.8 illustrates the decryption time of MHE scheme.

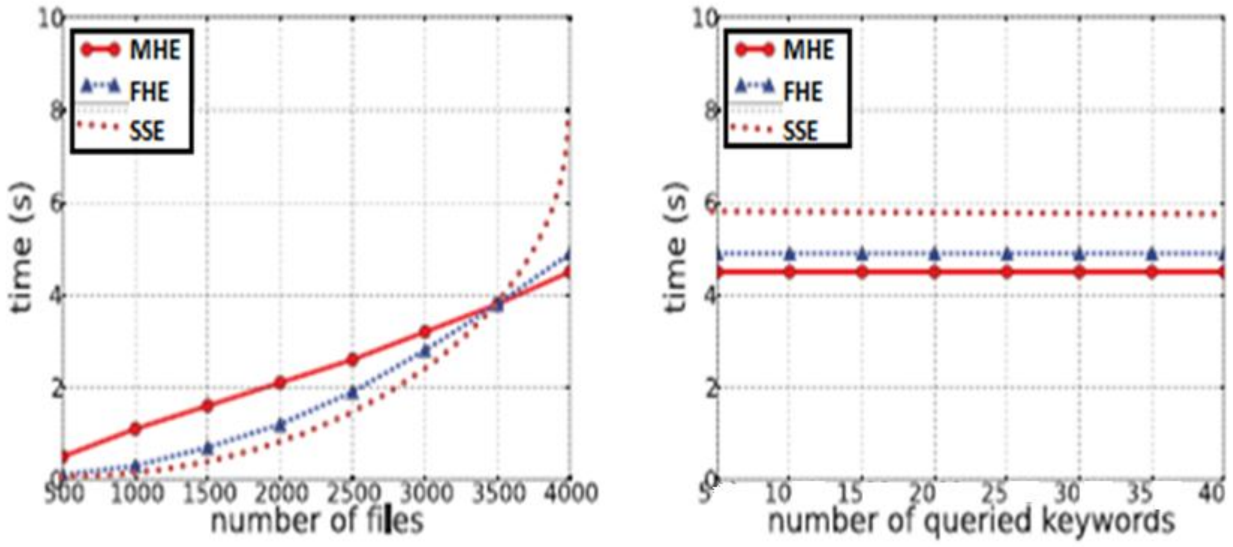


Figure 6.7: a) Encrypted Index Score Calculation Time by varying the total number of words in the document collection. b) Execution Time by varying the total number of terms in the query.

v. *Score Ranking Time*

Ranking and shuffling of File identifiers based on similarity score is the last stage to be executed, to identify the most similar documents. Modifying the sorting algorithm as described in algorithm 1, will itself reduce the execution time to O(d.n) and by introducing MR programming, it can be again reduced to O(d). More reduction is possible by introducing heap tree implementation.

Figure 6.9.a shows the execution time of ranking, with a variation in k, where k denotes the number of similar files to be retrieved by the user. Here, the number of documents is set to 1000. Then, figure 6.9.b illustrates how the performance varies, with an increase in the query terms. As the number of query terms change, there is no much observable difference in the execution time, as scoring is independent of number of terms queried. Scoring is dependent on the number of top files to be retrieved.

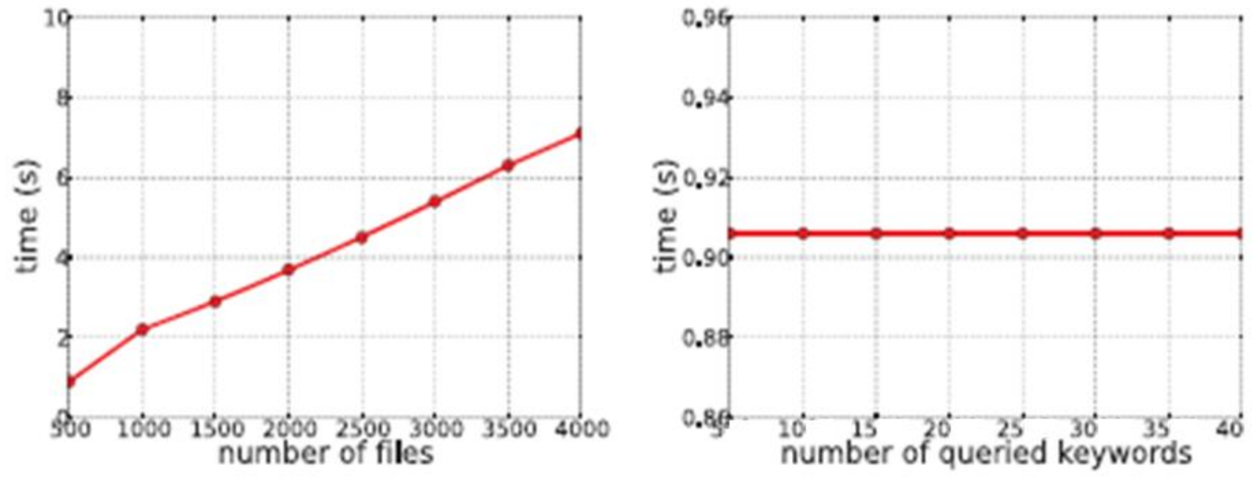


Figure 6.8: a) Decryption time by taking number of query terms constant b) Decryption time by taking number of files constant as 500

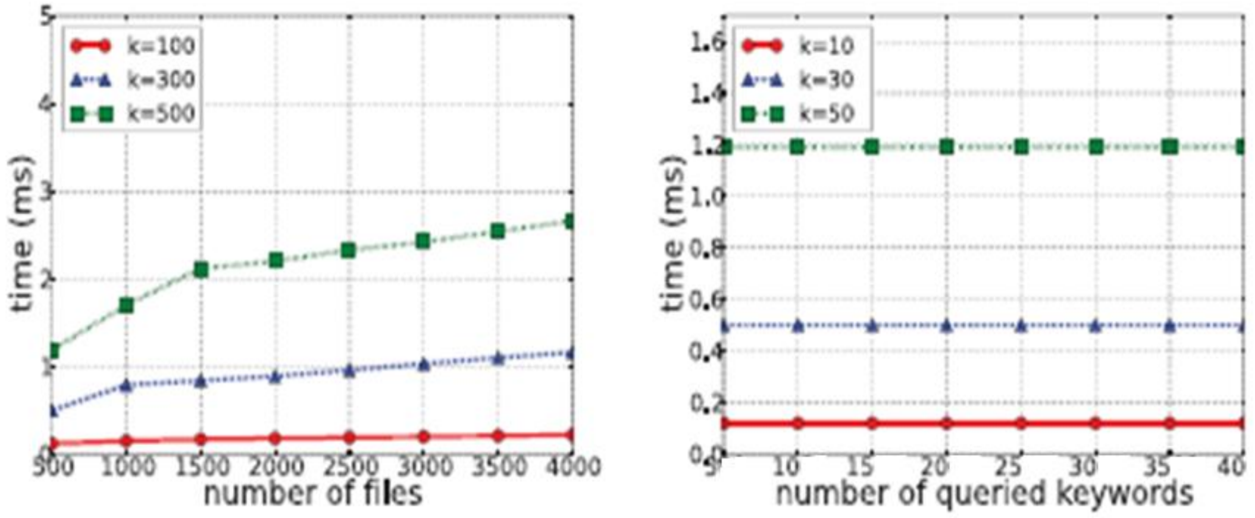


Figure 6.9: a) Ranking Time by varying number of files b) Ranking Time by varying the number of keywords in the query.

vi. *Communication Overhead*

The core of our approach is the homomorphic encryption of vectored index, which eliminates the need for transferring the entire index to the client side, for decryption and ranking. The score is calculated at the server side itself, and only the encrypted scores are forwarded to the client for ranking. Consider that there are 100 files and 1000 distinct keywords. Then, the size of the index file to be transferred for traditional SSE, will be approximately 100x1000x1024 bits (equal to 12MB), if each cell value is set to 1024 bits. But, for the Modified Homomorphic Encryption Scheme, it will only be 100x1024 bits, which are equal to .01MB. Hence, there is a large variation in the amount of data to be transferred through the network, when we compare SSE and MHE.

Summary

This chapter describes how the proposed Modified Homomorphic Encryption algorithm can be applied for secure and searchable encrypted vectored index construction and how to retrieve the top matching similar documents from the documents uploaded in a third party server. The algorithm for different stages as well as the security and performance analysis of the proposed scheme is compared. From the analysis of results, it is observed that, the performance of the DRIER framework is highly dependent on the retrieval stage, as it should be repeated each time a request comes from the user. Also, most of the operations in retrieval stage can be parallelized, which could reduce the execution time a lot.

Chapter 7

Accelerating MHE using Map Reduce

"The future has already arrived. It's just not evenly distributed yet."

- William Gibson

Distirbuted Programming powered by MapReduce, can significantly increase the performance of information retrieval processes. This chapter describes how to improve the DRIER performance, utilizing the Map Reduce programming model.

7.1 Introduction

Performance analysis done on the MHE scheme showed higher execution time for encrypted index generation and retrieval stages. This chapter details on how to improve the performance of different stages of the proposed DRIER framework, using distributed programming. Hadoop is utilized as a distributed platform and Map Reduce programming model is adopted for distributed processing. Since scoring happens at the cloud end, we are not describing any mechanism to improve the performance, even though the proposed scheme supports parallel execution.

7.2 Accelerating Secure Index Generation

Secure Index generation composes of two phases. First, the inverted list and vector space is computed and filled with TF-IDF values. The TF-IDF values thus obtained are encrypted using MHE, in the second phase. We utilize a 2 stage Map Reduce strategy as shown in figure 7.1 to accomplish the two phases of this secure index generation, and they are as described here.

Stage 1: Inverted Index Creation with the frequency of occurrence

An inverted index is a data structure which stores the details of mapping from words to files. After forming the inverted index, we need to form a vector space model, with each cell containing TF-IDF values. In order to simplify the vector space generation stage, we calculated the frequency of occurrence of each word in each document, simultaneously with the inverted index creation. This list is kept separately so that it can be re-used, when some modifications happen in the input document collection.

At Map-Reduce stage 1, each mapper (worker) will take a collection of documents and find the distinct tokens, remove the stop words, and return the count as 1, whenever an occurrence is found. Thus the <key,value> output of each mapper will be <word,1>. At the reduction stage, the reducer will combine the outputs from all mappers, to find the number of times each word is repeated in the whole document collection. Thus the reducer output will return a list of [<word, frequency>]. From the obtained results, TF-IDF is calculated and the final vector space is formed.

Stage 2: Encrypted Vector Index Generation

At Map-Reduce stage 2, each Mapper is programmed to process the TF-IDFs of distinct files. The Mapper will apply MHE encryption (chapter 5, section 5.2) to each TF-IDF value in its input list, to form the secure encrypted index. Thus, the output of each mapper will be <word, MHE(TF-IDF)>.The entire output from different mappers, is then merged to form the final encrypted vector space index. Now, remove the words and file IDs from the list and save it to be used as keys for decryption. This is then uploaded to cloud. Stage 2 does not require a Reduction stage. Figure 3 illustrates the details of Map Reduce Stages. Encryption of each document can also be done efficiently, by adding one more Mapper stage.

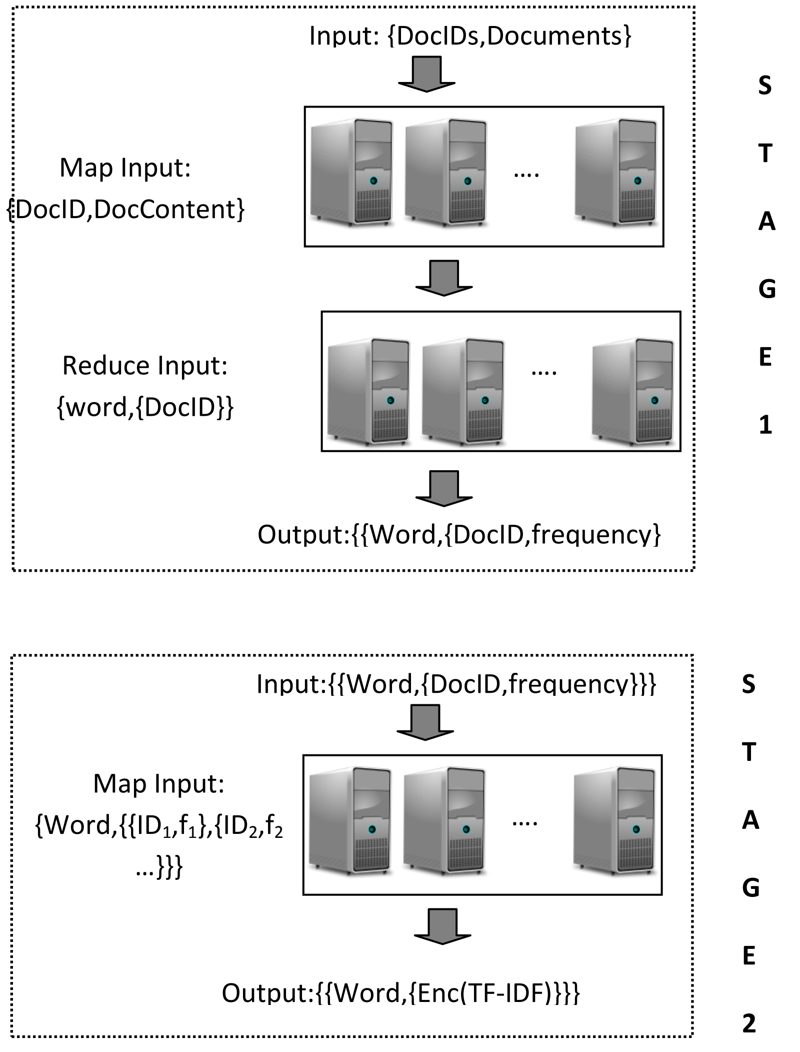

Input: {DocIDs,Documents}
Map Input: {DocID,DocContent}
....
Reduce Input: {word,{DocID}}
....
Output:{{Word,{DocID,frequency}
STAGE 1
Input:{{Word,{DocID,frequency}}}
Map Input: {Word,{{ID$_1$,f$_1$},{ID$_2$,f$_2$ ...}}}
....
Output:{{Word,{Enc(TF-IDF)}}}
STAGE 2

Fig 7.1: Map Reduce Implementation of Secure Index Creation

7.3 Ranking to retrieve K-Similar Documents

To retrieve top k similar documents, split the scores and give it to ‘X’ number of mappers available, where X<N. Each mapper evaluates algorithm 4 described in chapter 6, to find the top K scores, which are of highest value in their input split. Reducing the output from ‘X’ Mappers in a merged manner will give the top K matching documents.

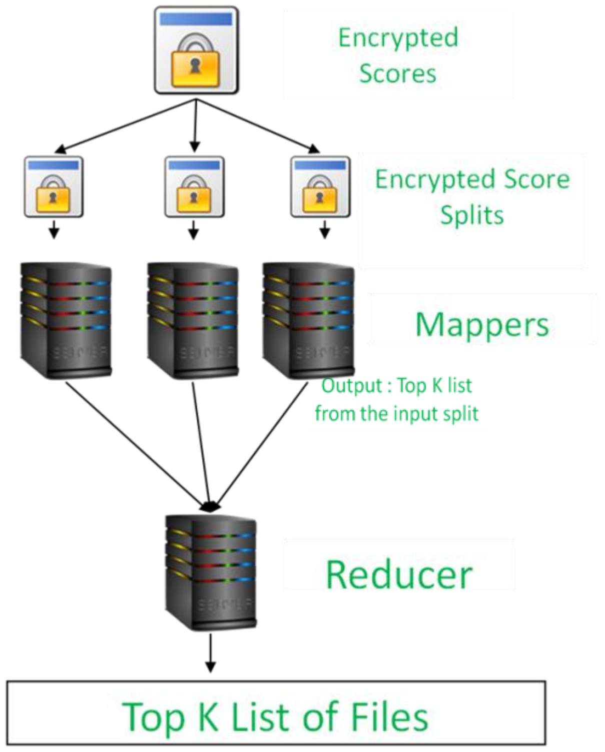


Figure 7.2 Map Reduce implementation of Ranking of files.

Figure 7.2 gives a higher level view of the Map Reduce implementation for ranking. The entire secure vector space is split. The detailed explanation with an example is provided next.

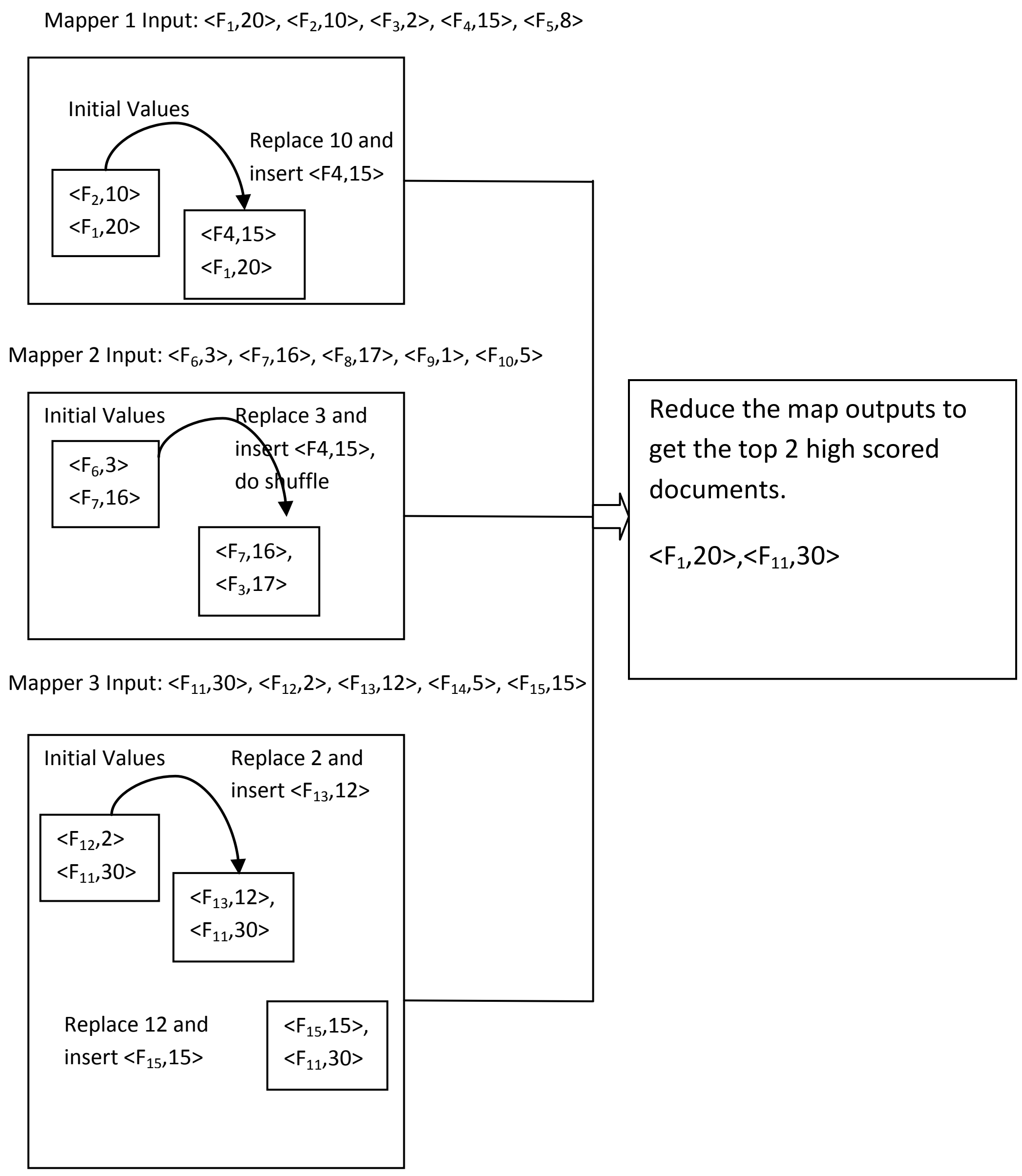
Input to Mapper : <RecordNumber,Encrypted Score>

Output of Mapper : <MapperID,

list of top 2 scores[<RecordNumber,Score1>,<RecordNumber,Score2>]>

Output of Reducer : <”Top”, list of top 2 scores[<RecordNumber,Score1>,<RecordNumber,Score2>]>

A detail of the scheme is illustrated with an example here.



Mapper 1 Input: <F1,20>, <F2,10>, <F3,2>, <F4,15>, <F5,8>
Initial Values
Replace 10 and insert <F4,15>
<F2,10> <F1,20>
<F4,15> <F1,20>
Mapper 2 Input: <F6,3>, <F7,16>, <F8,17>, <F9,1>, <F10,5>
Initial Values
Replace 3 and insert <F4,15>, do shuffle
<F6,3> <F7,16>
<F7,16>, <F3,17>
Mapper 3 Input: <F11,30>, <F12,2>, <F13,12>, <F14,5>, <F15,15>
Initial Values
Replace 2 and insert <F13,12>
<F12,2> <F11,30>
<F13,12>, <F11,30>
Replace 12 and insert <F15,15>
<F15,15>, <F11,30>
Reduce the map outputs to get the top 2 high scored documents.
<F1,20>,<F11,30>

Figure 7.3 Ranking scores using Map Reduce

Let K=2, N=15 and X=3, where K is the number of top similar documents to be retrieved, N is the total number of files and X is the number of mappers available.

Let the input to the Map Reduce stage with <DocID, Score> be : $<F_1,20>$, $<F_2,10>$, $<F_3,2>$, $<F_4,15>$, $<F_5,8>$, $<F_6,3>$, $<F_7,16>$, $<F_8,7>$, $<F_9,1>$, $<F_{10},5>$, $<F_{11},30>$, $<F_{12},2>$, $<F_{13},12>$, $<F_{14},5>$, $<F_{15},15>$

Since N= 15 and X=3, each mapper receives N/X, i.e. 3 tuples.

Here, for the first mapper, input is <F_1,20>, <F_2,10>, <F_3,2>, <F_4,15>, <F_5,8>. As <F_1,20>comes, top list is updated with a value 20. Then as <F_2,10> comes, the top_list is updated by inserting 10 to the top of the list, as 10<20. Next comes <F_3,2>. But no change is needed for the top_list as 2 is less than 10. For the next record, <F_4,15> , 10 is swapped out from the top_list and 15 is added. No change when <F_5,8> arrives, as 8 is less than 15. At the reduction stage again a top_list with k entries is created and updated, following the same strategy described above.

A point to be noted here is, the mapper will do decryption of encrypted scores before adding it to the top_list data structure. Ranking and shuffling of file identifiers based on similarity score, thus identifies the most top K similar documents. Modifying the sorting algorithm as described in chapter 6 itself will reduce the execution time to O(d.n), and by introducing MR programming, it can be again reduced to O(d). More reduction is possible by introducing heap tree implementation, where ‘d’ is the number of documents and ‘n’ is the number of words.

7.4 Experimental Setup and Evaluations

During the initial phases of the research, we utilized a virtual machine setup to run Hadoop. To analyze the performance in a distributed environment, Amazon Web Services is utilized.The specifications of machines used in AWS are as follows:

CPU	Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz
Cores	2 to 10
Cache Size	20480 KB
OS	UBUNTU Linux
Linux Kernel	3.2.30-49.59.amzn1.x86_64
Hadoop version	1.0.3
Hive Version	0.11.0.1
File System Size	8GB

The experiment is evaluated on a 10 node Hadoop cluster setup on Amazon Web Service (AWS). The namenode is a t2.large instance. The secondary namenode and datanodes are t2.micro instances. All machines are Ubuntu 14.2, installed with OpenJDK 1.7 and Hadoop stable version 1.0.2. The HDFS Replication factor is set to 3 and the HDFS block size is 8MB. To enable security between Hadoop machines, Kerberos protocol is enabled.

The dataset used for testing is Thomson Reuters Text Research Collection (TRC2). The dataset contains 1,800,370 stories, which occurred between the period 01-01-2008 00:00:03 to 28-02-2009 23:54:14. The size of the dataset is 10GB. TRC2 is a single long file with date, headlines and stories stored in a comma separated form. To match our testing requirement, we split this large file into multiple files, where each file is named with a date in ddmmyyy.txt format and the content of that file is the headlines and stories on that particular day. Thus 419 files have been generated, where each file size ranges from 8MB to 16MB. We replicated the files to make it 10GB. To store and retrieve these small files efficiently from Hadoop Distributed File System, we followed a technique called Balanced Multi-FileInputSplit (BaMS) Technique. BaMS is proposed to avoid small file problem in Hadoop. For Hadoop to access small files, it is always an overhead. BaMS is introduced to rectify that. The technique will be described in detail while discussing the applications of the proposed algorithm in chapter 8.

7.4.1 Performance Evaluation

This section compares the execution time of different stages of the information retrieval scheme proposed.

Secure Index Generation Time

Secure Index generation using a Map Reduce Programming model, utilizes two stages. The first stage will create the inverted index and the second stage will generate the encrypted vector space. The execution time needed for each stages is compared, and it is found that the time needed by Map Reduce programs are very much less.

Figure 7.4 shows the time needed for inverted index creation. The time needed can be reduced to milliseconds range by increasing the number of workers. Here, we considered sizes upto 2GB, as it is not possible to generate the encrypted index beyond that using

MHE, in a single machine having the mentioned specifications. Figure 7.5 shows the time needed for secure index generation. Thus, the total time to generate the final secure searchable index can be reduced to milliseconds, since distributed processing is introduced.

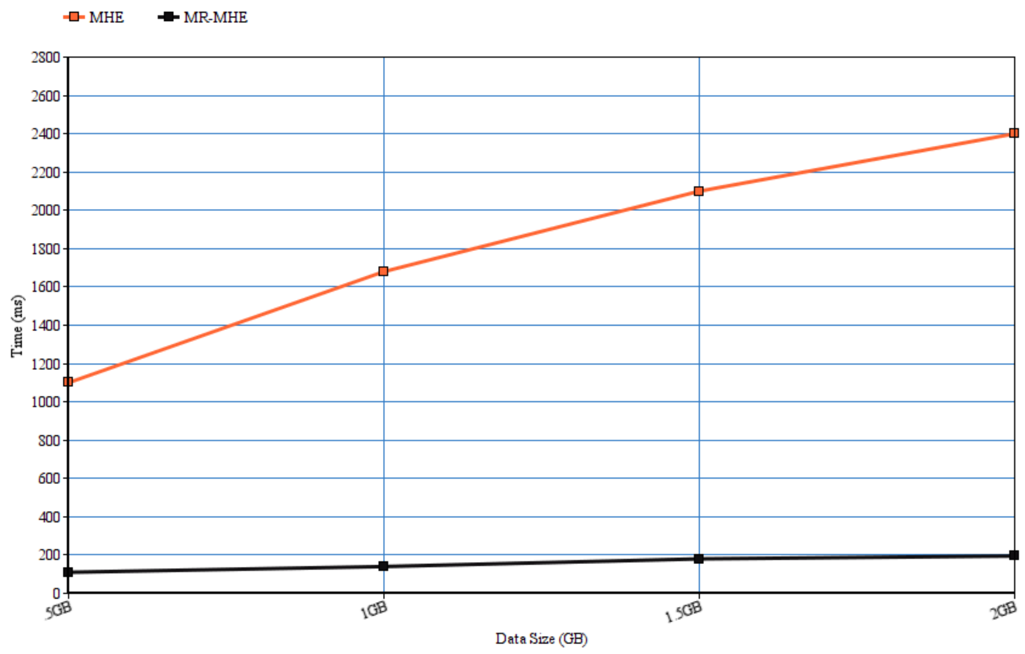


Figure 7.4 Comparison of inverted index creation time

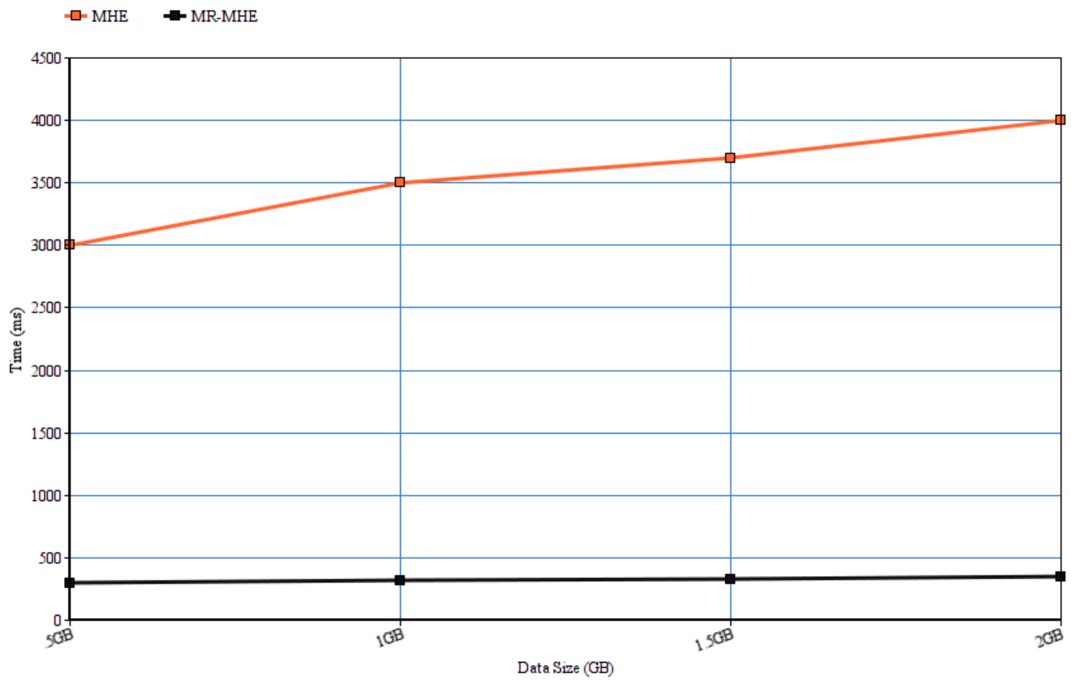


Figure 7.5 : Comparison of secure index generation time

Decrytpion and ranking time

Using the Map Reduce programming model, decryption and ranking is done using a single stage Map Reduce program. Figure 7.6 shows the variation in the decryption and ranking time, when the decryption of scores is done using a map reduce model. As the execution happens in parallel and distributed, the time needed is very less as compared to the original MHE. Another observation is that, as the number of files increase, there is no much varitaion in the execution time, if the operations are performed parallel. Also, the time needed can still be reduced by increasing the number of worker nodes. In the figure, MR-MHE implies the Map Reduce version of the original MHE scheme.

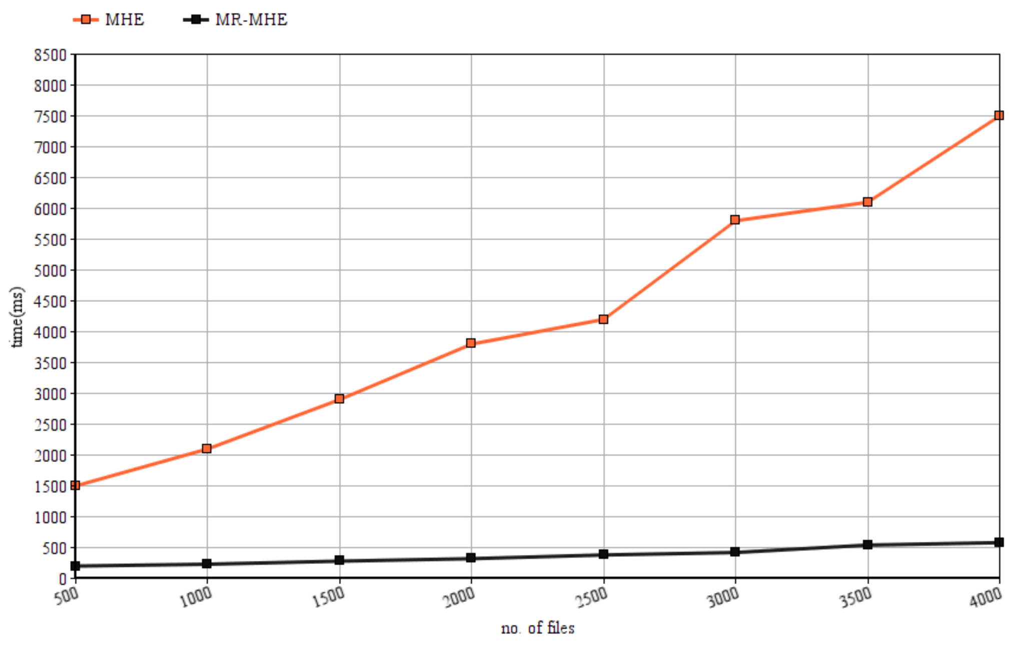


Figure 7.6 Decryption time using Map Reduce programming modal

Next, the change in execution time by varying the number of data nodes, is observed and plotted. File size is increased to 20 and 50GB by replicating the original files we have. Figure 7.7 shows the change in secure index generation time by varying the number of datanodes. As the number of datanodes increase to 50, even if to process large amount of files, the time needed becomes almost a constant. Also, another thing to be noted is that, if the number of datanodes is too large, and the file size is small, then, due to the overhead in processing, the execution time may become greater for small files. According to the study conducted on the execution time comparison by varying the datanodes, it is observed that fixing the datanodes to 10 is always optimal, even if the size of files increases.

Figure 7.8 shows the change in decryption and ranking time by varying the number of files in the input list. Here, the number of datanodes is varied from 5 to 20. But, the analysis shows that, even if the number of files in the input list is huge, there is not much variation in the decryption and ranking time. Also, from the study, it is obvious that the datanodes from 5 to 10 is sufficient to rank the documents, even if the number of files increases to 3 lakh.

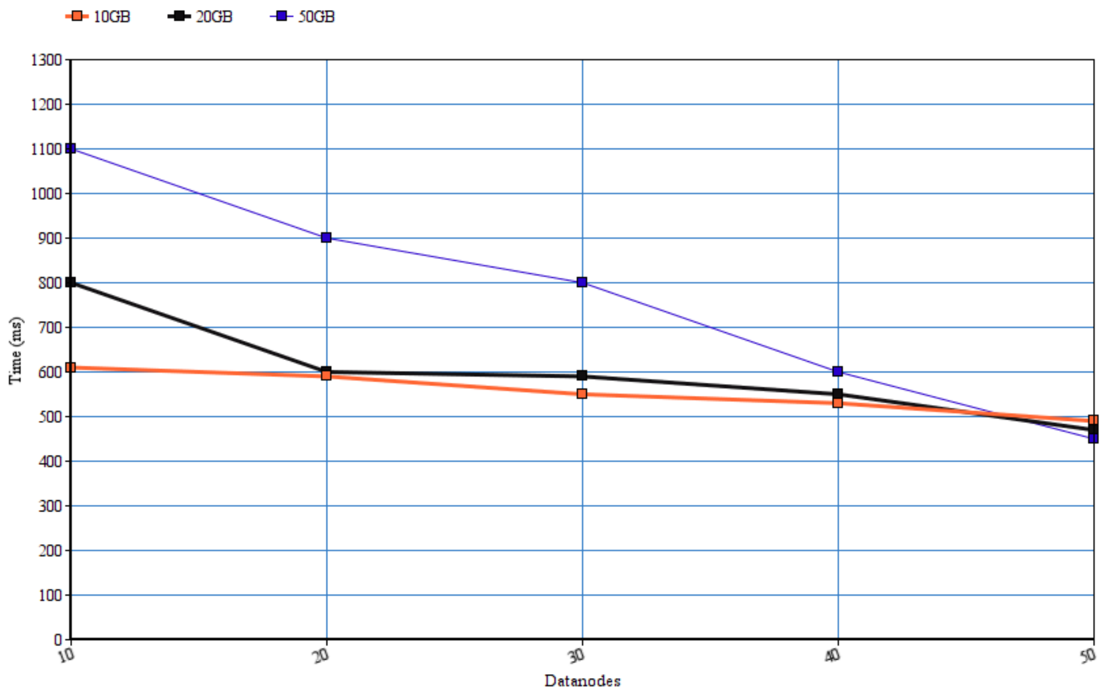


Figure 7.7 Comparison of secure index generation time by varying the number of datanodes.

Next, the change in execution time by varying the number of query terms and the number of top K files to be retrieved is analyzed. Figure 7.9 shows the comparison. Here, the number of datanodes is kept a constant 10, and the size of files is 10GB. It is observed that, the execution time for Map Reduce MHE is almost half for different K values. And as the number of query terms increase, there is no change in execution time.

Speedup

Scalability of the proposed MHE scheme is evaluated using a SpeedUp metric. The SpeedUp factor defines the ratio of time needed to execute an algorithm in one machine, to the time needed to execute it on *N* machines. In an ideal case, the method is considered scalable, if the speedup factor remains constant for different values of *N*.

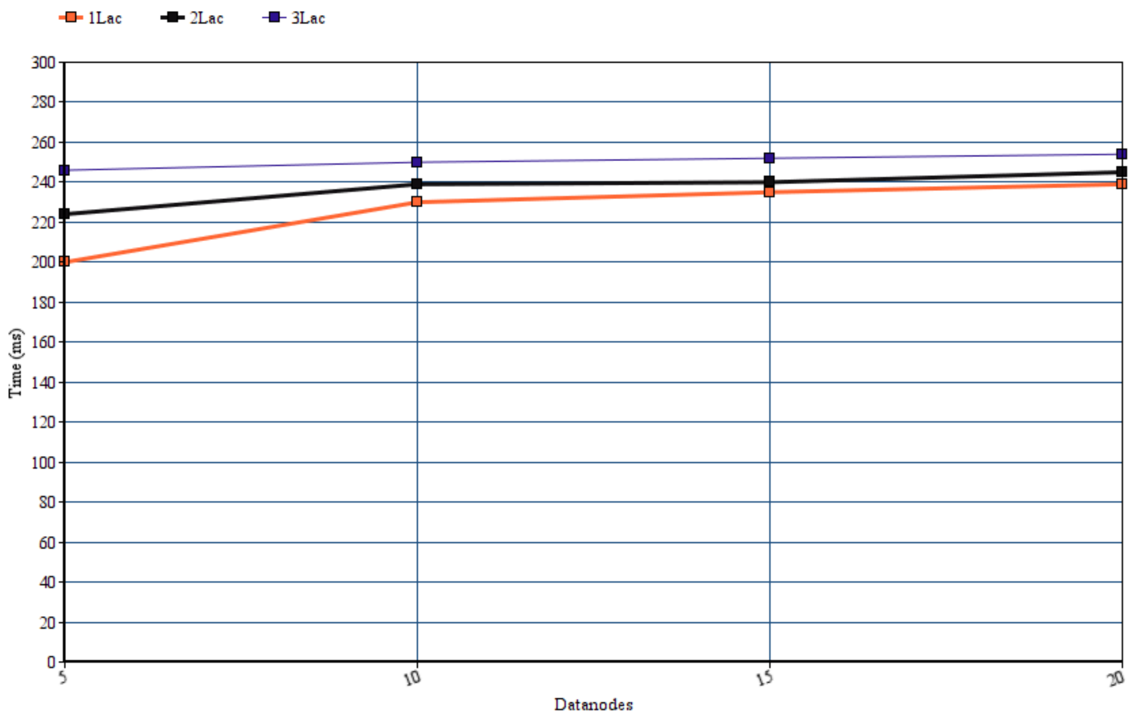
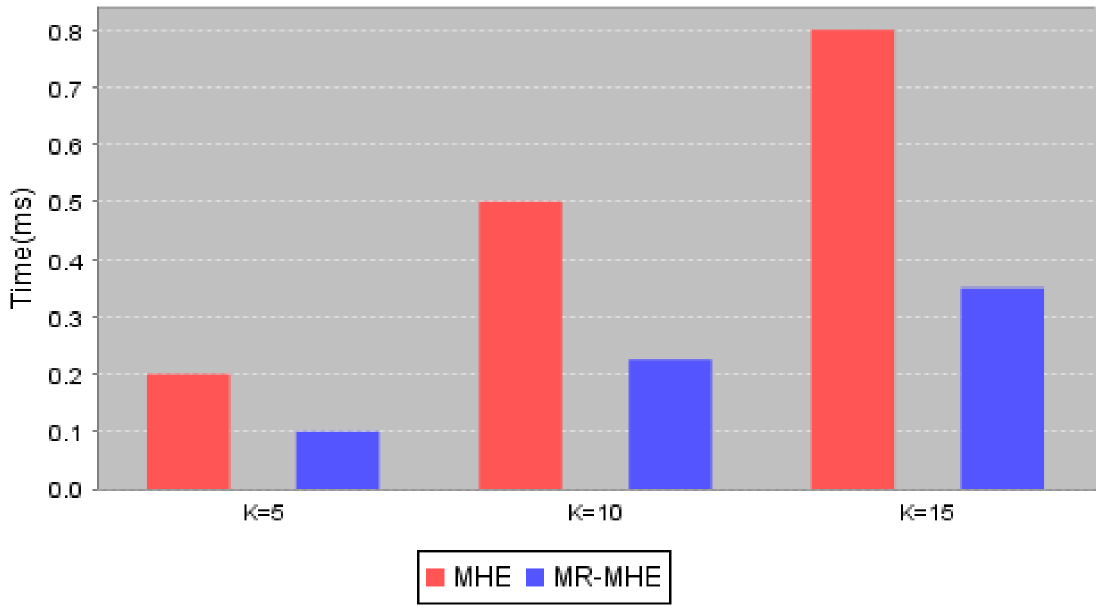


Figure 7.8 Decryption and Ranking time by varying the number of datanodes

$$SpeedUp,\ S_u = T_1/T_N \ \ldots\ldots\ (7)$$

Figure 7.10 illustrates the change in execution time for retrieving the files, varying the data size and the corresponding speedup. From the figure, it is clear that, even if the data size increases, there is no much variation in the execution time, as the number of nodes increase. Thus, the algorithm is becoming more scalable, and approaching ideal values with the increase in data size. Also, for practical scenarios, the number of datanodes can be fixed at 10, to achieve optimal performance for varied file sizes and query terms.



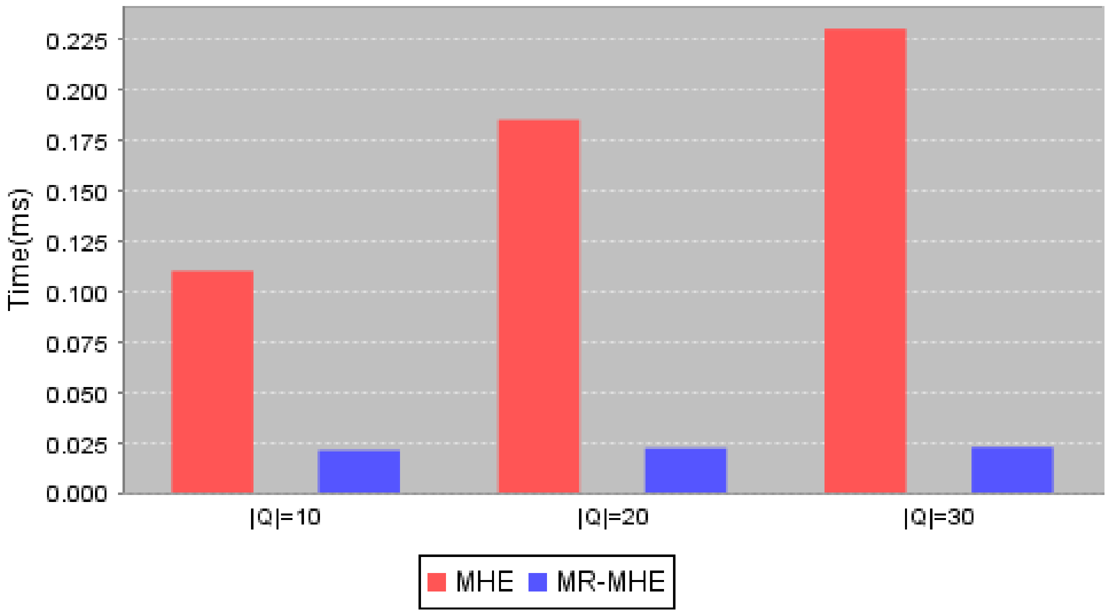
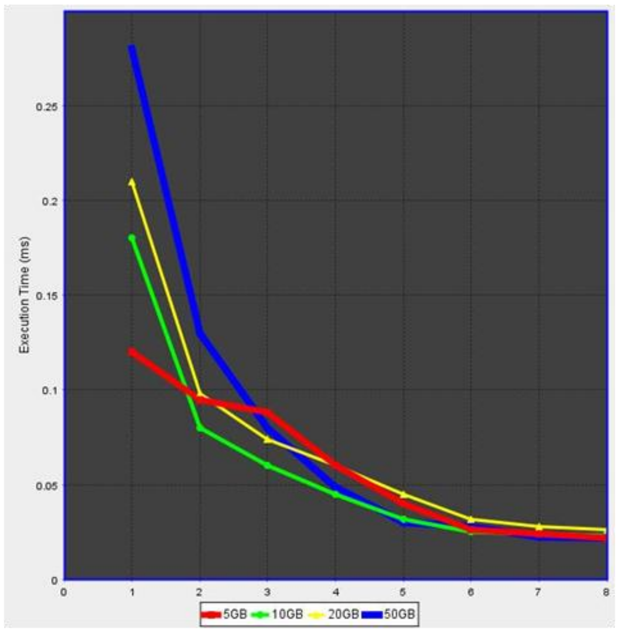


Figure 7.9 Comparison of Ranking time (a) Varying K value (b) Varying the number of query terms



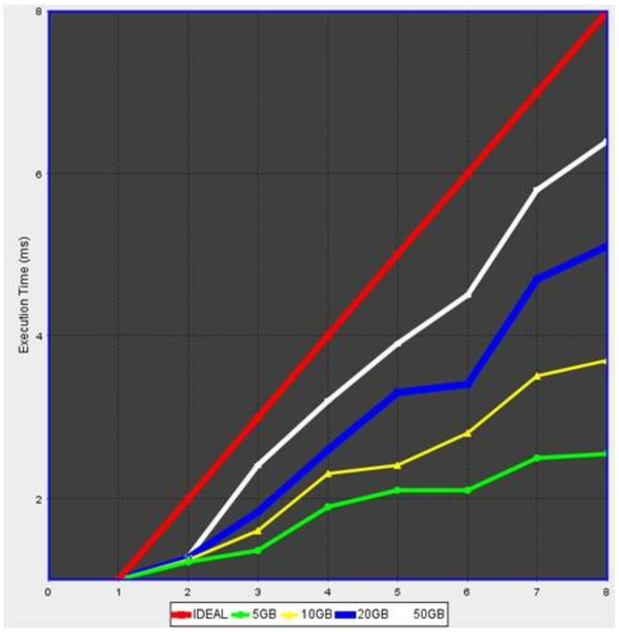


Figure 7.10 Comparison of Speedup (a) Varying the data size (b) Speedup

7.6 Distributed Methodology followed in DRIER

All the schemes mentioned in the thesis follow distributed and parallel execution strategy. Encryption and decryption in MHE scheme is implemented using Map Reduce which can be adopted for Hadoop distributed platform. Searching, ranking and retrieval are parallelized using Map Reduce which significantly improves the query response time. To summarize, all the schemes mentioned in the thesis can be parallelized using Map Reduce programming model which makes it suitable for any distributed applications like client server, peer to peer or cloud computing.

Summary

This chapter explained how to improve the performance of the proposed dual round encrypted information retrieval scheme, using distributed programming. Hadoop is used as the distributed framework. By increasing the number of workers to find the secure index and ranking, the speed of execution can be improved. From the performance analysis done on the distributed system, it is observed that by setting 10 to 15 node Hadoop cluster, the speed up of the system can be nearly made equal to ideal values.

Chapter 8

Applications of the Modified Homomorphic Encryption Scheme

“Privacy is not an option, and it shouldn’t be the price we accept for just getting on the Internet.”

– Gary Kovacs

Securing personal information as well as organizational information, is getting very difficult nowadays, with the emergence of internet and cloud computing. This chapter deals with two significant applications that demand high level security, and how to achieve that using the proposed MHE scheme.

8.1 Introduction

This chapter discusses some important applications where, the proposed secure and privacy preserving information retrieval scheme can be efficiently utilized. Different applications where the MHE based information retrieval can be utilized are electronic voting, secure Email servers, secure log analysis, storage of data that is of national importance, and many more. From the wide list of applications, this chapter describes the design criteria and method by which the proposed MHE scheme can be applied for secure storage of Emails and logs.

8.2 Secure Email Servers

Today, Email is becoming the easiest, most inexpensive and comparatively faster method of personal and formal communications. Many people utilize the free Email service provided by Google, Yahoo etc. Private organizations maintain their own mail servers, to ensure more privacy and security for the users and the data transferred. But, as the number of employees and the size of mails increase, these organizations should maintain a good infrastructure for the efficient storage, which will result in a heavy maintenance cost. Cloud computing comes to the rescue here. But, ensuring the privacy and security of the users and Emails is a challenging issue. “Hilary Clinton’s Email Leak”, “Effect of Email Leak during French Elections” [63] etc are the result of inefficient and insecure storage and transfer of Emails.

My research proposes a secure and privacy preserving technique to store, retrieve and transfer sensitive Emails. To ensure security, traditional encryption techniques can be utilized. Encrypt each Email before passing through the network and decrypt it at the receiver side. Also, before storing the mails in the cloud system encrypt it. Thus, the storage and transmission of the encrypted mail is possible, by utilizing existing well known cryptosystems. But the search and retrieval of some specific mail, is the difficult part. Since all mails are stored in an encrypted form, the direct solution is to download all the mails to the client machine, decrypt them and find the matching mails. But this will consume a large bandwidth and hence, is not at all an economic solution, considering the pay-as-you-use pricing model of the cloud. Also, if there are too many mails, the download and decryption of each mail will be a time consuming task, and will not be feasible if the client machine does not have much processing capability.

8.2.1 Motivation and Problem Definition

Electronic mail (Email) or the paperless mail is now becoming the most acceptable, fastest and cheapest way of formal and informal information sharing between users. Around five hundred billion mails are sent each day and the count is expected to be increasing. Today, even the most sensitive and private information is shared through Emails, thus making it the primary target for attackers and hackers. Also, the companies having their own mail server relies on the cloud system, to store the mails at a lower cost and maintenance. This affects the privacy of users, as the searching pattern is visible to the cloud. To rectify this, we need to have a secure architecture for storing the Emails and retrieving them according to the user queries. Data as well as the queries and computations needed to retrieve the relevant mails should be hidden from the third party.

The scenario given in figure 8.1 illustrates the need for a secure Email server. Alice is working in ‘ABC’ Company, which processes information dealing with the national security. They maintain their own mail server for the transfer of mails between their employees. The mail server is hosted on a Cloud system. Hence, to ensure the security, mails are stored in an encrypted form. Later, to retrieve all mails related to “Mission X”, either Alice needs to download all mails to her system, decrypt them & search, or, she has to decrypt all mails at the Cloud system and search & retrieve only the specific mails. The former method wastes a lot of bandwidth and latter results in security violation, as decryption is done at a cloud machine.

8.2.2 System Design for Secure Email Storage

Secure Mail Servers encrypt each mail before passing it through the network. The Public Key Cryptosystem powered by LDAP is utilized for this. For the storage of mails as well as for the secure transfer of mails, traditional cryptographic techniques are utilized, as it is found to be more efficient and less time complex. For encrypting the mails, AES is utilized. Each mail is encrypted by the user’s secret key, and then uploaded to the cloud.

While fetching a particular mail, the same key is used for decryption. Also, while sending a mail, it is encrypted by the receivers' public key, using the RSA encryption system. The receiver can use his secret key to decrypt and view the contents of the mail.

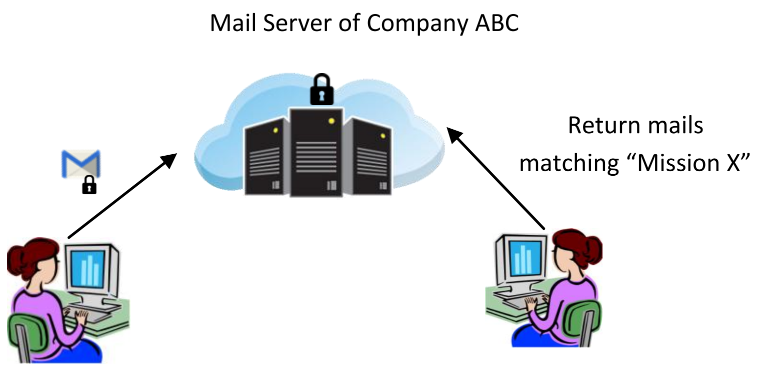


Figure 8.1: Scenario illustrating the need for secure Email server

Each mail will be stored in the cloud system in an encrypted form. To search and retrieve the matching mails from this encrypted domain, a vector space is generated and encrypted using the Modified Homomorphic Encryption scheme (discussed in chapter 5). A two round search and retrieval strategy is followed. During the first round, a trapdoor is generated with the query keywords, and is used to calculate the encrypted score of each mail. The cloud system will return the Mail-ID along with the encrypted score, to the user. The user will decrypt the scores, rank them and send the top-K mail-IDs to the cloud. The cloud will now send the corresponding encrypted mails to the user in the second round of communication. Figure 8.2 illustrates the two round search and retrieval scheme. Secure mail storage and transmission is achieved using traditional cryptosystems. Details on how to securely retrieve the relevant mails are discussed in the next section.

To search for a particular mail containing some query keywords Q = $(q_1,q_2,\ldots q_n)$, a string S is generated, which is a combination of 0s and 1s. The length of the string will be equal to the size of the Wordlist. Corresponding to each word in the Wordlist, if that word is present in the query, it will be set. Otherwise, it will be unset. Each bit of this string is then encrypted using MHE to form the trapdoor.

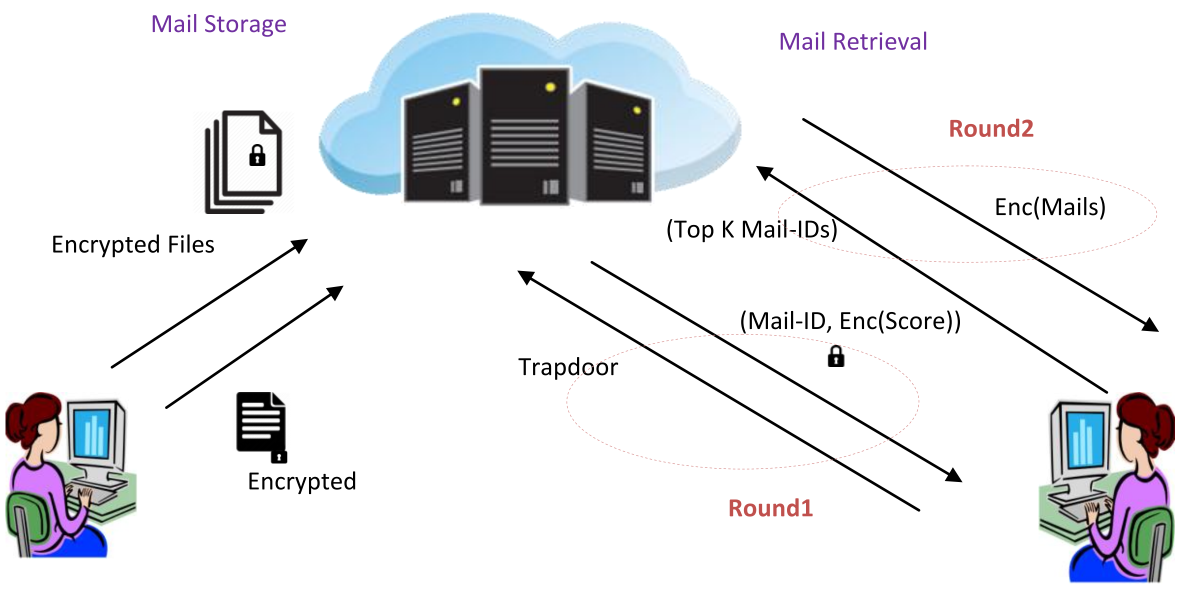


Figure 8.2: Dual Round Search and Retrieval Scheme

For each $word_i$ in the Wordlist,

If($word_i$ in Q) $S_i = 1$; else $S_i = 0$;

On receiving the trapdoor, the cloud will do multiplication and summation on the index, to obtain the encrypted scores corresponding to each column. Due to the additive and multiplicative homomorphic property of MHE, the operations done on this encrypted data will be homomorphic to the operations done on raw data. The list of encrypted scores thus obtained, is returned to the user.

$$Similarity\_Score, SS_m = \sum^{w}_{i=1} TF\text{-}IDF_{id} * Tw_i .$$

The user will decrypt the score with his secret key and rank them to identify the top-K matching mails. The corresponding mail IDs are then sent to the cloud. The cloud will return the encrypted mails, which are then decrypted at the client side. Thus, a two round communication is initiated between the user and the cloud system, to retrieve the matching mails. Decryptions take place only at the client side, to ensure absolute security. Also, compute intensive operations like score calculation takes place at the cloud, which ensures efficiency.

8.2.3 Improving the ranking of Emails

Apart from the content similarity of the mail with the query keywords, there are other factors that affect the ranking of similar documents. For example, if a user has marked one Email as 'important', then such mails shall be given some weightage, even if their similarity score is a bit less. This is achieved by adding one more row to the vector space, for including the weight of the mail. If the mail has been marked as 'important' by the user, the field will be set as 1, else 0. The value can be increased or decreased based on the application requirement. The same technique can be applied to Emails tagged as spam, promotions, etc.

8.3 Secure Log Storage

Logs store the data related to different levels of security. Companies and organizations use different types of logs, to store the activities related to firewalls, intrusion detection systems, anti malware systems, switches, routers, operating systems, applications, workstations etc. Secure logging is a critical part of any organization, and it is clearly understood by security practitioners as well as researchers. Logs record different details of a communication, from the moment a user logs into a system, to the moment the data reaches the destination. For example, when a user logs into the system, the details are logged into Active Directory Logs, then the domain access details gets logged in DNS, the browsing pattern gets logged in proxy logs and finally the packet level details gets logged in netflow logs. Analyzing the entire logs will help to identify different types of anomalies like brute force attempts (counting number of distinct user attempts within a time frame), visits to threatintel/blocked sites, and traffic outliers like DDOS etc.

To identify threats and to prevent the damages caused by these threats, companies are forced to store these logs over a long time. When the threat is identified, the logs generated over several years may be analyzed, to pin point the cause for the threat. It might have happened from a malware that was installed years back, which was activated after several years by some script. Today, efficient anomaly detection is made possible by applying Machine learning techniques. But these algorithms need large amount of training data, for effective threat analysis.

Thus, for the companies, it is very critical to store the logs over a long period. But the entire logs may sum up to terabytes of data. Dedicating some machines for log storage alone, is not a good solution. Most of the companies today make use of a third party service like Cloud storage, to sync their logs. When a threat is identified, the required logs are retrieved and analyzed.

8.3.1 Motivation and Problem Definition

The logs stored as plain text in third party systems, are vulnerable to a lot of attacks. A user who gains access to the cloud resources, can see the data and as well as obtain several critical information related to the machines of the organization, like IPs, Usernames, Passwords, Employee details, traffic details etc. To overcome this problem, encryption can be applied to the logs before uploading. However, the usefulness of your logging system will be highly dependent on how you make the information available for search, how fast you can search and whether your solution is scalable. Say for example, one day a threat is identified on a machine having the IP address ‘x’. The security engineer needs all the logs related to the IP ‘x’ for a period ‘a’ to ‘b’. Since we stored the entire logs encrypted, how can these particular logs be retrieved from the cloud? Either download the entire logs from the cloud, decrypt them and search for the needed log or else, decrypt at the cloud itself and search for the logs and download that alone. The former method leads to a lot of bandwidth wastage whereas the latter completely violates the security of the framework.

The Secure Log storage problem deals with how to securely store the logs in a third party system. such that the data is available for searching using keywords. The solution should be scalable and practical as well.

8.3.3 System Design for Secure Log Storage

The proposed MHE scheme can be used for secure storage and retrieval of the logs. Based on the logs, a secure index is generated and uploaded to the cloud, along with the symmetric encrypted version of each log file. To search for the logs meeting some criteria, issue queries with relevant keywords, generate trapdoor from the keywords and send it to the cloud. The cloud will return the encrypted similarity score corresponding to each log, and at the client side, decryption and ranking can be done to identify the most matching logs. The corresponding logs are retrieved and decrypted.

8.3.4 Challenges identified while storing logs

Challenge 1: Solving ‘Small File Problem’ in Hadoop

Logs are of different types and will be generated each second of the day. Instead of syncing the logs each second, it is better to aggregate the logs before uploading. But, as logs generated may be of very less size (most of them in KiloBytes), it is an overhead for the Hadoop framework to calculate the index. The problem is termed as ‘Small File Problem’ in Hadoop Terminology. The Hadoop Ditributed File system is designed to process a large amount of data. However, processing large number of small files seems inefficient, since Hadoop supports only block level operations. To overcome this problem, a ‘Balance Multi File Input Split Technique’ is proposed. Data is converted to bytes and collectively stored in ArrayWritable format. To avoid the need for separate indexing, we follow a hierarchical file naming & storing scheme. The method describes how to access the merged files through Map Reduce Programs. Analysis performed on BaMS proves that it is much more efficient compared to the existing methods like HAR and sequence files, in terms of storage and access efficiency.

i. Existing Methods to solve ‘Small File problem in Hadoop’

Processing small files significantly increase the overhead of Namenode, and adversely affect the performance of Map Reduce programs. There are several solutions [59-61] proposed in literature to solve this issue. Hadoop Archive Files (HAR) [52] compresses a lot of small files to form a separate layer over the HDFS. But reading HAR files require two disk accesses, i.e. to read the namenode address and to read the location of files within the HAR. The Federated Namenodes [53] approach involves multiple namenodes, where the subset of metadata is stored. But it does not change how the small files are stored inside each of these Namenodes. Sequence Files [54] feed the 'filename' as key, and the 'file content' as value, to a map reduce program. But it is also not efficient, because Hadoop supports only immutable, not appending operations. HBase [55] columnar database can be used to store frequently accessed file contents, but possess the overhead of installing the database over HDFS. Batch processing tools like PIG [56], HIVE [57] etc can be utilized, but does not solve the basic problem of small file storage. Amazon S3 storage [58] is preferable, but pre built tools will be needed, to access the data.

ii. Balanced Multifileinput Split (BAMS) Technique

To solve the problem of large number of small files in Hadoop, we adopt a software technique by exploiting the map reduce programming model. We consider files as the atomic unit of splitting. That is, no more splitting beyond files! CustomFileInputFormat available with Hadoop's org.apache.hadoop.mapred of Class MultiFileSplit, can be utilized to achieve this. The technique to process large number of small files in Hadoop using BaMS is described in Algorithm 8.1.

Initially, convert the entire input files into bytes format separately. The input files can be of any form, from text files to compressed file formats like Zip, GZip etc. The bytes are then wrapped to form a ByteWritable object, and a set of such objects merge to form an ArrayWritable object. To process the ArrayWritbale objects, use CustomeFileInput, where the granularity of splitting is a file. To feed input into the Mapper, use (NullWritable, ArrayWritable) as the (key, value) type. Here, NullWritbale is used to just fill the key position. It stores an immutable singleton value. Now, to process the files stored in ArrayWritable format, convert it back to text using get() method. The procedure is illustrated in figure 8.3.

Algorithm 8.1: Balanced MultiFileInput Split (BaMS)

Input: Large number of small files.

Output: Processed Files.

Steps:

1. Convert each file into bytes format.
2. Wrap the bytes obtained, into a ByteWritable object.
3. Multiple ByteWritables are combined to form ArrayWritable objects.
4. Repeat step 3 till the file size reaches default HDFS block size.
5. Use customFileInputFormat to process the ArrayWritable objects.
6. (NullWritable, ArrayWritable) is the key value pair input format, for each Mapper.
7. Restore the original data by converting the ArrayWritables to BytesWritbale inside the Mapper Logic.

iii. Retrieving Required Files using BaMS

BaMS processes a collection of files in ArrayWritable format. But in most of the applications (log processing, patient monitoring, market analysis etc), data need to be analyzed based on some criteria like time, date, period, etc. To make our algorithm efficiently handle such situations, we organize the files into a balanced hierarchical form, which helps to easily retrieve the specific files based on application requirement. A Balanced File Storage algorithm is provided by algorithm 2.

The procedure described in algorithm 8.2 helps to store files in a balanced and easy to retrieve form. Initially, name the files with the current date. For e.g., let us assume that the file is created on 23rd Jan 2016, then, the name is '23012016.txt'. Add leading zeroes to make the name 10 bit. That is the file name would be '0023012016'. In our application, we applied the hash function CRC-16, to get an output '10010011'. Split the name into 2 length tokens, '10/01/00/11'. So the file path will be 10/01/00/11/23012016.txt.

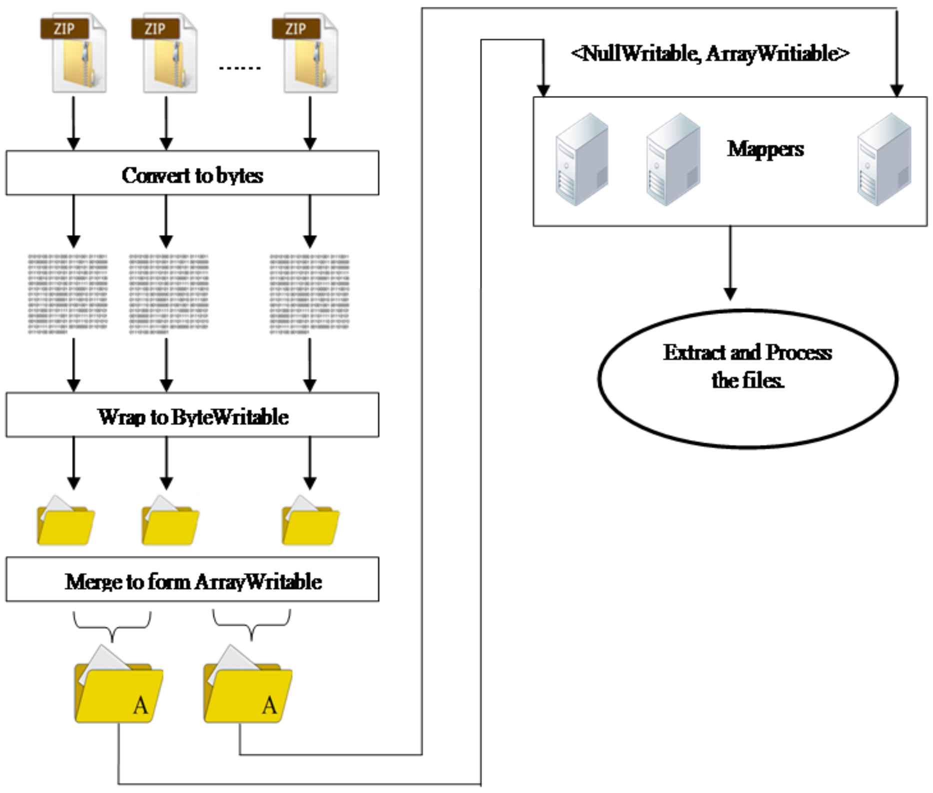

ZIP
ZIP
ZIP
......
Convert to bytes
Wrap to ByteWritable
Merge to form ArrayWritable
A
A
<NullWritable, ArrayWritiable>
Mappers
Extract and Process the files.

Figure 8.3 Balanced MultiFileInput Split (BaMS) Data Flow

Algorithm 8.2: Balanced File Storage for easy Retrieval

Input: Set of files to be retrieved, based on date.

Output: Balanced storage of files.

Steps:

1. Name the files with the current date, in (ddmmyyyy) format.
2. Pad it with leading zeroes to form a 10 bit number, and add the suffix.
3. Perform hash function on the name.
4. Divide the new name with two characters in each division. The tokens thus obtained indicate the directory structure.
5. Store the file under the generated directory.

iv. Analysis

The file stored will follow a balanced hierarchical structure. The hash function can be varied to MD5 or SHA, to increase the span of the directory structure. Since we used CRC-16, the output will always be in binary format. By splitting the hash value into tokens of length 2, we can store up to 65535000 files, if each directory is limited to 1000 files. We can vary the token length to increase or decrease the directory levels. Too many directory levels may waste disk space to store the directory structure itself. Also, it may result in holes. Hashing is applied to follow a balanced distribution. Even if we delete some files, it does not affect the directory structure, because we are performing the hash operation. Also, this type of storage will help easy retrieval of files of a specific time period. Suppose we need to analyze the logs for the month January 2016, create all possible dates within this month, apply hash function and obtain the directory level where the file is stored. Then, apply Algorithm 8.1 over the obtained files.

v. *Experimental Evaluation*

The application is tested on 10 node Hadoop cluster, setup on Amazon Web Service (AWS). The namenode is a t2.large instance. The secondary namenode and datanodes are t2.micro instances. All machines are Ubuntu 14.2 installed with OpenJDK 1.7, and the Hadoop version is 1.0.2. HDFS Replication factor is set to 3 and HDFS block size is set to 64MB.

BaMS technique is compared with the existing commonly used techniques, to avoid small file problems in Hadoop like HAR files and Sequencer (SQ). We are not taking ‘Consolidator’ approach, because it is found to be the least efficient and practical method [48]. Also, HBase is not considered, because it needs an additional database installed over HDFS. HBase is commonly used for streaming data analysis.

vi. DataSet

The dataset used for testing is Thomson Reuters Text Research Collection (TRC2). The dataset contains 1,800,370 stories, which occurred between a period 01-01-2008 00:00:03 to 28-02-2009 23:54:14. The size of the dataset is 2,871,075,221 bytes. TRC2 is a single long file with date, headlines and stories stored in a comma separated form. To match our testing requirements, we split this large file into multiple files, where each file is named with date in ddmmyyy.txt format, and the content of that file is the headlines & stories on that particular day. Thus 419 files are generated, where each file size ranges from 8MB to 16MB. Thus we created a large number of small files.

vii. Comparison of HAR, SQ and BaMS on storage efficiency:

The memory required by the Namenode to store small files in HAR, SQ and BaMS is compared. The Namenode storage size reduction is highly crucial for a Hadoop cluster. The memory of a Namenode is exhausted by the storage of metadata and block details of huge number of files. Thus, a large number of small files will limit the scalability of Hadoop cluster, and lead to the failure of the cluster, as Namenode is the single point of failure. The File Number Per KB of Memory of Namenode, (FNPKMN) [49] is an important measure to determine the storage efficiency of Namenode. FNPKMN is the ratio of number of files stored in HDFS to the memory required by Namenode.

$$FNPKMN = N / M_{NN}$$

Where, N is the number of files stored in HDFS and M_{NN} implies the memory required by Namenode to store the details of N files.

Due to the archiving capability, HAR and BaMS show maximum memory efficiency. Figure 8.4 compares the storage efficiency of different schemes. Also, in case of HAR and BaMS, for comparison, we deleted the original files after compression. From the graph it is clear that, the HAR and BaMS methods strictly increase the storage efficiency by a minimum of 42% and the efficiency increases as datasize increases. Also, the BaMS method maintains the consistency in storage efficiency, irrespective of the data sizes.

Figure 8.4.b illustrates how the storage efficiency varies, as the replication factor is varied. We tested for replication factor 2, 4 and 6, keeping 8 datanodes and dataset size 2GB to 5GB. The graph shows exponential decrease in FNPKMN as the replication factor doubles, for Native Hadoop and Sequencer method. The BaMS shows minimum improvement of 12% compared to the existing well known method, HAR.

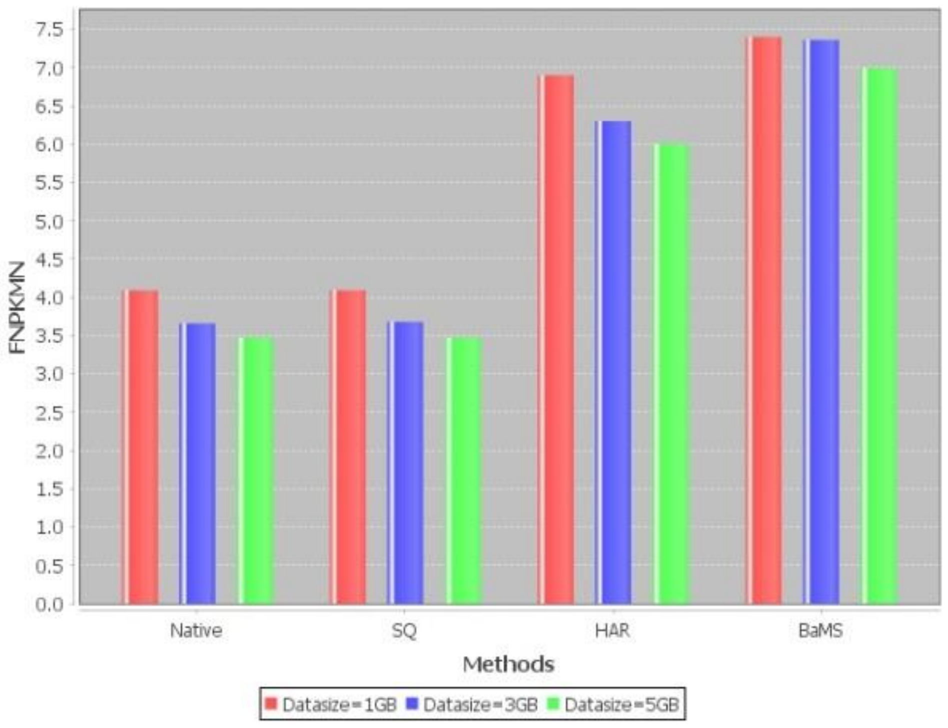
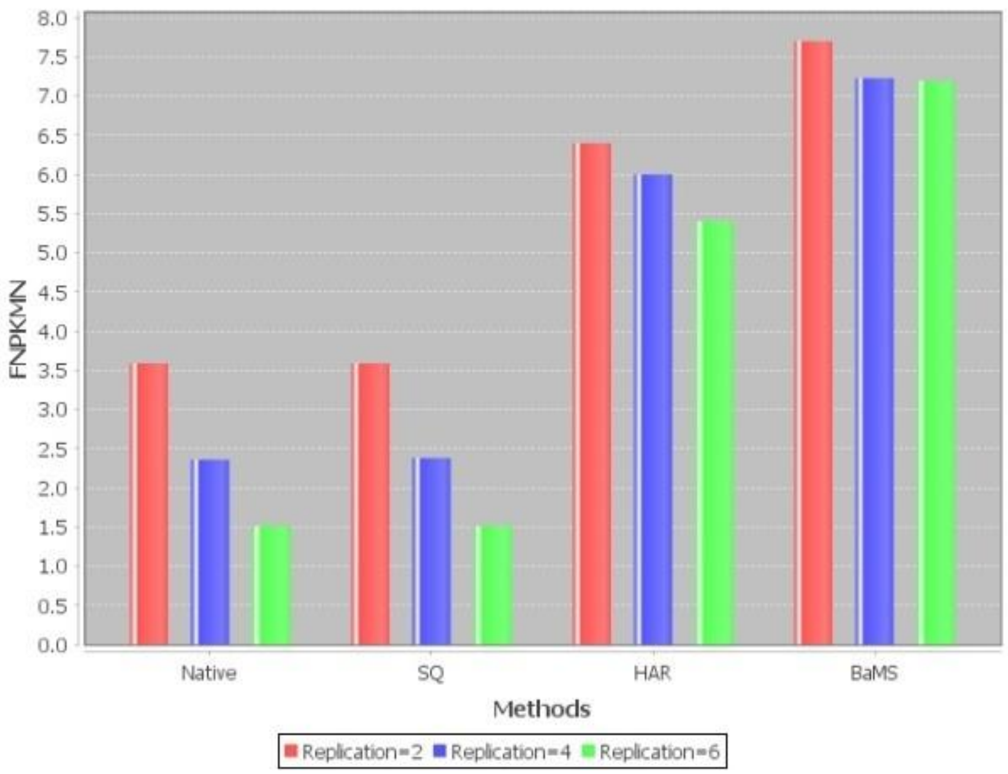


Figure 8.4 Comparison of FNPKMN a) Varying Datasize



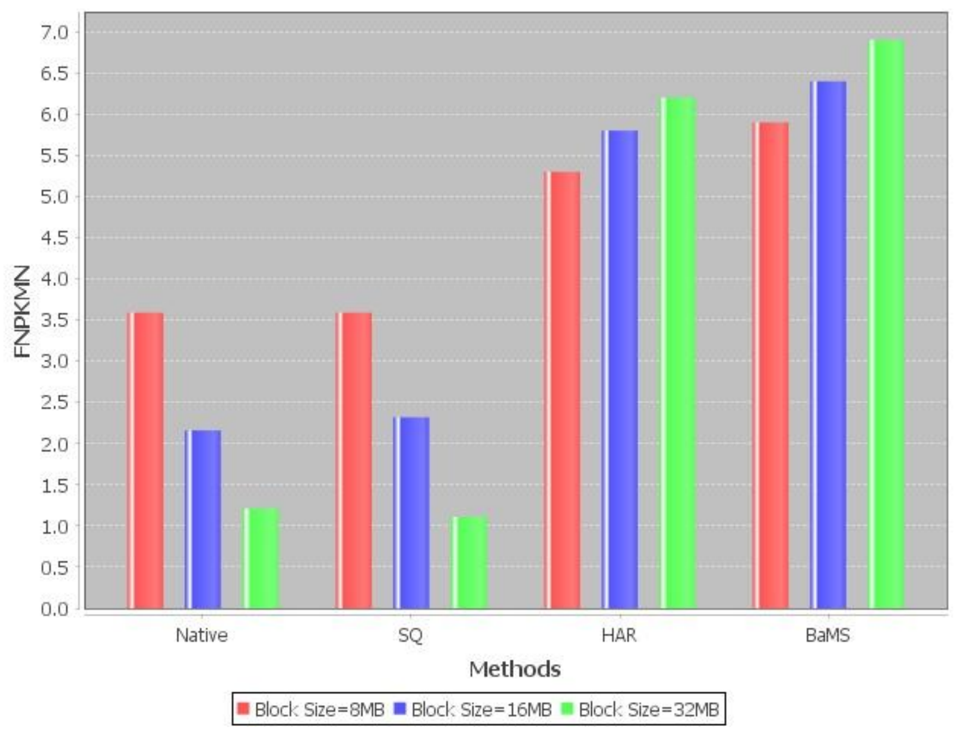


Figure 8.4: Comparison of FNPKMN a) Varying Datasize b) Varying Replication Factor c) Varying Block size

Figure 8.4.c illustrates the variation in storage efficiency as the default block size is varied. It can be noticed that, as the block size increases, HAR and BaMS show higher efficiency. In the native Hadoop as well as the Sequencer method, increasing the block size will lead to more holes in the memory. Hence, it is evident from the graph that, as block size increases, the storage efficiency is steadily decreasing in them. But in HAR and BaMS, compression is done till default block size is reached. Hence, a higher block size implies higher storage efficiency.

viii. Comparison of HAR, SQ and BaMS on Access Efficiency

Access efficiency is measured using the Millisecond Per Accessing a File (MSPF) [50] metric. MSPF calculates the average time needed to access files in the benchmarked datasets.

$$MSPF = T/N$$

Where, T is the time taken to access the files and N implies the number of files.

Figure 8.5 reveals that MSPF is inversely proportional to the block sizes. As the block size increases, MSPF reduces. But BaMS and HAR shows much improvement in access time, compared to the uncompressed method like Sequencer.

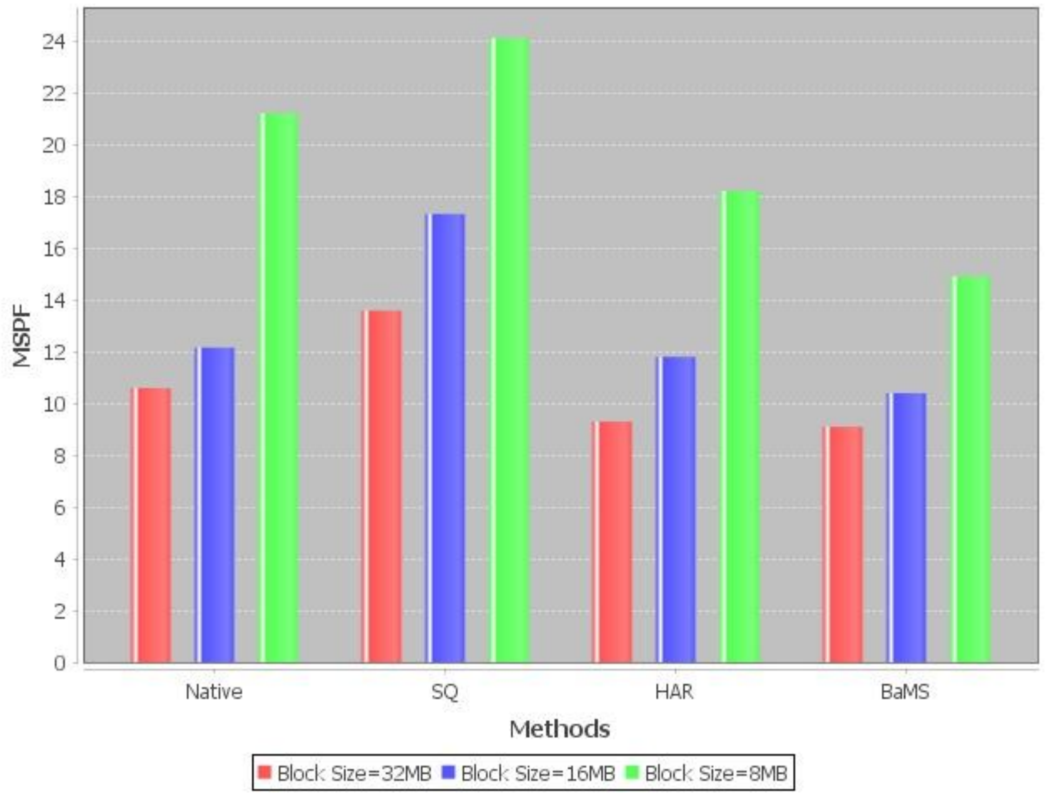


Figure 8.5 MSPF Comparison

ix. SpeedUp of BaMS

Another important parameter that should be analyzed is the Speed Up. Speedup implies the ratio of time required to execute a program, parallel to the time required to execute it sequentially. Speed Up can be represented as

$S_u = T_n / T_1$, where T_n is the time taken to execute on n number of machines and T_1 is the time required to execute on a single machine.

It is desirable to have linear speed up values, to ensure good scalability of the underlying approach.

To measure the speed up, we analyzed the time taken to execute an inverted index creation for the entire dataset, by varying the number of datanodes. It is observed from figure 8.6 that our method shows almost linear speedup for varying datanodes, Also, as data size increases, the speedup becomes more linear, which implies higher scalability.

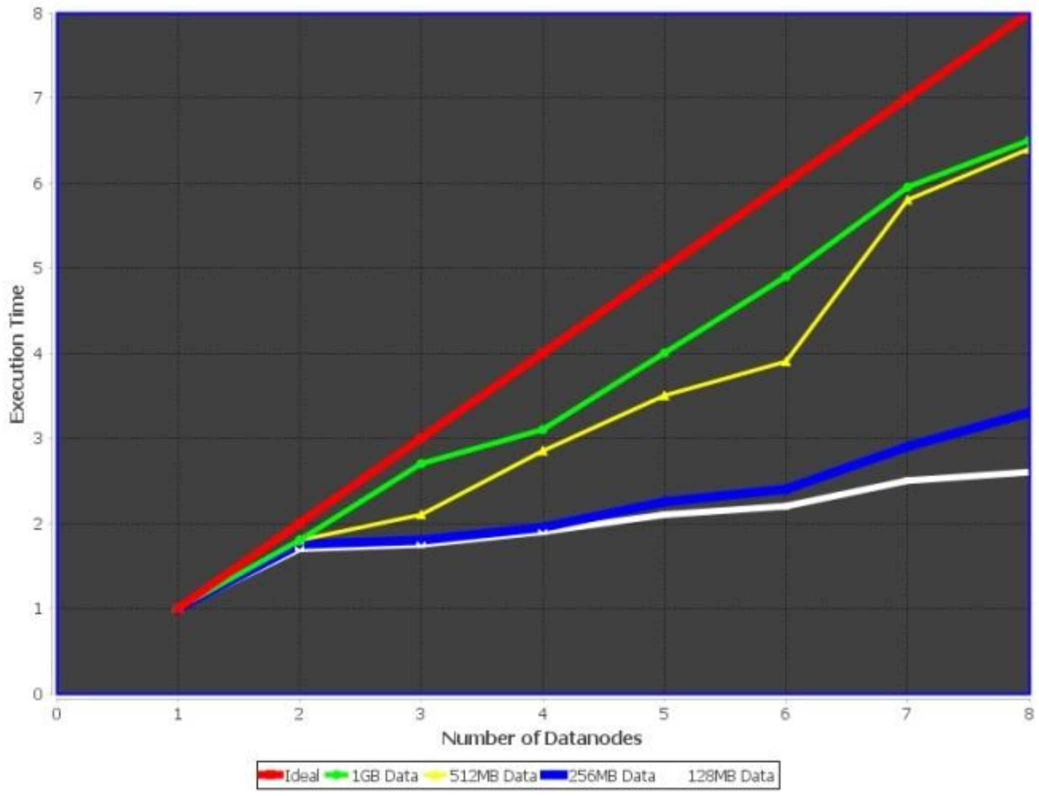


Figure 8.6 SpeedUp Comparison

Challenge 2: Reduce the communication overhead

Log analysis is usually done to detect some anomalies or threats. If these anomalies can be identified from the encrypted logs itself, there is no need for a second round of communication. For example, to find the brute-force attempts, the primary data needed is the total number of login attempts from a source IP. Since the proposed MHE scheme is additive homomorphic, the cloud server can return the encrypted sum of total attempts. The value can be decrypted at the client side, to find the number of login attempts from each IP. Here, there is no need for a second round of communication.

Similarly, to find the traffic outlier, to detect DDoS attacks etc, the sum of bytes transferred from a network is what is needed. This can also be calculated in a single round. Thus, by efficient storage of logs as per retrieval needs, the communication overhead of the proposed scheme can be reduced.

Summary:

This chapter illustrates the two significant application scenarios, where the proposed MHE scheme can be efficiently applied. The challenges and design details are also discussed. To deal with the ‘Small File Problem’ in Hadoop, Balanced MultiFile Input Split is also proposed. The chapter discusses how to feed input to Map Reduce programs using BaMS technique, and the analysis performed on the approach proves that it is efficient in terms of storage, access and speedup. Secure Email server need mails to be stored in encrypted format and retrieve relevant mails based on keywords issued. Secure log storage need storage of data in encrypted form and retrieval of data after applying some operations.

Chapter 9

Conclusion

"Whether you think you can, or that you can't, you are usually right."

- Henry Ford

This chapter summarizes the entire research, discussing the limitations of the work as well as shredding light on some future directions in this research area.

9.1 Secure and Privacy preserving search

Today, as internet and social media has taken over the control of the day to day activities of each person, the privacy of a person and the security of their personal information has become a severe issue. Even though a lot of personal information leakage and privacy breaches are being reported, the true impact is identified only when a person is genuinely affected by that. By the time a privacy breach is identified, it might be too late to apply some rectification measures. On 5th April 2018, the official report came out from Facebook [64] confirming that half a million Indian users who have installed their app, might have become victims of data breaches. The situation becomes more critical, when the data that is stolen contains some sensitive information like data of national importance, government data, healthcare data, organization data etc. An exclusive report [65] sent out by ZDNet on 28th May 2018, says that even without enrolling to our national database on Aadhar, companies like Amazon or Uber can download the entire personal information of all the 11 billion users, who have registered with the database. Even though the report has not been officially confirmed by any government officials, the threat and impact is significant.

Apart from security of the stored data, the privacy of the user who searches for any particular information is another factor to be considered seriously. For example, if a person searches for the keyword “secret mission” and retrieves some file, it implies that the person is associated with some secret mission or he is in need of information regarding a secret mission. Thus, the queries issued by users can be used to obtain information about them, leading to privacy breaches. It should be kept in mind here that security and privacy are very much critical for today’s digital world. The research primarily addressed this problem of secure and privacy preserving searches that can be applied over sensitive information.

To ensure security of the data stored in third party systems, encryption is utilized. Searching is made possible within this encrypted domain, by creating and uploading a secure and searchable index along with the encrypted data, to the third party system. To implement ranked information retrieval from the stored encrypted data, well known techniques in information retrieval like the TF-IDF and vector space model, are utilized.

Also, since data is evolving at a huge rate, the method that we proposed is accelerated by adopting distributed processing strategies.

9.2 Modified Homomorphic Encryption (MHE)

The research proposed a fully homomorphic encryption scheme, deriving insights from the Craig Gentry's integer based bootstrappable FHE scheme. Currently, the scheme supports integer operations at the cost of higher execution steps and space complexity. We aimed to reduce the time and space complexity, by modifying the existing Gentry's scheme in such a way so as to make it applicable for secure information retrieval. The TF-IDF values are scaled to a specified range so that always the MHE needs to be applied only to a specified set of values. Thus, my research modified the original scheme to suit the secure and privacy preserving search. The scheme is additive and multiplicative homomorphic. Security analysis done on the MHE scheme proves that the scheme is secure against many known attacks like Bruteforce, fraction attack and Howgrawe Graham's GCD attack.

9.3 Dual Round Encrypted Information Retrieval (DRIER)

Ranked information retrieval needs to find a score associated with each file, based on the degree of similarity with the query keywords. To implement a scoring scheme, the vector space model with TF-IDF values is combined. The index is encrypted using MHE and uploaded. As a user issues a query, a bit pattern of zeroes and ones are made, which is then encrypted using MHE. Multiplication and summation operation is performed on each encrypted score of the words in a file, and the encrypted score associated with each file is identified. The cloud server will return this encrypted score to the client. At the client side, the decryption and ranking of the score is performed, to identify the top K matching documents. The corresponding File IDs are then sent to the server, and the server in turn, returns the encrypted files. The files are then decrypted at client side. Thus, a two round communication is initiated between the client and the server, to retrieve the most matching documents from the server. The security analysis done on the scheme proves that it is secure and privacy preserving. As the documents are encrypted and stored in the cloud, data is not available in raw form. Only authorized users gain access to the contents. As the index is stored in an encrypted form, any statistical leakages or guessing attacks are prevented. The words and file names are deleted from the index. The queries are encrypted and send to the cloud. Hence, the privacy of the user is preserved. Multiple queries issued with even the same keywords will result in different results, thus avoiding any guessing attacks.

9.4 Accelerating MHE using Map Reduce Programming Model

One of the drawbacks associated with the DRIER scheme is the execution time needed for computing a secure index, for large data. Also, the retrieval stage should be made faster, as this should be repeated several times. The advantage of the proposed scheme is that, all the operations can be parallelized and executed. So, the power of Hadoop with MapReduce programming model is utilized to make the uploading and retrieval stages scalable and fast. Map Reduce always works on <key,value> pairs. The DRIER document indexing and retrieval stages are modified to match this <key, value> type of input and output parameters. The analysis done on the Hadoop cluster hosted on Amazon Web Services, demonstrated better scalability and faster execution.

9.5 Overcoming 'Small File Problem' in Hadoop

Most of the applications like logs, emails etc need to index a large amount of small files. But, this causes some overhead to Hadoop, as the files are smaller than their default block size. To overcome this limitation, the Balanced MultiFileInput Split technique is proposed. The scheme utilizes the inherent data structures available in Map Reduce framework like Array Writable, ByteWritable etc, to process data efficiently.

9.6 Merits of the proposed Secure and Privacy preserving information retrieval

1. Security of the data is ensured.
2. User's privacy is preserved.
3. Multiple keywords can be issued.
4. Ranking based on similarity of the keywords in queries.

5. Resistant to Statistical leakages and Term distribution/inter-distribution attacks.

6. Third party data storing server cannot see the file contents, name of files, words present in files, keywords issued by user, original similarity score or any other details that could initiate some sort of guessing attacks.

7. The method can be applied to store large amount of data as it has been proven scalable.

8. The scheme proposed a BaMS technique to process large amount of small files, which is very critical for the processing of real time logs, outputs of each Map Reduce stages, logs, emails, reports, transaction details etc.

9.7 Limitations of the proposed scheme

1. Needs two rounds of communication between the client and the server, thus causing a communication overhead.

2. Along with the encrypted files, a secure and searchable index should be stored, which causes a space overhead.

9.8 Applications of the proposed scheme

1. Secure Email storage.

2. Secure Log storage.

3. Secure storage of data of national importance like Unique Identification, surveillance data, government reports etc.

4. Secure storage and sharing of Health records.

5. Electronic Voting.

9.9 Future Directions of our Research

1. The scheme can be extended in such a way so as to suit more applications, by altering the design of storage and retrieval. For example, to make the scheme applicable to

Electronic Voting, there is no need of information retrieval based on keywords. The MHE scheme can be directly applied to encrypt the votes. The total votes can be counted by taking the sum of encrypted votes, as the scheme is additive homomorphic.

2. A separate area of research can be initiated on the encrypted operations that can be applied using MHE. For example, in a secure log storage application, how to identify the networks that deviate from the mean of traffic utilization compared to others, how to identify the people who logged in from different geographic locations within a small time frame etc. Apart from the MHE scheme, some geo hashing techniques can also be incorporated for better analysis.

3. The research utilized Term Frequency – Inverse Document frequency for identifying similar documents to a keyword. Apart from this, if the meaning of the words can also be taken into consideration, it will improve the ranking a lot. For example, if a person searches for “apple phone”, here, apple is more associated to a product than a fruit. If this information can be included in such a way that the security is not much compromised, the application gains a lot more credibility.

4. Extend the encrypted operating capability of the server such that the operations required can be performed in a single round of communication.

Summary

This chapter enclosed in a nut shell, the entire work done as part of the research. Each stages of the research have been summarized following the merits and limitations of the scheme. Some of the significant application scenarios have also been provided. The chapter concludes by mentioning some of the future enhancements of the research.

APPENDIX 1

Research Publications

International Journal/Article Publications

1. Lija Mohan, Sudheep Elayidom M., "Secure and privacy-preserving multi-keyword ranked information retrieval from encrypted big data", Accepted for publication in International Journal of Information and Computer Security(IJICS), INDERSCIENCE Publishers.
http://www.inderscience.com/info/ingeneral/forthcoming.php?jcode=ijics

2. Lija Mohan, Sudheep Elayidom M., "Encrypted Data Searching Techniques And Approaches For Cloud Computing - A Survey", SPRINGER Advances in Intelligent Systems and Computing series, AISC, volume 458, pp 97-104, 23rd November 2016.
https://link.springer.com/chapter/10.1007/978-981-10-2035-3_11

3. Lija Mohan, Sudheep Elayidom M., "Fine grained Access Control in Secure Cloud Environment- a polynomial based approach", ELSEVIAR Procedia for Computer Science,
Volume 46, Pages 719-724, April 2015.
http://www.sciencedirect.com/science/article/pii/S1877050915001970

4. Lija Mohan, Sudheep Elayidom M., "Secure and Privacy Preserving Mail Servers using Modified Homomorphic Encryption (MHE) Scheme",International Journal of Advanced Computer Science and Applications (IJACSA), pages 53-65, Volume 9, Issue 3, March 2018.

5. Lija Mohan, Sudheep Elayidom M., "Predicting the Winner of Delhi Assembly Election, 2015 from Sentiment Analysis on Twitter Data – A BigData perspective", Accepted for publication in International Arab Journal of Information Technology (IAJIT).

6. Lija Mohan, Sudheep Elayidom M., 'Collective Tweet Analysis for Accurate User Sentiment Analysis - a Case Study with Delhi Assembly Election 2015', Accepted for publication in International Journal of Big Data Intelligence (IJBDI), INDERSCIENCEPublishers.
http://www.inderscience.com/info/ingeneral/forthcoming.php?jcode=IJBDI

7. Lija Mohan, Sudheep Elayidom M., "A Practical Scheme for Implementing Client Side Encryptions and Decryptions", International Journal of Information Technology and Computer Science (IJITCS), MECS Publishers DOI: 10.5815,pp 81-90 Volume 9,2016.

Book Chapters

1. Secure Distributed Data Storage in Cloud – Methods and Practices & Implementation and Testing of Map Reduce Programming Model in Cloud – A case study with Amazon EC2 in a text book to be published by Cengage Publishers

2. Classification Models and Practical Data Mining Tools, in 'Data Mining and Warehousing' text book published by Dr. Sudheep Elayidom under Cengage Publishers on 2014.

International Conferences

1. Lija Mohan, Sudheep Elayidom M., "Balanced MultiFileInput Split (BaMS) Technique to solve Small File Problem in Hadoop", IEEE International Conference on Industrial and Information Systems (ICIIS), IIT Roorkee, 20th December, 2016 (Available in IEEEXplore).

2. Lija Mohan, Sudheep Elayidom M., "Hadoop – A Big Data Analytic Tool", International Conference On Recent Trends In Engineering & Technology, ICRTET, 8th December, 2014.

Research Symposium

1. Lija Mohan, Sudheep Elayidom M., Research Proposal on "Secure and Efficient Mining over Large Scale Data using Map Reduce programming model" presented in Ph.D forum at IEEE sponsored International Conference on Communication Systems and Networks (COMSNETS), Bangalore, 28th January 2015.

2. Lija Mohan, Sudheep Elayidom M., "Big Data Analytics for Social Wellness – Implementing Twitter based customer recommendation for banks", ACM IRISS Research Symposium, Trivandrum on 8th January, 2016.

National Conference

1. Lija Mohan, Sudheep Elayidom M., "Who Wins Delhi Election – Prediction by HIVE", National Conference on Adaptive Techniques in Engineering and Technology (NCATET), Thrissur, 13th April, 2015.

APPENDIX 2

Time based Collective User Sentiment Analysis for Twitter Data

Overview

Analyzing social media contents will excavate knowledge treasures in the form of customer behavior, feedback, suggestions, opinions etc which could be utilized for business intelligence. Even though, sentiment analysis from social media is a severely explored domain, this part of the research studies the importance of “Time of posting the contents”. There are several real world applications where mere sentiment analysis is not sufficient and ‘time’ at which that content is posted should be given a weightage. For eg., to compare two products or to do trend analysis, recent posts should be given more weightage. The authors introduce a novel and scalable algorithm to introduce the time factor to improve the accuracy of sentiment analysis. To support Big Data, Hadoop Map Reduce based implementation is provided. To prove the efficiency of the method, Delhi Assembly Election Winner Prediction by Twitter Analysis is taken as a case study. The results prove that, the algorithm is accurate, scalable and time efficient as compared to the existing ones.

Steps in Implementation

1. Building a sample sentiment domain thesaurus for tweet classification

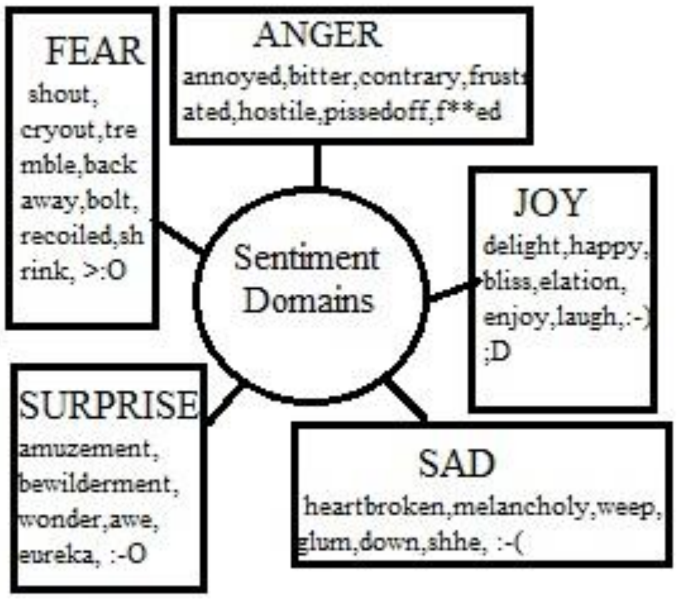

FEAR
shout, cryout,tremble,back away,bolt, recoiled,shrink, >:O
ANGER
annoyed,bitter,contrary,frustrated,hostile,pissedoff,f**ed
JOY
delight,happy, bliss,elation, enjoy,laugh,:-) ;D
Sentiment Domains
SURPRISE
amuzement, bewilderment, wonder,awe, eureka, :-O
SAD
heartbroken,melancholy,weep, glum,down,shhe, :-(

Figure A2.1: Sample Sentiment domain thesaurus for Tweet Classification

2. Cluster tweets from different users. Find the sentiment class of each tweet based on TF-IDF

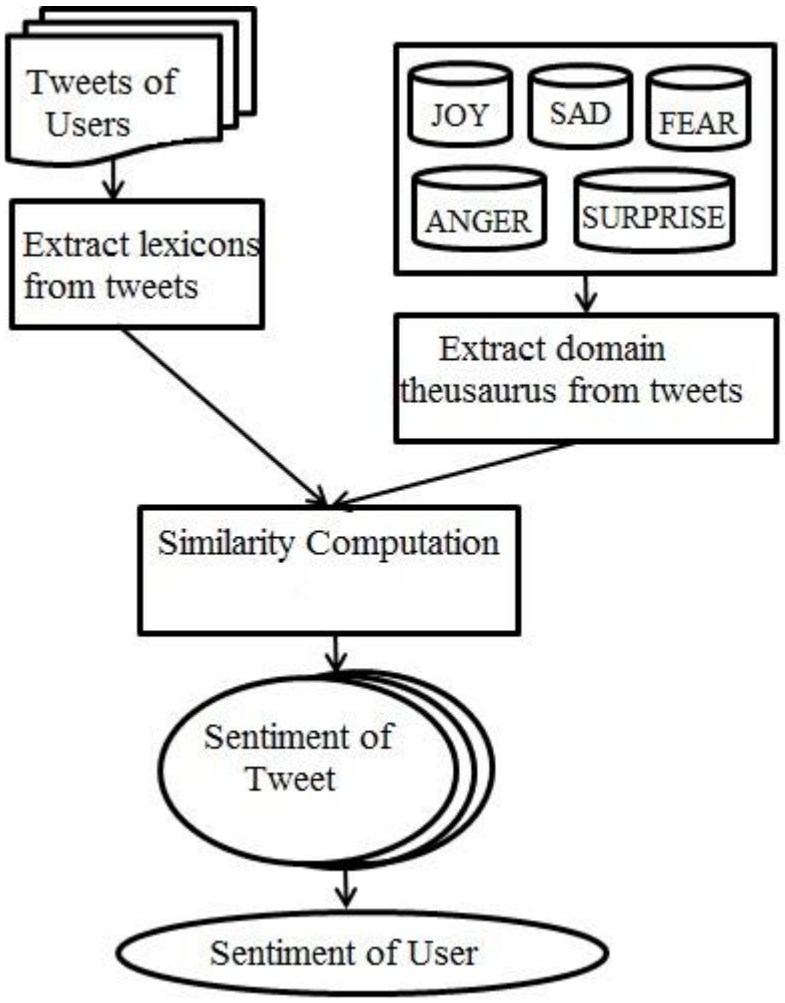


Figure A2.2: Overall Data Flow in tweet sentiment Classification of each User

3. Apply Map Reduce programming model to improve the execution time and ensure scalability

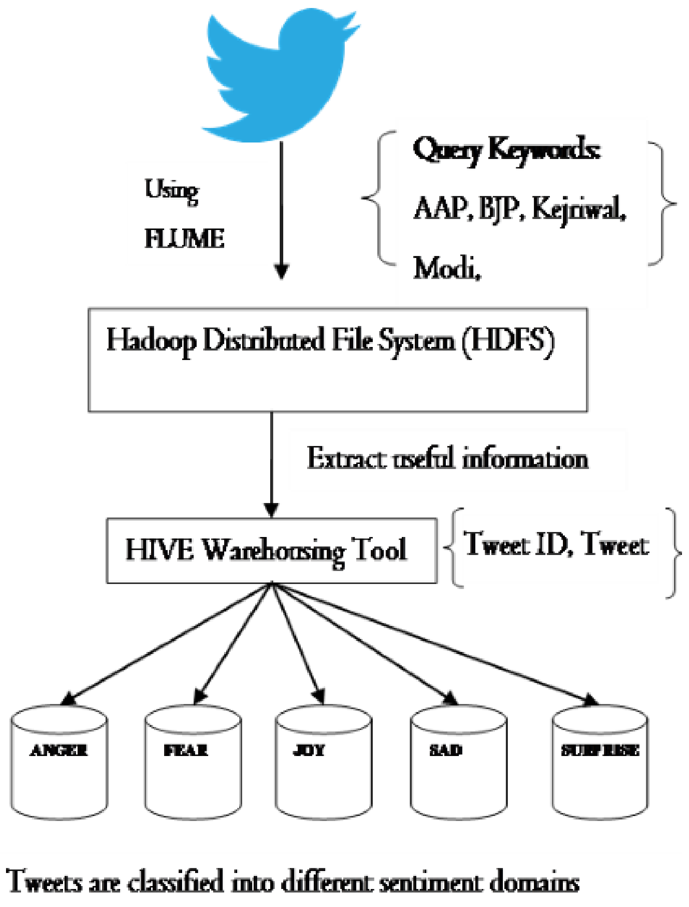

Using FLUME
Query Keywords: AAP, BJP, Kejriwal, Modi,
Hadoop Distributed File System (HDFS)
Extract useful information
HIVE Warehousing Tool
Tweet ID, Tweet
ANGER
FEAR
JOY
SAD
SURPRISE
Tweets are classified into different sentiment domains

Figure A2.3: Information Flow in the Classification of Tweets

4. Assign weightage to tweets based on time.

Algorithm A.1**:** Sentiment Classification giving weightage to latest tweets

Input: Lexicons extracted from tweets, T_k

Sentiment Domains SD ={sd_1, sd_2, ..., sd_N} , The threshold δ in the filtering phase, The number *K*

T_l = last time of posting a tweet

T_i = Initial time of posting a tweet

Output: Identified sentiments with the Top-K highest similarity {tsd_1,tsd_2, ..., tsd_K}

```
1: for each sentiment $sd_i$ Ɛ $SD$
    2: $Q$=ϕ, $count$=0, $r$=0, λ= $T_l - T_i$
    3: for each feature word $R_j$ of sentiment $sd_i$
        4: process the tweet into a probable keyword set $PK_j$
        5: if $PK_j \cap T_K \neq \phi$ then
            6: insert $PK_j$ into $Q$
        7: end if
    8: end for
    9: for each keyword set $PK_j \sum Q$
        10: $similarity\_score = SIM(T_K, PK_j)$
        11: if $similarity\_score \leq \delta$ then
            12: remove $PK_j$ from $Q$
            13: else $count = count + 1, r = r + r_j$
        14: end if
    15: end for
    16: $r^* = r / count$
    17: $p_s = r^* + k \sum_{PKj \sum Q} SIM(P_K, T_K). (r_j - r^*)$
    18. $p_t = \lambda * p_s$
19: end for
20: sort the sentiment domains according to the personalized sentiment matching
    with weightage to time, $p_t$
21: return the sentiment domains with the Top-K highest matching {$tsd_1$,$tsd_2$, …, $tsd_K$}
```

5. *Find the aggregated sum of support giving weightage to latest tweets.*

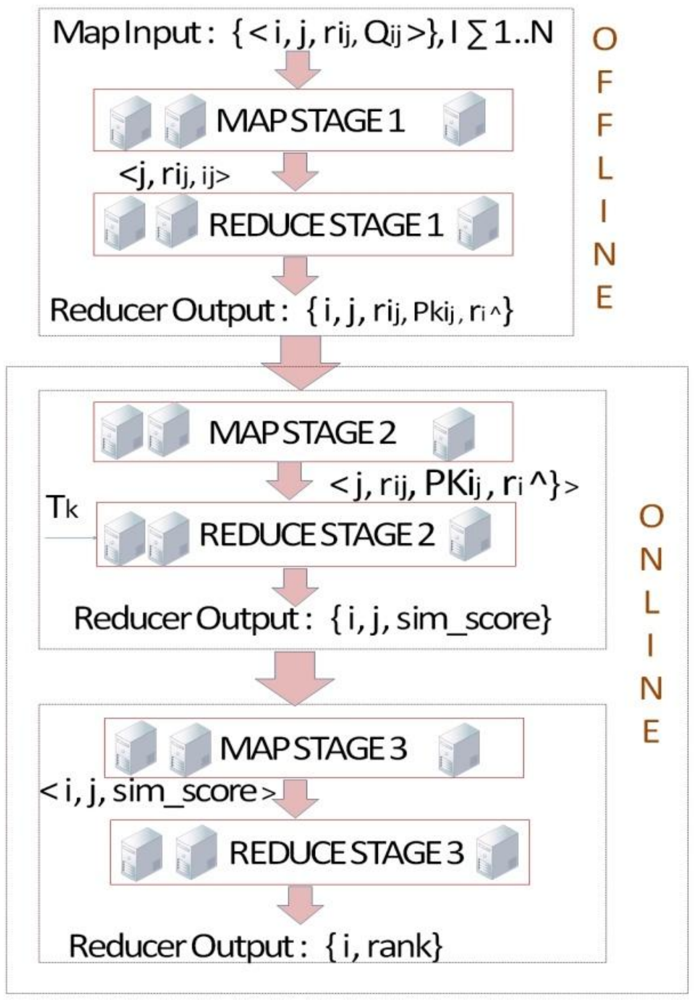

Map Input : {< i, j, rij, Qij >}, I ∑ 1..N
MAP STAGE 1
<j, rij, ij>
REDUCE STAGE 1
Reducer Output : { i, j, rij, Pkij , ri ^}
OFFLINE
MAP STAGE 2
< j, rij, PKij , ri ^}>
Tk
REDUCE STAGE 2
Reducer Output : { i, j, sim_score}
ONLINE
MAP STAGE 3
< i, j, sim_score >
REDUCE STAGE 3
Reducer Output : { i, rank}

Figure A2.4: MapReduce Implementation Stages

Summary

From the results, it was inferred that, the newly formed party named ‘Aam Aadmi Party (AAP)’ had higher support among people compared to the well established party like “Bharathiya Janatha Party (BJP)”. By altering the domain thesaurus, this method can be adopted for multiple applications like product marketing based on customer feedback, spam detection, deal recommendations, collecting people’s feedback on political reforms, and potential customer identification in agricultural sector.

APPENDIX 3

HELib Code for Modified Homomorphic Encryption

This source code is a modified version of FHE that shows how to fills two vectors with integrals, encrypts them homomorphically, and performs component-wise addition and multiplication. This process is actually more involved, the real steps we will go through are: declare our parameters (plaintext space, levels, columns, secret key hamming weight, security), generate a secret key, obtain an EncryptedArray, which is a class that aids in later computations, encrypt our vectors, perform addition and multiplication, decrypt the results and print. The code is based on one of HELibs 'Test_*.cpp' examples.

```
#include "FHE.h"
#include "EncryptedArray.h"
#include <NTL/lzz_pXFactoring.h>
#include <fstream>
#include <sstream>
#include <sys/time.h>

int main(int argc, char **argv)
{
   // On our trusted system we generate a new key (or read one in) and encrypt the secret
data set.

   long m=0, p=2, r=1; // Native plaintext space. Computations will be 'modulo p'
   long L=16;          // Levels
   long c=3;           // Columns in key switching matrix
   long w=64;          // Hamming weight of secret key
   long d=0;
   long security = 128;
   ZZX G;
   m = FindM(security,L,c,p, d, 0, 0);

/*The parameter names are pretty consistent in HELib examples as well as the literature.
In this case, I am building for GF(2) - so my homormorphic addition is XOR and
multiplication is AND. Changing this is as easy as changing the value of p. To
perform 2+2=4, set p to something that matches their desired domain, such as 257 to
obtain 8 bit integer.*/

   FHEcontext context(m, p, r);     // initialize context
   buildModChain(context, L, c);     // modify the context, adding primes to the modulus
chain
   FHESecKey secretKey(context);     // construct a secret key structure
   const FHEPubKey& publicKey = secretKey;    /* an "upcast": FHESecKey is a
subclass of FHEPubKey */
```

```
              G = context.alMod.getFactorsOverZZ()[0];
               secretKey.GenSecKey(w);
   // actually generate a secret key with Hamming weight w

                 addSome1DMatrices(secretKey);
               cout << "Generated key" << endl;
```

We have now generated a secret key. Notice the public key was extracted from the private key. Interestingly, a public key in this context need only be an encryption of "1".

Instantiating Helper class:

```
   EncryptedArray ea(context, G);   /* constuct an Encrypted array object ea that is
associated with the given context and the polynomial G */

   long nslots = ea.size();
```

For encryption:

```
   vector<long> v1;
   for(int i = 0 ; i < nslots; i++) {
       v1.push_back(i*2);
   }
   Ctxt ct1(publicKey);
   ea.encrypt(ct1, publicKey, v1);

   vector<long> v2;
   Ctxt ct2(publicKey);
   for(int i = 0 ; i < nslots; i++) {
       v2.push_back(i*3);
   }
   ea.encrypt(ct2, publicKey, v2);

   // On the public (untrusted) system we can now perform our computation

   Ctxt ctSum = ct1;
   Ctxt ctProd = ct1;

   ctSum += ct2;
   ctProd *= ct2;
```

```
// Finally, decrypt the sum and product results:

    vector<long> res;
    ea.decrypt(ctSum, secretKey, res);

    cout << "All computations are modulo " << p << "." << endl;
    for(int i = 0; i < res.size(); i ++) {
        cout << v1[i] << " + " << v2[i] << " = " << res[i] << endl;
    }

    ea.decrypt(ctProd, secretKey, res);
    for(int i = 0; i < res.size(); i ++) {
        cout << v1[i] << " * " << v2[i] << " = " << res[i] << endl;
    }

    return 0;
}
```

REFERENCES

[1] Cloud Security Alliance, “Top Threats to Cloud Computing,” http://www.cloudsecurity alliance.org, 2010.

[2] “The NSA Scandal’s Impact on the Future of Cloud Security - Data Security and Protection in the Wake of the Spying Scandal”, White Paper issued by Porticor, February, 2017.http://mobility-sp.com/images/gallery/PORTICOR-The-NSA-Scandal%27s-Impact-on-the-Future-of-Cloud-Security.pdf

[3] Guidelines on Security and Privacy in Public Cloud Computing, NIST Special Publication 800-144, December 2011.

[4] Dawn Xiaodong Song, David Wagner, Adrian Perrig,“Practical Techniques for Searches on Encrypted Data”,Proceedings of the 2000 IEEE Symposium on Security and Privacy, ISSN 1081-6011, Page 44 , 14th May 2000, Berkeley, CA, USA.

[5] R. Curtmola, “Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions,” Proceedings of ACM Conference on Computer and Communications Security (CCS), ISBN:1-59593-518-5, Page 79–88, 3rd November 2006, Alexandria, Virginia, USA.

[6] S. Zerr, D. Olmedilla, W. Nejdl, and W. Siberski, “Zerber+r: Top-k Retrieval from a Confidential Index,” Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology (EDBT), ISBN: 978-1-60558-422-5, Pages 439-449 , March 24 – 26,2009, Saint Petersburg, Russia.

[7] Jiadi Yu, Peng Lu, Yanmin Zhu, Guangtao Xue, and Minglu Li, “Toward Secure Multikeyword Top-k Retrieval over Encrypted Cloud Data”, IEEE Transactions On Dependable And Secure Computing, ISBN: 978-1-60558-506-2, Pages 169-178, Vol. 10, Issue. 4, July/August 2013.

[8] C. Gentry, “Fully Homomorphic Encryption Using Ideal Lattices,” Proceedings of the 41st Annual ACM Symposium on Theory of computing (STOC), ISSN 1545-5971 ,pages 169-178, Vol. 10, Issue 4,2009.

[9] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, “Fully Homomorphic Encryption over the Integers,” Proc. 29th Ann. International Conference on Theory and Applications of Cryptographic Techniques, ISBN 978-3-642-13189-9, vol 6110,pages 24-43, 3rd December,2010, Springer, Berlin, Heidelberg.

[10] C. Leslie, “NSA Has Massive Database of Americans’ Phone Calls,” http://usatoday30.usatoday.com/news/washington/2006-05-10/, 2013.

[11] R. Curtmola, J.A. Garay, S. Kamara, and R. Ostrovsky, “Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions,” Proc. ACM 13th Conf. Computer and Comm. Security (CCS),ISBN 1-59593-518-5, Pages 79-88,15th November 2006, Virginia, USA.

[12] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, “Secure Ranked Keyword Search over Encrypted Cloud Data”, Proc. IEEE 30th International Conference on Distributed Computing Systems (ICDCS), ISBN 978-1-4244-7262-8, 25th June 2010, Genova, Italy.

[13] S. Zerr, D. Olmedilla, W. Nejdl, and W. Siberski, “Zerber+r: Top-k Retrieval from a Confidential Index,” Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology (EDBT), ISBN: 978-1-60558-422-5, Pages 439-449 , March 24 – 26,2009, Saint Petersburg, Russia.

[14] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, “Fully Homomorphic Encryption over the Integers,” Proc. 29th Ann. Int’l Conf. Theory and Applications of Cryptographic Techniques, ISBN 978-3-642-13190-5, pages 24-43, vol 6110. Springer, 2010, Berlin, Heidelberg.

[15] M. Perc, “Evolution of the Most Common English Words and Phrases over the Centuries,” J. Royal Soc. Interface, 2012.

[16] O. Regev, “New Lattice-Based Cryptographic Constructions,” Journal of ACM, vol. 51, no. 6, pp. 899-942, 2004.

[17] Mihir Bellare, Thomas Ristenpart, Phillip Rogaway, and Till Stegers. “Format-preserving encryption”,Lecture Notes in Computer Science, ISSN:0004-5411, pages 295–312.vol. 5867, Springer, 2009, New York, NY, USA .

[18] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O’Neill. Order-preserving symmetric encryption. EUROCRYPT, vol. 5479 of Lecture Notes in Computer Science, ISSN:0004-5411, pages 224–241. Springer, 2009.

[19] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam ONeill, “Order-preserving symmetric encryption”, Advances in Cryptology – EUROCRYPT, Lecture Notes in Computer Science, ISSN 978-3-642-01000-2, vol 5479, pages 34-53, Springer, Berlin, Heidelberg.

[20] Alexandra Boldyreva, Nathan Chenette, and Adam O’Neill, “Order-preserving encryption revisited: Improved security analysis and alternative solutions”, Lecture Notes in Computer Science, ISBN 978-3-642-01001-9, pages 578–595, Vol. 6841, Springer, 2011.

[21] Alexandra Boldyreva, Nathan Chenette, and Adam ONeill. “Order-preserving encryption revisited: Improved security analysis and alternative solutions”. Cryptology ePrint Archive, Report 2012/625, 2012. http://eprint.iacr.org/.

[22] Dan Boneh and Xavier Boyen, “Efficient selective identity-based encryption without random oracles”, Journal of Cryptology, ISSN 0933-2790, Volume 24, Issue 4, pages 659–693, 17th September 2011.

[23] Dan Boneh, Xavier Boyen, and Eu-Jin Goh, “Hierarchical identity based encryption with constant size ciphertext”, Lecture Notes in Computer Science, Springer, ISBN 978-3-540-32055-5, pages 440–456. Vol. 3494, 2005.

[24] E. Bach and J.O. Shallit. Algorithmic Number Theory. Foundations of computing. MIT Press, 1996.

[25] Goldreich, O. “Towards Theory of Software Protection and simulation by Oblivious RAMs" , Proceedings of the nineteenth annual ACM symposium on Theory of computing, ISBN 0-89791-221-7, Vol 11, Issue 2, Pages 182-194, 1987.

[26] Yun Zhang, David Lo, Xin Xia, Tien-Duy B. Le, Giuseppe Scanniello, Jianling Sun, "Inferring Links between Concerns and Methods with Multi-abstraction Vector Space Model", IEEE International Conference on Software Maintenance and Evolution (ICSME), ISBN 978-1-5090-3806-0,Pages 110 - 121, 2016, Raleigh, NC, USA.

[27] Nasser Alsaedi, Pete Burnap, Omer Rana, "Temporal TF-IDF: A High Performance Approach for Event Summarization in Twitter", IEEE/WIC/ACM International Conference on Web Intelligence (WI), ISBN 978-1-5090-4470-2, Pages 515 - 521,, 2016.

[28] Yasmina Bensitel, Rahal Romadi, "Secure data storage in the cloud with homomorphic encryption", 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech), ISBN 978-1-5090-4470-2, Pages 1 - 6, 13-16 Oct. 2016, Omaha, NE, USA.

[29] G. Salton, A. Wong, C. S. Yang, “A Vector Space Model for Automatic Indexing”, Communications of the ACM, ISSN 0001-0782, Volume 18 Issue 11, Pages 613-620, November 1975, Pensilvania, USA.

[30] Wong S. K. M., Ziarko Wojciech, Wong Patrick C. N., “Generalized vector spaces model in information retrieval”, Proceedings of the 8th annual international ACM SIGIR conference on Research and development in information retrieval, ISBN 0-89791-159-8, Pages 18-25, June 05 - 07, 1985, Montreal, Quebec, Canada.

[31] Waitelonis Jörg; Exeler, Claudia Sack, Harald, "Linked Data enabled Generalized Vector Space Model to improve document retrieval", The 14th International Semantic Web Conference,Pennsylvania, October, 2015.

[32] George Tsatsaronis and Vicky Panagiotopoulou, "A Generalized Vector Space Model for Text Retrieval Based on Semantic Relatedness", Proceedings of the EACL 2009 Student Research Workshop, pages 70–78, April 2009, Athens, Greece

[34] Dan Boneh,Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, "Public Key Encryption with keyword Search", Lecture Notes in Computer Science, Springer, vol 3027. ISBN 978-3-540-24676-3, pages 506-522, Volume 3027, January 2004, Berlin, Heidelberg.

[35] A. Swaminathan, Y. Mao, G.-M. Su, H. Gou, A. Varna, "Confidentiality-Preserving Rank-Ordered Search", Proceedings of the 2007 ACM workshop on Storage security and survivability, StorageSS '07, ISBN 322-44-234, Pages 7-12,October 29 - 29, 2007, Virginia, USA.

[36] R. Rivest, A. Shamir, L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", Communications of the ACM, Pages 120-126,Volume 21 Issue 2, Feb. 1978, ACM, New York, NY, USA

[37] C. Gentry, Fully Homomorphic Encryption Using Ideal Lattices. PhD thesis, 2009

[38] D. Stehle, R. Steinfeld, Faster fully homomorphic encryption, Proceedings of Advances in Cryptology, Lecture Notes in Computer Science, ISBN 78-3-642-17373-8, vol 6477, Issue 14, 2010, Springer, Berlin, Heidelberg.

[39] N. Smart, F. Vercauteren, Fully homomorphic encryption with relatively small key and ciphertext sizes, Proceedings of Public Key Cryptography PKC'10, Lecture Notes in Computer Science,ISBN 978-3-642-13013-7, vol 6056, pp. 420–443, Springer, Berlin, Heidelberg, 2010

[40] C. Gentry, S. Halevi, "Implementing Gentry fully-homomorphic encryption scheme", Proceedings of Advances in Cryptology, EUROCRYPT'11, Lecture Notes in Computer Science,ISBN 978-3-642-20465-, vol 6632. pp.129-148, Springer, Berlin, Heidelberg 2011

[41] J.S. Coron, D. Naccache, M. Tibouchi, Public key compression and modulus switching for fully homomorphic encryption over the integers, Proceedings of the 31st Annual international conference on Theory and Applications of Cryptographic Techniques, ISBN 978-3-642-29010-7,Pages 446-464, 2012, Berlin, Heidelberg

[42] S. Halevi, An implementation of homomorphic encryption. http://github.com/shaih/HELib

[43] Z. Brakerski, C. Gentry, V. Vaikuntanathan, (Leveled) fully homomorphic encryption without bootstrapping, in Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, ITCS'12, pp. 309–325,2012, Berlin, Heidelberg

[44] N. Howgrave-Graham, Approximate integer common divisors, Proceedings of Cryptology and Latticed, CaLC'01,ISBN 978-3-540-42488-8, vol 2146, pages 51–66, Nov 2001, Berlin.

[45] J.C. Lagarias, The computational complexity of simultaneous Diophantine approximation problems, Journal of Computing, ISSN 3241-2342-11, Vol 14, Issue 1, pages 196-209,1985, Chicago, IL, USA.

[46] D. Coppersmith, Small solutions to polynomial equations, and low exponent RSA vulnerabilities. Journal of Cryptology, ISSN 0933-2790, Vol 10, Issue 4,pages 233–260, Jul 1997

[47] P.Q. Nguyen, J. Stern, Adapting density attacks to low-weight knapsacks, Proceedings of Advances in Cryptology, ISBN 978-3-540-32267-2, vol 3788, pp. 41–58, 2005, Springer, Berlin, Heidelberg.

[48] Bo Dong, Qinghua Zheng , Feng Tian, Kuo-Ming Chao, Rui Ma, Rachid Anan, "An optimized approach for storing and accessing small files on cloud storage", Journal of Network and Computer Applications, Elsevier, ISBN 344-567-456-06, Volume 35,Issue 6, Pages 1847-18622, Nov 2012.

[49] Yang Zhang, Dan Liu, "Improving the Efficiency of Storing for Small Files in HDFS", International Conference on Computer Science and Service System, ISBN 978-1-4673-0721-5, August 2012, Nanjing, China.

[50] Fang Zhou, Hai Pham, Jianhui Yue, Hao Zou, Weikuan Yu, "SFMapReduce: An Optimized MapReduce Framework for Small Files" , IEEE International Conference on Networking, Architecture and Storage (NAS), ISBN 978-1-4673-7890-1,pages 23-32, August, 2015, Boston, MA, USA.

[51] Karthik Kambatla, Giorgos Kollias, Vipin Kumar, Ananth Grama, "Trends in big data analytics", Parallel and Distributed Computing Articles, Elsevier, ISSN 4523-4322-23-2, Volume 74, Issue 7, Pages 2561-2573, January 2014.

[52] K. Shvachko, H. Kuang, S. Radia, R. Chansler, "The hadoop distributed file system", IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), ISBN 2232-2342-56-7, pp. 1–10, China, 2010

[53] Mackey G, Sehrish S, Wang J.,Improving metadata management for small files in hdfs", IEEE international conference on cluster computing and workshops, CLUSTER'09, ISBN 978-1-4244-5011-4, New Orleans, LA, USA, January 2009.

[54] SequenceFile. Sequence file wiki; 2011. /http://wiki.apache.org/hadoop/ SequenceFileS.

[55] HBase. Apache HBAse documentation. https://hbase.apache.org/

[56] PIG Programming Tool, Apache PIG, https://pig.apache.org/

[57] HIVE Data Warehousing Tool, https://hive.apache.org/

[58] X. Liu, J. Han, Y. Zhong, C. Han, X. He, Implementing webgis on hadoop: a case study of improving small file i/o performance on hdfs, IEEE International Conference on Cluster Computing and Workshops, CLUSTER'09,ISBN 978-1-4244-5012-1,IEEE, 2009.

[59] Shen C, Lu W, Wu J, Wei B. A digital library architecture supporting massive small files and efficient replica maintenance, Proceedings of the 10th annual joint conference on digital libraries,ISBN 978-1-4244-8133-0, pages 391–402, Gold Coast, Queensland, Australia, June 2010.

[60] H. Liao, J. Han, J. Fang, Multi-dimensional index on hadoop distributed file system, International Conference on Networking, Architecture and Storage (NAS), IEEE, ISBN 978-1-4244-8133-0,pp. 240–249, 2010, Coimbatore, India.

[61] S. Chandrasekar, R. Dakshinamurthy, P. Seshakumar, B. Prabavathy, C. Babu, A novel indexing scheme for efficient handling of small files in hadoop distributed file system, International Conference on Computer Communication and Informatics (ICCCI), IEEE, ISBN 4556-5645-3334-09, May 2013, Seoul, Korea.

[62] Apache Hadoop, http://hadoop.apache.org/

[63] Guidelines on Security and Privacy in Public Cloud Computing, NIST Special Publication 800-144, December 2011.

[64] Report by Times of India https://timesofindia.indiatimes.com/india/over-half-a-million-indian-facebook-users-may-have-been-affected-in-data-breach/articleshow/63625675.cms

[65] AAdhar Data Leakage Report: https://www.zdnet.com/article/another-data-leak-hits-india-aadhaar-biometric-database/

[66] White Paper on Big Data by International Data Corporation (IDC) https://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm

[67] White Paper on Security Leakage from AWS Cloud by TechTarget https://searchsecurity.techtarget.com/news/450422709/Misconfigured-AWS-S3-bucket-exposes-millions-of-Verizon-customers-data

[68] Pinkas B., Reinman T. (2010) Oblivious RAM Revisited. In: Rabin T. (eds) Advances in Cryptology – CRYPTO 2010. ISBN 978-3-540-32267-2, vol 6223, pp. 151–158, 2005, Springer, Berlin, Heidelberg

[69] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, US Nat'l Inst. of Science and Technology, 2011.

[70] R. Curtmola et al., "Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions", Proceedings of the 13th ACM conference on Computer and communications security, ISBN 1-59593-518-5, pages 79–88, 2006, Virginia, USA

[71] P. Samarati and S. De Capitani di Vimercati, "Data Protection in Outsourcing Scenarios: Issues and Directions", Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ISBN 978-1-60558-936-7, Pages 1-14, 2010, Coimbatore, India.

[72] C. Gentry, "Computing Arbitrary Functions of Encrypted Data," Communications of the ACM, DOI 10.1145/1666420.1666444, vol. 53, Issue. 3, pp. 97–105, 2010, New York, USA.

[73] Z. Zhang et al., "Bedtree: an All-Purpose Index Structure for String Similarity Search based on Edit Distance", Proceedings of the ACM SIGMOD International Conference on Management of data, ISBN 978-1-4503-0032-2, pages 915–26, June 2010, Indiana, USA.

[74] J. Li et al., "Fuzzy Keyword Search Over Encrypted Data in Cloud Computing," Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, ISBN 978-1-4503-0032-2, pages 441–45, Dubai, May 2010.

[75] C. Wang et al., "Achieving usaBle and Privacy-Assured Similarity Search Over Outsourced Cloud Data" Proceedings of IEEE INFOCOM, ISBN 978-1-3333-0755-8,pp.451–59, July 2010, Indiana, USA

[76] J. Sun et al., "Hcpp: Cryptography based Secure EHR System for Patient Privacy and Emergency Healthcare," Proceedings of International Conference on Distributed Computing, ISBN 978-1-4673-0775-8,2011, pp.373–82, 2018,Orlando, USA

[77] E.-J. Goh, “Secure indexes,” Cryptology ePrint Archive, Report 2003/216, 2003, http://eprint.iacr.org/.

[78] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, “Public key encryption with keyword search,” Lecture Notes in Computer Science,ISBN 978-3-642-13013-7, vol 6056, pp. 420–443, Springer, Berlin, Heidelberg, 2010

[79] B. Waters, D. Balfanz, G. Durfee, and D. Smetters, “Building an encrypted and searchable audit log,” 11th Annual Network and Distributed System, ACM, ISBN 324-433-22-78, Canada, USA, January 2004.

[80] Y.-C. Chang and M. Mitzenmacher, “Privacy preserving keyword searches on remote encrypted data”, Lecture Notes in Computer Science, ISBN 978-3-540-31542-1, vol 3531. Springer, Berlin, Heidelberg, 2005.

[81] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, “Searchable symmetric encryption: improved definitions and efficient constructions,” Proceedings of the ACM SIGMOD International Conference on Management of data, ISBN 978-1-4503-0032-2, pages 915–26, June 2010, Indiana, USA.

[82] D. Boneh and B. Waters, “Conjunctive, subset, and range queries on encrypted data,” Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ISBN 978-1-60558-936-7, Pages 1-14, 2010, Mumbai, India.

[83] F. Bao, R. Deng, X. Ding, and Y. Yang, “Private query on encrypted data in multi-user settings,” International Conference on Computer Communication and Informatics (ICCCI), IEEE, ISBN 4556-5645-3334-09, May 2013, Seoul, Korea.

[84] C. Li, J. Lu, and Y. Lu, “Efficient merging and filtering algorithms for approximate string searches,” IEEE International Conference on Cluster Computing and Workshops, CLUSTER’09,ISBN 978-1-4244-5012-1,IEEE, 2009.

[85]. Behm, S. Ji, C. Li, , and J. Lu, “Space-constrained gram-based indexing for efficient approximate string search,” in IEEE international conference on cluster computing and workshops, CLUSTER’09, ISBN 978-1-4244-5011-4, New Orleans, LA, USA, January 2009.

[86]B. Ji, G. Li, C. Li, and J. Feng, “Efficient interactive fuzzy keyword search,” Proc. IEEE 30th International Conference on Distributed Computing Systems (ICDCS), ISBN 978-1-4244-7262-8, 25th June 2010, Genova, Italy.

[87]J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss, and R. N. Wright, “Secure multiparty computation of approximations,” ACM Transactions on Algorithms (TALG), DOI 10.1145/1159892.1159900, Volume 2 Issue 3, July 2006.

[88] V. Levenshtein, “Binary codes capable of correcting spurious insertions and deletions of ones,” Problems of Information Transmission, Soviet Physics Doklady, Vol. 10, p.707, 1965. http://adsabs.harvard.edu/abs/1966SPhD

[89] Nancy Amato and Michael Loui. Checking linked data structures, In Proceedings of the 38th Annual Symposium on Foundations of Computer Science, ISBN 0-8186-5520-8, IEEE, 06 August 2002, Austin, TX, USA.

[90] M. Bellare. Practice-oriented provable-security. Proc. IEEE 30th International Conference on Distributed Computing Systems (ICDCS), ISBN 978-1-4244-7262-8, 25th June 2010, Genova, Italy.

[91] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption: Analysis of the des modes of operation. Proceedings of the 38th Annual Symposium on Foundations of Computer Science. IEEE,ISBN 506-345-121-06, 1997, https://cseweb.ucsd.edu/~mihir/papers/sym-enc.html.

[92] M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code,Elsevier, DOI: https://doi.org/10.1006/jcss.1999.1694,Journal of Computer and System Sciences, Volume 61, Issue 3, Pages 362-399, December 2000.

[93] Matt Blaze. A cryptographic file system for unix. 1st ACM Conference on Communications and Computing Security, ISBN:0-89791-629-8, Pages 9-16, 1993, Virginia, USA.

[94] M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Noar. Checking the correctness of memories. Algorithmica (1994) 12: 225. https://doi.org/10.1007/BF01185212

[95] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication, Lecture Notes in Computer Science, ISBN 978-3-540-65889-4, vol 1592. Springer, Berlin, Heidelberg

[96] R. Canetti. Studies in Secure Multi-Party Computation and Applications. PhD thesis, Weizmann Institue of Science, Israel, 1995.

[97] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In Proceedings of the 68th Annual Symposium on Foundations of Computer Science, DOI 10.1145/293347.293350,Volume 45 Issue 6, Pages 965-981, Nov 1998.

[98] Benny Chor, Niv Gilboa, and Moni Naor. Private information retrieval by keywords. Report 98-03, Theory of Cryptography Library, 1998.

[99] M. Crispin. Internet message access protocol - version 4. RFC1730, December 1994.

[100] Premkumar T. Devanbu and Stuart G. Stubblebine, “Stack and queue integrity on hostile platforms”, IEEE Symposium on Security and Privacy, ISSN 1081-6011, Oakland, CA, USA, 1998.

[101] Y. Gertner, Y. Ishai, and E. Kushilevitz. Protecting data privacy in private information retrieval schemes. Proceedings of the 13th ACM conference on Computer and communications security, ISBN:1-59593-518-5, Virginia, USA, November 03, 2006.

[102] Oded Goldreich. Secure multi-party computation.Working Draft, https://www.cs.tau.ac.il/~iftachh/Courses/Seminars/MPC/Intro.pdf 1998.

[103] S. Goldwasser and M. Bellare. Lecture notes on cryptography. available online from http://wwwcse.ucsd.edu/users/mihir/papers/gb.html.

[104] E. Kusilevitz and R. Ostrovsky, Replication is not needed: single database, computationally-private information retrieval. 38th Annual Symposium on Foundations of Computer Science, IEEE, ISBN 3428-5632-43-43, London UK, February 1997.

[105]Ralph C. Merkle. “A certified digital signature”,11th Annual Network and Distributed System, ACM, ISBN 324-433-22-78, Canada, USA, January 2004.