Reduction of Information Leakage Related to the Access Patterns in Searchable Symmetric Encryption

MCS

By

Muhammad Awais

A thesis submitted to the faculty of Information Security Department, Military College of Signals, National University of Sciences and Technology, Rawalpindi in partial fulfilment of the requirements for the degree of MS in Information Security

Aug 2021

Thesis Acceptance Certificate

Certified that final copy of MS/MPhil thesis written by Mr. Muhammad Awais student of **MSIS-17** Course Reg.No. **00000274121**, of **Military College of Signals** has been vetted by undersigned, found complete in all respect as per NUST Statutes/Regulations, is free of plagiarism, errors, and mistakes and is accepted as partial, fulfillment for the award of MS/MPhil degree. It is further certified that necessary amendments as pointed out by GEC members of the student have been also incorporated in the said thesis.

Signature:______________________________

Name of Supervisor: **Asst. Prof. Dr. Shahzaib Tahir**

Dated:______________________________

Signature (HoD):__________________________

Dated:______________________________

Signature (Dean):__________________________

Dated:______________________________

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Muhammad Awais

August 2021

Dedication

This thesis is dedicated to my Family, Teachers, and Friends, and my supervisor for their love, endless support, and encouragement

Acknowledgement

All praises to Allah Almighty for the strengths and His blessing in completing this thesis. I would like to convey my gratitude to my supervisor Asst. Prof. Dr. Shahzaib Tahir and co-supervisor Asst. Prof. Dr. Fawad Khan, for their supervision and constant support. I would like to thank my committee members; Asst. Prof. Dr. Mir Yasir and Asst. Prof. Dr. Hasan Tahir (SEE*CS*, NUST) for their support, guidance and knowledge regarding this topic. Their invaluable help of constructive comments and suggestions throughout the thesis work are major contributions to the success of this research.

I would also like to thank my friend Ibrahim Yousef for his support, knowledge, and continuous help in accomplishment of my thesis. I would like to extend my gratitude to my colleague Mehmood-ul-Hassan, and all my family members for their valuable feedback, support, and suggestions.

Abstract

Cloud Computing is becoming a necessity with each passing day. As more and more Cloud Service Provider (*CSP*) are entering the market, the data security factor, which was the foremost concern, is still lurking around. As soon as the data is outsourced to a *CSP*, the original owner would depend on the *CSP* for its security. This may pose as a threat to the security of the data if the confidentiality of data of *CSP* is breached. Due to these issues, the clients are always concerned about their data being outsourced to cloud. Searchable encryption gives clients control over their sensitive data on the cloud. This thesis explores the effects of information leakage when access patterns are disclosed by searchable encryption scheme to an adversary. It is observed that in most of the existing schemes, a successful statistical attack can help the adversary learn the access pattern, provide him with the required document and reveal the underlying data. This paper addresses these issues by devising a SE scheme which leaks minimum possible information related to access pattern. The scheme uses an inverted index-based approach to preserve the high search efficiency inherited from the inverted index while lifting the one-time-only search limitation of the previous solutions. Compared with the existing techniques, this solution uses keyword analysis to extract limited words from the documents based on their occurrence frequency. The test results demonstrate that our scheme is suitable for practical usage with minimal overhead

List of Abbreviations and Symbols

Abbreviations

AES	Advanced Encryption Standard
AE	Application Encryption
BBT	Balanced Binary Tree
CS	Cloud Server
CSP	Cloud Service Provider
CSPRNG	Cryptographically Secure Pseudo Random Number Generator
EHR	Electronic Health Records
FHE	Fully Homomorphic Encryption
HVE	Hidden Vector Encryption
HE	Homomorphic Encryption
IBE	Identity-based Encryption
I	Index Table
IPE	Inner Product Encryption
IP	Internet Protocol

MITM	Man-in-the-middle
MAC	Message Authentication Code
MRSE	Multi-keyword Ranked Searchable Encryption
OPE	Order Preserving Encryption
OPH	Order Preserving Hashing
SO	Search Outcome
PHE	Partial Homomorphic Encryption
PE	Predicate Encryption
PIR	Private Information Retrieval
PKE	Public Key Encryption
PEKS	Public Key Encryption with Keyword Search
RF	Relevance Frequency
SE	Searchable Encryption
SHE	Somewhat Homomorphic Encryption
SSE	Symmetric Searchable Encryption
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
ORAM	Oblivious Random-Access Memory
SPP	Search Pattern Privacy

APP	Access Pattern Privacy
SzPP	Size Pattern Privacy

Symbols

K	Master key
T_w	Trapdoor
IV	Initialization Vector
st_A	State of Adversary
KW	Set of keywords $KW = kw_1, kw_2, \ldots kw_n$
I	Index Table
$a \leftarrow b$	a contains the value of b
λ	security parameter
H(.)	Hash function
O(.)	Big O notation
id(doc)	Document identifier
A	Polynomial Time Adversary

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter covers the brief introduction of the research work and problem formulation of the thesis. It also states the scope, significance and methodology adapted to complete the research work.

1.1 Overview

Considering the huge storage requirements of data these days, the sole solution lies in the usage of cloud computing for day-to-day tasks, thus making it a necessary requirement. As more and more Cloud Service Provider (*CSP*) are entering the market, the security of the critical data, which was the primary concern has not been addressed appropriately. It is evident that cloud computing has provided companies with the opportunity of gaining a major competitive edge against their competitors who have not adopted this technology. Cloud Computing has many advantages like the ease of storage, consistent backups, convenience with easy accessibility and savings on hardware and software maintenance [1]. Multi-tenant customers, who are storing sensitive data in this storage container (cloud), utilize these storage containers provided by *CSP*s. This implies that the control of data is shifted to the *CSP* and the administrators of the respective *CSP*. As control is no longer available to the owner of the data the confidentiality and integrity of data is vulnerable to internal and external attacks. This could possibly cause catastrophic result if the confidentiality of data is breached. Due to these issues, the businesses are always concerned about their data being outsourced to the cloud and where it might be a potential victim of some sort of data breach [1][2]. As reported by McAfee, about 3.1million external attacks were carried out in 2020 on the cloud user accounts. Most of these attacks focused on the vulnerabilities, which involved stolen credentials, IOT, SQL injection attacks along with XSS and malicious file inclusion [3]. Another way to visualize the prevalent threat is being realizing that the medical record of patients, which is to be kept confidential at all cost but when outsourced to a cloud and is obviously vulnerable to cloud attacks because of legal regulations. Another such example would be the criminal records being utilized by the Law Enforcement Agencies.

SE is a relatively new technology in the field of cryptography which helps not only individuals but enterprises as well in storing their data on a cloud server in a secure manner. It gives control to the client over their data on the cloud. The vast usage of cloud and rapid growth in file sharing over the cloud has forced the researchers for devising newer techniques to place their data with sufficient amount of trust on a cloud server. A survey [4] conducted in 2019 indicates that 150,000,000 estimated unique visitors visit just Dropbox in a month. These staggering stats are enough to justify the criticality of the privacy and security of the clients. Figure 1 shows the approximate number of users that the most popular cloud service provider are handling.

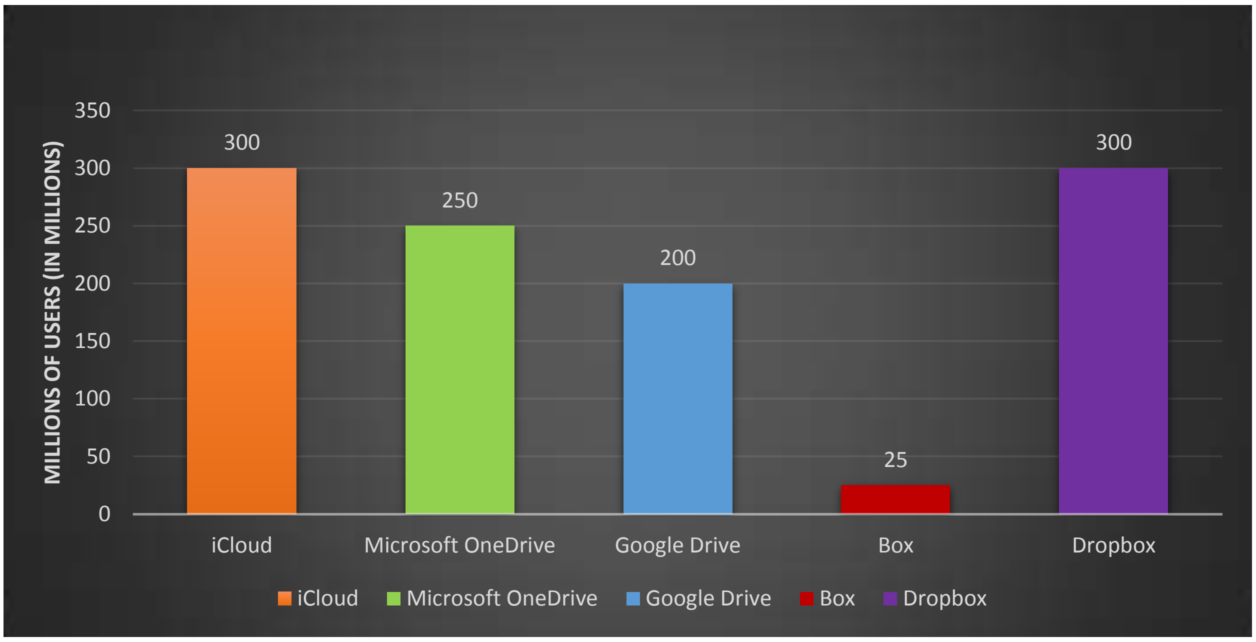


Figure 1: Number of Users per Cloud Storage Service Provider.

1.2 Research Motivation

According to Gartner, 2020 promises to be a big year for cloud computing. The market is expected to grow to $266 billion, up from $227.8 billion in 2019 [69]. The expenditure on IT is expected to increase by 3.2% up on 2019, reaching $3.76 trillion dollars. A study indicates that 75% of the enterprise are having security concerns related to cloud computing even with this much of market. According to Norton, 4.1 billion data records were compromised in 2019 which means that about 130 data records were stolen every second [6]. These are some serious concerns which prevent people from sharing resources on the cloud. Sharing of resources has several advantages but the lack of security and confidentiality raises big concern. For clients, the solution lies in data encryption prior to outsourcing it on any other platform. The model encryption is straightforward but because of efficiency and sometimes practicality issues, the deployment of such solutions is quite difficult. A naïve approach towards outsourcing of data is to encrypt the whole data then outsource it. But what if someone wants a particular document which contains a specific keyword? Then the whole data set needs to be downloaded, decrypted and then each document needs to be checked for the particular keyword matching. To resolve this critical efficiency issue, the scheme of SE is utilized which helps to search for specifics in terms of keywords over the encrypted data placed on any cloud platform. The individual generates a trapdoor which is primarily a search token to search for keywords over the encrypted data. Only the owner of the data has the authority to perform search by using the credentials in the form of correct trapdoors. Thus, a SE scheme proves to be a vital part in providing confidentiality and privacy preservation for the client.

1.3 Security Aims

The primary aim of this research work is to increase security and privacy of client's data while outsourcing it to another entity. Mainly, the focus is on the reduction of information leakage related to access pattern. The access pattern indicates which particular files were retrieved when a query word is issued to the cloud. In this way, the adversary can keep a track by generating a table containing the file identifiers along with the query words issued to the server. Security and privacy work alongside, so the security aims will lead the associated privacy concerns. From the previous discussion, the following security aims can be taken into consideration.

- Authorized Search: Only the owner of the data has the authority to perform search by using the credentials in the form of correct trapdoors.
- Data Confidentiality: The data outsourced to the cloud should not be tempered with and it should be in a secure encrypted format. The corresponding index table should also be secure.

- Access Patterns Leakage: The patterns even should either be concealed completely or the information being leaked does not provide any sort of useful insight to the critical data.

1.4 Design goals

The proposed construction should follow the following performance and security goals:

Trapdoor Unlinking: As discussed before, we are using probabilistic encryption for 3rd party adversary. Thus, the goal for trapdoor is that it should be different for the same keyword throughout a session with a unique key. This helps in avoiding the indistinguishability attacks in searchable encryption.

Privacy Guarantee: The data should remain obfuscated with minimum access pattern leakage, whereas the search pattern may or may not leak. The *CS* and the 3rd party adversary (polynomial time adversary) must not be able to deduce any important especially information related to keyword from the secure index table and trapdoors must remain obfuscated at all time.

Practicality: The proposed scheme should be secure and practical in-terms of efficiency and must provide the client with ease of use.

1.5 Application of Searchable Encryption

The primary application of SE includes those areas where the privacy of a client is of foremost concern. The use of SE in healthcare cloud applications can be considered as one of such examples. The demand in this sector is to protect data privacy and access privacy. In [10], the author has presented a survey of SE for healthcare clouds. Another important emerging field which lacks security since day one is IOT. The devices such as Alexa, google home etc. use the information of the user to store on their respective cloud servers. Considering this information as private, the leakage of such information results in breach of confidentiality. In [11], authors have proposed a framework for secure and privacy preservation of connected cars applications. Another application of SE could be in the field of Ecommerce.

1.6 Thesis Statement

Cloud storage is widely used for providing services including backups or outsourcing data with minimal cost. However, these servers cannot be completely trusted as they often get compromised or the administrator can become a potential threat. Furthermore, a trusted cloud service provider may also outsource data of a customer without his or her consent. To eliminate these threats altogether, the data should be encrypted completely and then stored on the cloud. This causes the search capabilities of data owner to be reduced to bare minimum. To cater this issue, the server can search the data for the data owner by running the search query and then submit the result to the data owner without ever having to decrypt the information during the whole communication. Using SE, we can reduce the possibility of breach of confidentiality along with data searching capability. SE schemes are built on the client/server model. The server stores the data of a single or multiple clients (writer) on the server. A single client or multiple clients (readers) can request the server to retrieve the contents they need. SE comprises of four major architectures:

1. Single reader/Single writer (S/S)
2. Single reader/Multiple writer (S/M)
3. Multi reader/Single writer (M/S)
4. Multi reader/Multi writer (M/M)

The SE scheme is briefly described in Figure 2. This technique, initially lacked security but with the advancement in cryptography and especially in SSE the security of these schemes is improving regularly. Some of the prevalent problems in SSE includes information leakages, which include Search pattern, Index pattern and Access pattern.

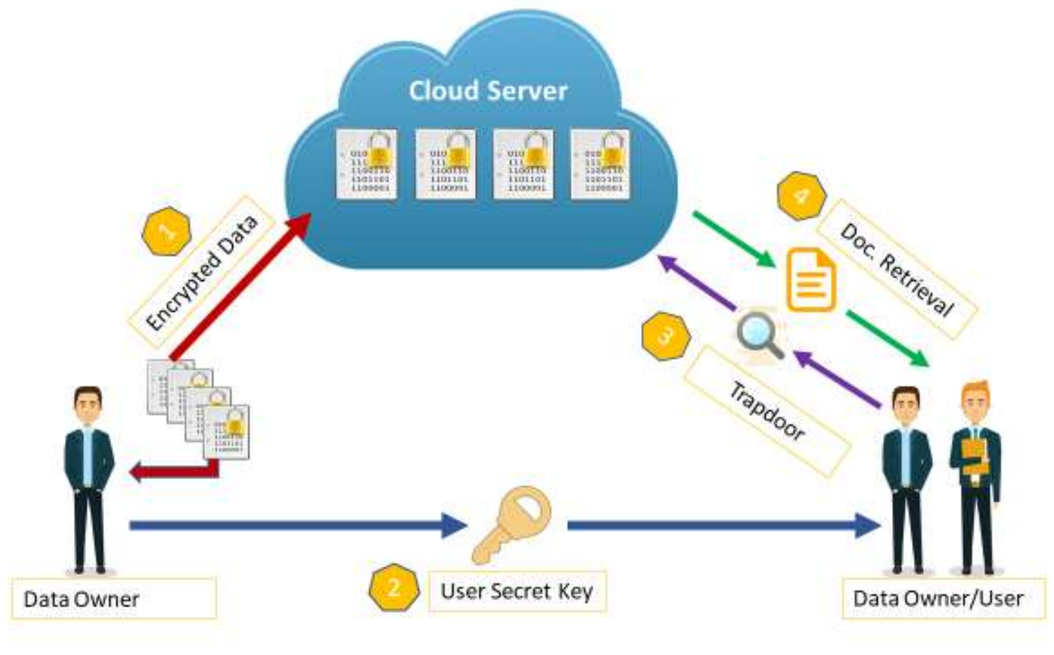


Figure 2: Searchable Encryption.

A survey of the available schemes has been presented in [5]. From this research work, it was concluded that almost all of the schemes leak information in some sort of patterns. Conventional SE schemes [12][13][14] are based on deterministic trapdoors (search query), i.e., for the same keyword searched again the same trapdoor is generated. Some schemes have been previously proposed which provide security against access pattern leakage but they are either computationally expensive or lack practicality

1.7 Thesis Contribution

This thesis explores the possibility of securing data owner's confidentiality. The major areas covered are:

1. For working out the solution, the problems of existing solutions, highlighted in chapter 2, are analyzed.
2. We propose a practical inverted index based single keyword searchable symmetric encryption system. Our system removes the one time only search limitation which is a prevalent issue in the existing systems.
3. We use a probabilistic trapdoor function against 3rd party adversary to break the trapdoor link ability. This helps to counter attacks launched by 3rd party adversaries.

4. The access pattern leakage is reduced by encoding the dataset into multiple fragments which include false positives as well as true positives. The adversary in polynomial time is unable to construct the original file. Also, with the help of hashing along with encryption of keywords, strong security for keywords is ensured.
5. The practicality of the system is also studied through the computation and complexity overhead analysis. A real-world dataset is used and the simulation results show that the system is efficient and strong enough for practical usage with minimum overhead.

1.8 Research Methodology

The research work starts from literature review of the existing techniques being used in searchable encryption. The literature review is done from various academic sources. This research then narrows down to access patterns leakage reduction in the searchable encryption while listing down the drawbacks of existing schemes and formulates the problem. Then, it discusses the access pattern leakage reduction in detail and covers thoroughly the literature, design and implementation part of the thesis.

Implementation of the scheme is carried out using Python in Spyder (Anaconda) in a Linux based environment. A client server architecture is implemented by creating an FTP server. Several modules are also integrated for the complete implementation, which will be discussed in detail in relevant chapter. In the end, a road map for future research areas in the searchable encryption will be discussed. Figure 3 represents the major highlights of the research methodology.

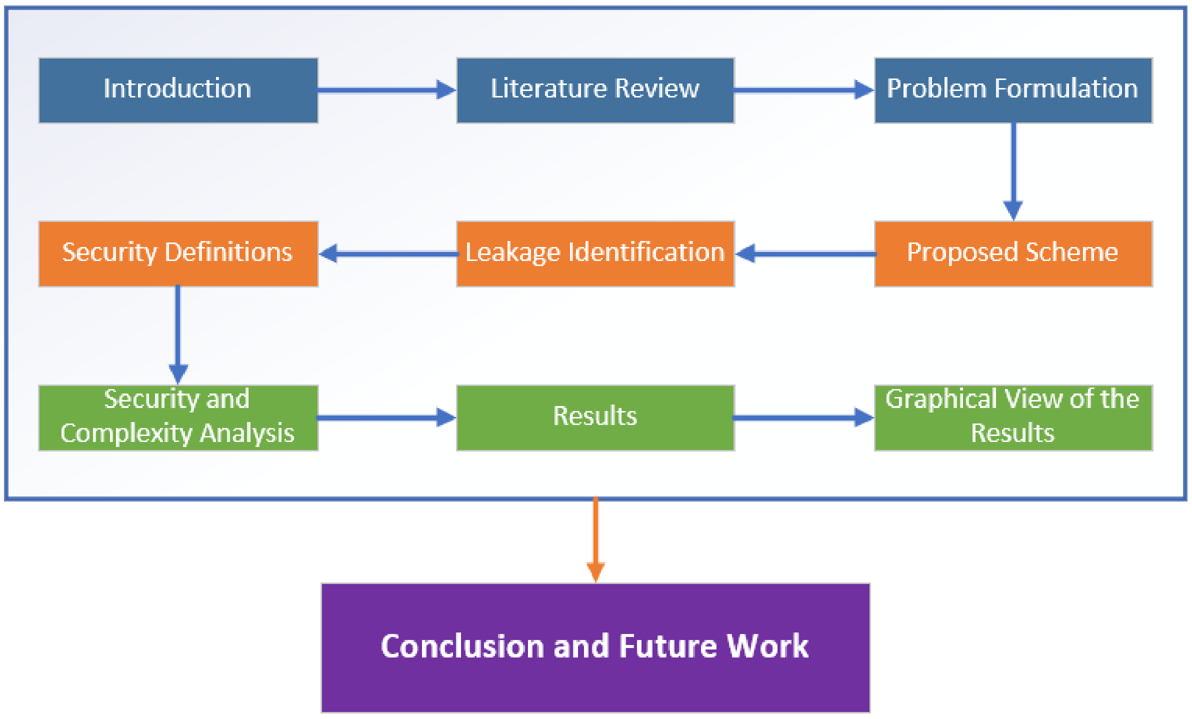


Figure 3: Research Methodology

1.9 Basic Definitions

Table 1: Basic Definitions

Searchable Encryption:	A way of protecting user data while providing searching ability on server side.
Symmetric Encryption:	An encryption technique which uses same key for encryption and decryption.
Search Pattern:	A unique pattern identified during the searching of a keyword in a document.
Access Pattern:	A unique pattern identified during the access and retrieval of a document.
Index Pattern:	An index patterns directs the server which indexes contain the data a particular user is interested in.
Trapdoor:	A function easy to compute in one direction but difficult to compute in the opposite direction.
FTP Server:	A server having the ability to store files and transfer the files to the client when a particular query is provided to the server.
Keyword:	A unique word in a document.
Index Table:	A table that contains all the document ids along with the unique keywords generated against each document.
File Encoding	Split file into several fragments.
Tokenization	Separating each word and punctuation mark for text analysis.

1.10 Thesis Structure

In summary, the thesis breakdown is as follows:

- **Chapter 2: Literature Review** discusses the existing literature on SE. The existing SE schemes along with their limitations are analyzed. The existing security definitions are also analyzed. The schemes which help in securing confidentiality and access patterns are discussed as well.
- **Chapter 3: Proposed Work** introduces a scheme for the access pattern leakage reduction. The proposed scheme is based on single keyword SE scheme. The scheme uses probabilistic trapdoors. The algorithm aims to reduce the leakage of data during the access of a file. The practical feasibility of the scheme is also discussed in this chapter.
- **Chapter 4: Implementation and Performance Analysis** covers the complete implementation of the proposed scheme in a python-based environment. The scripting language is python used a Spyder, which is one of the well-known python compilers. The operating system is Ubuntu because the encoding part of the scheme is only possible in the Linux based environment. Two datasets are used to provide a comparative study. The results are generated in the form of graphs and are also presented in this chapter.
- **Chapter 5: Conclusion and Future Work** concludes the thesis by discussing the directions that can be explored in the future research.

Chapter 2

Literature Review

Searchable Encryption (SE) is a technique which has provided the clients/data-owners with the option of searching for a set of keywords over the encrypted data stored in the cloud while keeping the privacy intact. The privacy preservation is an attribute which reduces the data leakage to an adversary when any SE scheme is being used. This type of encryption was firstly introduced in 2000 by Song *et al.* [20]. The criticality of privacy preservation demands that the information of a client should not be leaked to any adversary or any cloud service provider. In the previous chapter, a brief overview of the SE framework is presented. The chapter presented some brief introduction of techniques which inhabit the property of hiding access patterns with pros and cons included.

The design of any SE scheme is linked with the domain usability, the under-lying use case and the associated cloud infrastructure [24]**.** The design primitives need to be addressed before designing any SE scheme. This chapter discusses the pioneering works in SE including techniques, mechanisms and algorithms which are being used for performing computations on encrypted data. Each of these techniques are different in terms of performance, scalability and security. In this section, we categorize the several approaches that are presented based on the leakage of information, key management and their summarized limitations. Figure. 4 represents the different types of SE schemes that are being currently used.

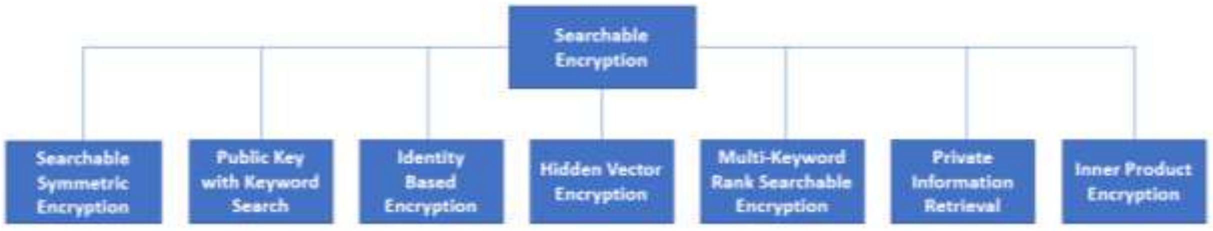

Searchable Encryption
Searchable Symmetric Encryption
Public Key with Keyword Search
Identity Based Encryption
Hidden Vector Encryption
Multi-Keyword Rank Searchable Encryption
Private Information Retrieval
Inner Product Encryption

Figure 4: Types of Searchable Encryptions.

2.1 Preliminaries

- **Inverted Index:** Inverted index generates and stores a mapping from words or numbers to their location in documents. There are two types of inverted indexes. Record-level inverted index, which maintains a mapping of word to documents. Word-level inverted index, along with the mapping of words stores the position of each word within a document as well. The purpose of the inverted index is to allow quick full-text searches at the cost of increased pre-processing when a document is added to the database. For better searching, data is usually transformed. This transformation can be done by dropping the stop words i-e drop the words that have no semantic weight because these are the words that appear in almost all of the documents and they do not help in relating any specific information with the underlying document.
- **Erasure Encoding:** To tolerate failures in the storage system, erasure encoding is used. This is done by adding redundancy to the stored data. An erasure code converts the input message (documents for example) into a longer message. Erasure coding ensures that message reconstruction is possible even if some parts of the longer message have been lost. Precisely, for input message of arbitrary length L, a large message of length M is generated in such a way that reconstruction of input message is done by using both the parity elements along with data elements.
- **Trapdoor Unlinkability:** The trapdoors should be probabilistic and must able to resist distinguish-ability attacks. This will also prevent search pattern leakage. Our system should provide security against an adaptive polynomial time adversary.

2.1.1 Symmetric vs Asymmetric Primitives

These primitives are related to the constructional requirement of a system. The construction may take single or multiple parameters depending upon the underlying use-case. An entity named as a writer, is considered as the data owner of the encrypted documents. The other is reader, whose is considered as the client who needs to search for a particular keyword inside that encrypted document set. If the SE scheme is limited to single reader/single writer [20]**,** then the reader and writer will be the same. In this case, AES [25] algorithm will be used to generate a master key, while the client generates the trapdoors and thereafter searches the cipher-texts. If the architecture of the underlying scheme requires the use of multiple keys, then multiple writer/single reader [21], single writer/multiple reader [22] and multiple writer/multiple reader [26] are used with asymmetric primitives. The requirement of multiple keys is evident and thus this leads to the use of a mechanism which can generate, store, manage and revoke the keys across the network. The key management is the responsibility of a Key Distribution Manager (KDM) [27] or an Application Encryption (AE) [28] server.

2.1.2 Forward privacy vs Backward privacy

Indexing helps in increasing the efficiency of the search process. The forward index helps in finding the documents containing a particular word or a list of words. The requirement of forward privacy is that during any update request, no information should be leaked to the adversary. Chang and Mitzenmacher [29] presented the forward privacy with communication complexity leading to a linear search against the number of updates for the searched keyword. This concept was precisely introduced by Stefanov *et al.* [30]. In the same paper the concept of backward privacy was also introduced. The authors designed a forward private Dynamic Searchable Symmetric Encryption (DSSE) scheme which was based on ORAM. Another SE scheme research work was conducted by Bost [31] who proposed the first forward private SE scheme with optimal computational and communication complexity, achieved forward privacy using trapdoor permutation instead of ORAM-like structure. It provides forward privacy but until the issuance of a new query. This indicates that the adversary has the potential of learning about files that contain the keyword searched previously by analyzing and comparing the access pattern of this newer query with the previous generated queries.

Backward privacy protects the information leakage when a particular keyword pair is added to a database and then deleted, the subsequent search queries on the keyword does not reveal any information about the document. Backward privacy ensures that when two search queries on the same particular keyword are performed, the information about the previously added files and later deleted are not leaked. In 2017, Bost *et al.* [32] presented three levels of backward privacy on the basis of the meta data leaks about the deleted or inserted records. The highest privacy level i-e backward privacy with insertion pattern, leaks the records currently matching each query, when they are inserted, along with the total number of updates on each matched record. But this proves to be insufficient for defense against record-injection or file injection attack. The *CSP*, using insertion times, learn which record was injected by itself or not. It will also aid the *CSP* in recovering the queries. To avoid this leakage, [33] has introduced a strong backward privacy technique. In this technique, the *CSP* can only identify which records match each query but cannot identify when a record is deleted or inserted along with update of any record in the system.

2.1.3 Single Keyword vs Multiple Keyword

The scheme with a specific use-case would demand to be constructed either for a single keyword or multiple keyword query. The use-case is solely dependent on the requirement specified by the client for query effectiveness. However, the ease of effectiveness comes at a cost of lack in privacy and overall efficiency of the scheme. The choice between the type of query needs to be adopted by exploring a balanced approach which maintains a balance between the challenges which are discussed in the section 2.1.

2.1.4 Static vs Dynamic Searchable Encryption

Several approaches have been adopted which address the information leakage issues. In [34]**,** Naveed et al, used blind storage that allows the client a storage for files in such a manner that the server is unable to identify the number of files or the size of the individual file; the server is limited to the existence knowledge of a file. The contents and file contents are hidden even if the same file is downloaded multiple times. The problem with this technique is the leakage of access and search patterns. This scheme only helps in securing the index pattern while leaking access and search patterns.

In [35] the authors have presented two dynamic SE schemes with limited amount of information leakage. Using ORAM algorithms, the authors achieved forward privacy, search and update complexity. Ishai *et al.* [36] has presented a technique which is a combination of Private Information Retrieval (PIR) technique with B-tree data structure. The author has introduced an SSE scheme design which provides constructions with sub-linear search time with no limitations of the inverted index approach. The approach is highly parallel yet simple and can handle updates easily. The construction of this scheme is resistant against adaptive chosen-keyword attacks (CKA-2) This technique provides protection against both search and access patterns leakage. Cao *et al.* [37] designed a static scheme that supports multi keyword ranked search. This scheme hides trapdoor link-ability. Wang *et al.* [38] has proposed a public multi-keyword SE scheme based on Paillier which hides the size, access and search patterns. Wang has proposed an inverted index based public key searchable encryption scheme. This scheme removes the problem of one-time only search limitation contained in the available schemes. With support of conjunctive multi-keyword search using a single trapdoor, the scheme also uses a probabilistic trapdoor linkability. This scheme aims to hide the access pattern by using an efficient oblivious transfer protocol. Compared to the existing schemes, this scheme uses only multiplication and exponentiation for improving efficiency.

In [12], Seny Kamara presented another dynamic SE scheme. This scheme provides the users with sublinear search time, security against adaptive chosen keyword attacks, compact indexes along with the ability of addition or deletion of files effectively. The author provides a formal security definition for dynamic SSE which is CKA2-secure and also achieves the optimal search time. Their scheme is based on the inverted index approach of [40]. The implementation of this scheme shows that this SSE scheme is very efficient.

In [30], stefanov *et al.* proposed another DSSE scheme which leaks very small information during searching of the document identifiers that were deleted in the past. This scheme achieves forward privacy but fails to achieve backward privacy. This scheme is efficient considering the worst-case search complexity and the space for the data structure.

Another approach of Dynamic SE via ORAM is presented in [42]. The scheme provides oblivious access scheme over the encrypted data structure for SE. The scheme is named as Distributed Oblivious Data structure DSSE (DOD-DSSE). The main idea is to create a distributed encrypted incidence matrix on two non-colluding servers such that no arbitrary queries on these servers can be linked to each other. This prevents statistical attacks and threats exploitation of query link-ability. The problem with these schemes is that their construction is static and they have no support for update, insert or delete operations. The author claims that DOD-DSSE ensures the unlinkability of queries and thus offer more security than the existing DSSE schemes. The performance is also almost two times faster than Path ORAM because of less no. of communication overhead. The deployment of this scheme is on Amazon EC2 servers. The search/update time for a query takes no more than one second with very large datasets while the Path ORAM may take 3-13 minutes.

Another Dynamic SE proposed by Bost [31] achieves forward privacy by performing trapdoor permutation but it only provides forward privacy until a new query is issued. These two techniques fail in providing backward privacy and the *CSP* can easily learn about the keywords contained in a file by analyzing the access patterns of a new query with those of previous queries.

In [17], the author uses ORAM to reduce the access pattern leakage. His work starts off with four leakage classes and a single keyword scheme for leakage classes. The scheme is analyzed based on whether it shows performance worse than the streaming entire outsourced data or they don't provide necessary leakage reduction.

Bosch *et al.* [44] scheme is based on inner products. The scheme introduces a cryptographic primitive named as Selective Document Retrieval (SDR). The newer SDR scheme supports equality test predicates and proves its security in the proposed security model. The framework can support flexible search features. The proposed scheme can easily be adapted to support aggregating search results, supporting conjunctive keyword search queries and advanced keyword search. It uses index generation method proposed by Chang and Mitzenmacher, which works with homomorphic encryption. This separates query phase from the document retrieval phase by adding another round of communication. The BTH$^+$ works with PIR for hiding the access patterns, with fulfillment of additional round of communication.

Ferretti *et al.* [46] and Hang *et al.* [47] have presented two different collision resistant techniques which support multiuser access to the outsourced data. They have also included approaches that support the avoidance of key sharing among users. After user revocation it becomes necessary to generate a new key and again encrypt the data. Sun *et al.* [48] uses a CP-ABE (Ciphertext-Policy Attribute-Based Encryption) technique to achieve a scalable Searchable encryption scheme which has support for multiuser read and write operations without needing to share any key. In [49], Shangqi *et al.* proposed a new SSE scheme which is named as Hidden Cross-Tags (HXT). This removes "Keyword Pair Result Pattern" (KPRP) leakage for conjunctive keyword search. The two cryptographic primitives being used here are Hidden Vector Encryption (HVE) and probabilistic (Bloom filter) indexing into the HXT protocol.

2.1.5 Ranked vs Non-Ranked Searching

Ranked search [50] helps the client by providing the most relevant documents from a corpus. This is accomplished by identifying the frequency of a keyword within in a corpus. This returns only the relevant documents but not all the documents that contain the specified keyword provided by the client during the query evaluation phase. The ranking can be in ascending or descending order which depends on the settings specified by the server. The other technique i-e non-ranked search [51] [52] provides all the documents just by identifying whether the specified keyword is in the document or not. The ranking of the document is a bit computationally expensive than the unranked search.

2.1.6 Index-based SE vs Homomorphic-based SE

For an index-based approach, the client is required to first extract the keywords from all the documents in a corpus/document set. The keywords act as an identifier for the document recovery during the query phase. This works in both symmetric and asymmetric modes and can also be deterministic and probabilistic with both being having their own specific advantages and leakages as well. On the other hand, the homomorphic-based SE schemes don't require any sort of index for searching over encrypted data. This also eliminates the need of preprocessing of the data as well. A comparison of both the schemes can be seen in [53].

2.2 Related Work

This section highlights the existing work that many authors have contributed towards the Searchable Encryption. This section is divided into 3 major sub-sections which discuss the various categories of the SE schemes based on the query effectiveness along with pros and cons of the relevant scheme as well.

2.2.1 Single-Keyword SE Schemes

Song *et al.* [20] proposed the single keyword search scheme that did not use an index. Due to this, the server needed to scan each document completely to find the search result. Goh *et al.* [54] was amongst the first to propose a bloom filter-based index which supported single keyword search. Wang *et al.* [55] proposed a ranked keyword search scheme which uses the frequency of keywords to rank the results. The scheme uses deterministic encryption due to which it lacked resistance against distinguishability attacks. Furthermore, the scheme has been successfully compromised using a differential attack leading to leakage of information.

Works from authors in [21] [41] also performed single keyword search on encrypted documents. But, the increase in the requirements of client are growing rapidly and a more accurate and more secure search result is required. Kamara *et al.* [12] utilized dynamic SSE scheme. The dynamic behavior in their scheme provides the ability to add, delete and even modify the document files in the corpus. Additional data structures are used for providing these additional features. A major contribution of this scheme is the use of homomorphic encryption to encrypt the pointer; based on homomorphic encryption the server can modify the file. The problem with this scheme is the use of deterministic trapdoor which makes an adversary to perform distinguishability attacks.

In [30], stefanov *et al.* proposed another single keyword DSSE scheme which leaks very little information during searching of the document identifiers which were deleted in the past and match the keyword. This scheme achieves forward privacy but fails to achieve backward privacy. This scheme is efficient considering the worst-case search complexity and the space for the data structure.

In [16], Naveed *et al.* achieved full adaptive security while revealing minimum information to the server. This is the first scheme that introduces the concept of blind storage in the field of SE. The blind storage helps to hide the number of files along with length of each file while revealing this information only when the same file is downloaded subsequently. The operation of reading, deleting, writing and modification are hidden from the server. In this scheme, each file is divided into a collection of blocks which are kept in a pseudorandom location. The server can only see a superset of locations. The scheme is still lacking security against actively corrupt servers and uses only single-keyword search.

2.2.2 Multi-keyword SE Schemes

Curtmola *et al.* [40] introduced the concept of inverted-index data structure, storing a hash value against each keyword while obfuscating the number of documents matching against each keyword. This results in complexity proportional to the number of documents matching the most frequent queried keyword.

An important work in the field of SE is done by Wang *et al.* [38], in which the authors have given the concept of probabilistic trapdoors over inverted index. The scheme uses Paillier homomorphic algorithm to provide semantic security. An encryption is semantically secure if the adversary cannot extract any useful information about the plaintext from the ciphertext. The scheme is efficient but the exponentiations still contribute to the computational overhead and is a bit more resource intensive than the existing schemes.

Hongwei Li *et al.* [39] provided an efficient, secure and accurate search over the encrypted mobile cloud data using a multi-keyword ranked search scheme. The security analysis of this scheme can effectively achieve confidentiality of trapdoor along with privacy and unlinkability of documents and index to conceals access pattern of the search user. Efficient index tends to provide improved search efficiency. This achieves enhanced efficiency in terms of functionality and search efficiency compared to existing schemes.

Yanzhi Ren et.al [56] provided a lightweight scheme that also supports multi-keyword ranked in cloud computing system. The scheme uses polynomial functions for obfuscating the encrypted keyword and search patterns for efficient multi-keyword search. The scheme is tested using real world datasets with extensive experiments for providing privacy guarantee of the proposed scheme. The problem with the scheme is the issue of important information leakage of search patterns and access patterns in cloud.

In [57], the authors proposed a multi-keyword multi-user SE scheme. The scheme provides secure interactions between authorized users and *CS*. The scheme uses secure trapdoor which provides IND-CKA. The scheme incorporates dynamic addition and revocation of authorized users at the same time of dynamic update of the user accessing to files.

Authors in [58] used another multi-user multi-keyword privacy preserving ranked based search scheme. In this paper, the two major issues related to privacy preservation issues have been identified i-e data fetched as a result of queries and acuteness of keywords sent in queries. For privacy of the documents, the symmetric cipher named Twofish is used. To retrieve the documents of user interest which will result in accessing the top n ranked documents. Obfuscating the query pattern will provide privacy preserving access to data. An index containing hashed values of keywords is used. The authors have analyzed and implemented lucene indexing algorithm.

2.3 Attacks against SE Schemes

In this section, we present how the existing leakage-based attacks can recover not only the queries but the plaintext as well. This includes frequency analysis attack [59], the IKK attack [36] and the file injection attack [60]. Details about these attacks in the context of SE are discussed below.

2.3.1 Frequency Analysis Attack

In [59], the authors have identified an attack on Privacy Preserving Encryption based SE schemes, in which the *CS* could retrieve the items from encrypted databases using the data distribution/frequency information. For a successful attack, the *CS* would require some additional information. This includes, details about application, publicly available stats and some previous versions of the database. This information can be retrieved by a data breach, dumpster diving or through other social engineering techniques. The data distribution information for an encrypted database is equivalent to the database in plaintext. Comparing the leaked information related to frequency of data with obtained stats, which are relevant to the application, the *CS* can recover the encrypted data items. Naveed *et al.* were able to recover more than 60% of the records during the evaluation of this attack with real world medical data using CryptDB. This attack does not require any sort of interaction with users.

2.3.2 IKK Attack

This is one of the first attacks that has displayed the potential damage that can be caused by the exploitation of access pattern leakage. The objective of this attack is to retrieve the plaintext from the encrypted queries. The adversary needs to know some background knowledge about the targeted database through which the possible keywords in the dataset could be recovered. After this, the adversary can guess the keywords in the dataset along with co-occurrence probability. This information is not difficult to gather and it can be analyzed whether the files belong to e-g a university's admin branch, accounts branch or a student information portal. The adversary can simulate the expected co-occurrence of any two keywords by carrying out probabilistic analysis over publicly available online datasets, e-g the documents and news published on the university's website. The adversary guesses n potential keywords and constructs a $n * n$ matrix M whose elements are the co-occurrence probability of each keyword pair. The adversary initiates the *IKK* attack by analyzing the access pattern revealed by the encrypted queries. Specifically, by checking if any-two queries match the same files or not, the co-occurrence rate can be reconstructed. Using simulated annealing technique [19], the adversary can find the best match between M and M and maps the keywords to the guesses. The IKK attack [18] performed by Islam *et al.*, recovered about 80% of the queries with variable vocabulary sizes.

2.3.3 File-Injection Attack

The file-injection attack is an active attack which is mounted on encrypted file collections, also known as chosen-document attack. This attack aims to exploit the access pattern in any encrypted file system by recovering encrypted queries. For this attack, the adversary does not need any sort of auxiliary information; the adversary only requires the keyword universe of the system. In [60] the authors have proposed first file-injection attack. The authors have identified that encrypted queries can be retrieved with a small set of injected files. The adversary sends files which consists of keywords of choice to the user who will encrypt and upload them to the *CS* which will be the injected files. The adversary can know the location of injected files only if no other files are uploaded at the same time. The adversary can check which of the injected files match the relevant query. By injecting files with several keyword combinations, the adversary could retrieve the underlying keyword by analyzing the keywords included in the unmatched and matched injected files. The keywords which are included in the matched injected files but not in the unmatched injected files are the possible searched keywords. For example, if the injected files matching any query *Q* all contain w_1 and w_2, but w_1 is also included in other injected files that unmatched with *Q*, the keyword involved in *Q* must be w_1.

Schemes	**Search Pattern Privacy**	**Access Pattern Privacy**	**Size Pattern Privacy**	**Forward Privacy**	**Remark**
Naveed *et al.* [16]	x	x	✓	x	SSE
Ishai *et al.* [36]	✓	✓	✓	static	ORAM-based
Kamara *et al.* [41]	x	x	x	x	SSE
Chen *et al.* [14]	-	✓	✓	static	SSE
Stefanov *et al.* [30]	x	x	x	✓	DSSE
Hang *et al.* [47]	x	x	x	x	PEKS
Ferretti *et al.* [46]	x	x	x	x	SSE

Table 2: Research Methodologies Comparison

Sun *et al.* [48]	x	x	x	x	SSE
Hoang *et al.* [42]	✓	✓	x	✓	SSE
Cao *et al.* [37]	✓	x	x	static	SSE
Wang *et al.* [38]	✓	✓	✓	static	HE-based
Bosch *et al.* [5]	✓	x	x	static	SSE
Naveed *et al.* [17]	✓	✓	x	x	ORAM-based
Proposed	✓	✓	✓	✓	-

Chapter 3

Proposed Work

Cloud environment enables the clients to access data and utilize on-demand resource sharing remotely. The major services offered by the cloud services include SaaS, PaaS and IaaS. Development in the field of cloud has provided the clients with another service i-e Database-as-a-Service (DaaS) [67] which enables the client to store their data/files on the cloud. The prevalent problems associated with DaaS are trust, security and performance issues along with expectations. In the context of DaaS, searching over the encrypted text or SE is a tedious job and a resource consuming task. Thus, a scheme is required which can perform search over the encrypted data. This would facilitate the client in maintaining the privacy even when the data is outsourced while enabling the searching capabilities over the encrypted documents.

This chapter covers the proposed scheme in detail, modules required for the implementation, index generation, keyword searching, file encoding/decoding, file encryption/decryption and file transfer environment in a client server environment. The trapdoor generated is probabilistic which helps in resistance against distinguishability attacks and mitigate the risk of passive attacks.

3.1 Problem Formulation

SE helps the clients in outsourcing their critical data securely. During the outsourcing of the data, certain significant leakages can be observed by the malicious or curious server. Threat modelling helps in finding potential exploits which can later become stern threats. Also, a threat model helps in securing the data from the entities like internal threat actors or external malicious threats. Our approach is similar to [43] as it also uses a similar symmetric searchable encryption and also encodes the data for obfuscation of the access patterns. The threat model works with our system model and helps in establishing the effectiveness of our scheme using our dataset.

3.1.1 Threat Model with Assumptions

A SE scheme typically has either two or three entities: a data owner, cloud server and sometimes data user. The client encrypts the data and generates an encrypted index for searching capability. The documents and the index are outsourced to the cloud server. The main threat in the system is either the cloud server or any eavesdropper/3rd party adversary. Some of the characteristics of a possibly malicious adversary are given below:

A) Trusted/Honest but curious server

Besides protection of data from visible threats, the data owners need to keep an eye out for the servers that have the authority over your data but are needed to be trusted by the client. These servers are not necessarily corrupt but they may develop an interest in a client’s data.

Recent events of data breaches have forced the researchers to identify the problems associated with data protection in a cloud-based environment. In our proposed scheme, the cloud server can only view the encoded files, which even if decoded would give out the encrypted file. The encrypted file can only be decrypted by the key that was used to encrypt it in the first place. Even after decrypting the server would get the hashed value of the text that was present in the document. Thus, we try to limit the amount of useful information being leaked to the server which in any way affects the confidentiality of the file. We assume that the cloud server is not completely malicious and thus would not modify the contents of the data i-e integrity remains intact. In our research, we also try to eliminate maximum exploits. Thus, integrity check is also ensured here.

B) Polynomial time adversary

The adversary is capable of performing a particular number of operations for guessing the contents of the file. The polynomial time here can vary for server but would have a well-defined limit before losing the purpose of all these computations.

C) Adaptive Adversary

The cloud server can maintain a history of all the searches performed over the encrypted data. This implies that the server knows about the search patterns and access patterns. For security analysis of the system, the adversary is provided access to the history and capability to analyze the history by choosing keyword adaptively.

D) Standard Model

A model where it is assumed that the time is limited along with the resources for computation by an adversary. The system is not ideal and is replaced by a random oracle. The proposed scheme provides high level of security in this model using the Advance Encryption Standard (AES). Thus, the adversary with limited resources, cannot possibly decipher data in reasonable amount of time.

3.1.2 System Model

In chapter 1, we described the 4 major architectures which govern the searchable encryption depending on the requirements of client. For our scheme, we use the single writer, single reader scheme. A client server model is implemented by assuming an entity named Bob to be the client, who wants to outsource his data to a cloud server *CS*. Bob has a collection of documents *D* which he intends to upload to the *CS*. The *CS* will act as a storage container for Bob and will not perform any complex computations on behalf of Bob. The only computation that the *CS* needs to perform is the keyword matching for relevant document retrieval for Bob. Before outsourcing his data, Bob must create an inverted or inverted index table which would contain keywords against respective filenames. The *CS* would utilize this index as a lookup table when a query is issued from Bob for document retrieval. The keywords in the index table are first hashed using *SHA-384* and later on encrypted using *AES-256*; which provides the appropriate security needed for keywords during index generation. For documents, we split them into multiple equal sized fragments. The no. of fragments remains the same for a particular dataset. The fragments are encoded and are then sent over to the *CS* for storage.

Now, client has an index table *I* and encrypted encoded documents set *D* which is outsourced to the *CS*. If bob intends to search for a particular document containing a specific word, he would simply compute a probabilistic trapdoor *Tw* and send it to the *CS*. *CS* will use the trapdoor *Tw* to search the keyword in its index table *I* and would return all the relevant documents matching the particular trapdoor (query).

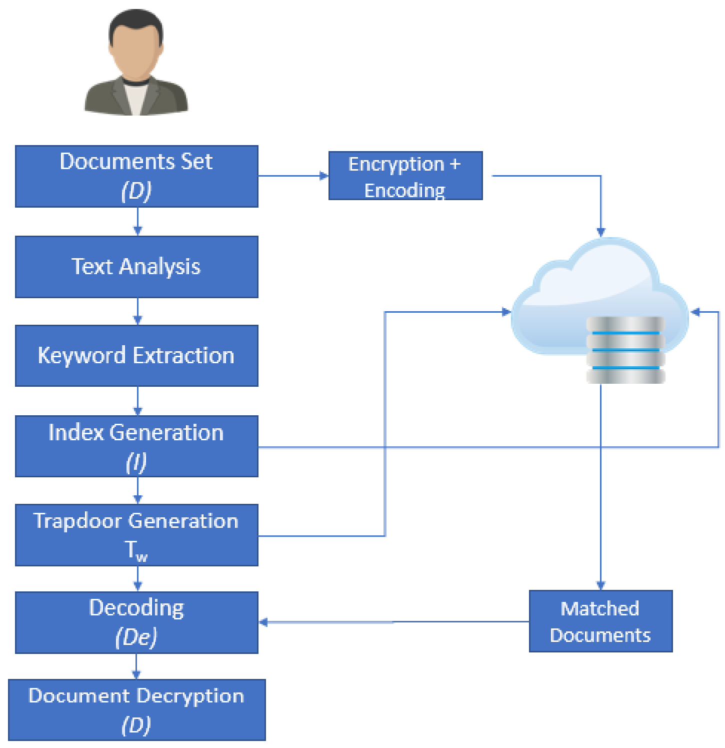


Figure 5: Generalized System Architecture for Scheme.

Figure 5. shows the general system architecture diagram for our scheme. It can be seen that the cloud server is used as a storage container only and no additional computation except searching on encrypted data is being performed on it; which is equivalent to a real word scenario. The major tasks performed by the client are highlighted here. The server returns the relevant document when search is complete.

3.1.3 Deterministic Encryption

In order to highlight the importance of deterministic encryption for our scheme, we need to analyze the definition of deterministic encryption introduced by Bellare, Boldyreva, and O'Neill [23].

A deterministic encryption is a triad of (M, K, C), where M is the message space, K is the key space and C is the ciphertext space *s.t* for each key $k \in K$ and each message $m \in M$, there is one and only one ciphertext $c \in C$ *s.t* $c \leftarrow (m,\ k)$, where c is a unique ciphertext generated for a message m even when the key k remains constant.

The encryption process works as follows:

Depending on the mode of encryption, a message/document is encrypted using a unique key $k \in K$ and sometimes an *IV* as well. The ciphertext remains same till the key is unchanged for a message e-g if you know that the message 'hello world' has the ciphertext '&yy/ m/jyp' under some form of deterministic encryption, then that message will always produce the same ciphertext. This is used when we consider the server as curious.

The advantage of using the deterministic encryption is its applicability in searchable encryption while maintaining efficient search time. The search time in the case of deterministic encryption is logarithmic on encrypted data while the probabilistic encryption offers linear search time [20], [21]. The difference is crucial for large outsourced databases which cannot afford to slow down search. A problem with the deterministic encryption is that it lacks the classical notions of security of randomized encryption [22].

3.1.4 Probabilistic Encryption

In our version of probabilistic encryption, we consider any third person as an eavesdropper in the conversation between client and the server. A random number π is generated whenever a trapdoor is to be generated. The generated the random number is converted into hexadecimal format before concatenating it with the trapdoor to help the client in achieving the randomization between two same trapdoors. With the concatenation, the adversary is unable to identify the length of the trapdoor and thus fails to extract the exact trapdoor for any word.

Thus, our probabilistic encryption is a quadruple of (M, K, C, π), where is M is the message/document, K is the master key, C is the cipher text and π is the random number.

- For every $k \in K$, $m \in M$ and each $c \in C$, there is one and only one $m \in M$ such that $(m, k, c) \in \Omega$, where $\Omega \subseteq M \times K \times C$.

3.1.5 Phases of Algorithm

The scheme comprises of *N* polynomial time algorithms. Following are the phases involved in the algorithm.

1. $(\boldsymbol{K}) \leftarrow \boldsymbol{Keygen}(\boldsymbol{\lambda})$ $\boldsymbol{Phase}$**:** takes as input a security parameter λ and outputs master key *K*.
2. $(\boldsymbol{kw_{list}}) \leftarrow \boldsymbol{Keyword_{Extraction}}(\boldsymbol{D})$ $\boldsymbol{Phase}$**:** takes the document set *D* as input and after performing keyword analysis generates a list of unique keywords against each document.
3. $\left(\boldsymbol{Enc}(\boldsymbol{D})\right) \leftarrow \boldsymbol{Encryption}\ (\boldsymbol{D})$ $\boldsymbol{Phase}$ **:** takes the document set *D* as input as outputs the encrypted document set.
4. $(\boldsymbol{En}\ (\boldsymbol{Enc}(\boldsymbol{D})) \leftarrow \boldsymbol{Encoding}\ \left(\boldsymbol{k}, \boldsymbol{m}, \boldsymbol{ec_{type}},\ \boldsymbol{Enc}\ (\boldsymbol{D})\right)$ $\boldsymbol{Phase}$: takes the encrypted document *D*, number of data fragments *k*, number of parity fragments *m*, and type of erasure encoding ec_{type} as input as outputs the encoded document.
5. $(\boldsymbol{I}) \leftarrow \boldsymbol{Build_{index}}\ (\boldsymbol{En}(\boldsymbol{D}),\ \boldsymbol{kw_{list}})$ $\boldsymbol{Phase}$**:** takes encoded encrypted fragments and keywords of the respective documents to build a secure inverted index.
6. $(\boldsymbol{T_w}) \leftarrow\ \boldsymbol{Build_{Trap}}(\boldsymbol{K}, \boldsymbol{num}, \boldsymbol{kw})$ $\boldsymbol{Phase}$**:** takes as input the master key *K*, random number *num* and a keyword *kw*. It outputs the Trapdoor *Tw* for the query.
7. $(\boldsymbol{d_i}) \leftarrow \boldsymbol{Search_{Outcome}}\ (\boldsymbol{I},\ \boldsymbol{T_w})$ $\boldsymbol{Phase}$**:** takes as input the encrypted index *I* and the trapdoor *Tw* to output *di* which is one of the matched fragments, as the search outcome.
8. $\boldsymbol{Enc}(\boldsymbol{D}) \leftarrow \boldsymbol{Decoding}\ (\boldsymbol{d_i})$ $\boldsymbol{Phase}$**:** takes in the fragments retrieved during the search outcome and returns the original encrypted document.
9. $\boldsymbol{D} \leftarrow \boldsymbol{Decryption}\ (\boldsymbol{Dec}\left(\boldsymbol{Enc}(\boldsymbol{D})\right)$ $\boldsymbol{Phase}$: takes the encrypted document *Enc(D)* and returns the plaintext file.

3.2 Correctness

An SSE scheme is correct [66] if for the security parameter λ, the master key *K* generated by the KeyGen (λ), for *I* output generated by the $Build_{Index}(K, D)$, the search using the trapdoor T_w will always return the correct set of encoded documents fragments.

- If *kw* belongs to d_i, then the following must hold with overwhelming probability:

$$Search_{Outcome}(I,\ T_w) = En(d_i)\ where\ 1 \leq i \leq n$$

- If *kw* does not belong to d_i, then the following must hold with overwhelming probability:

$$Search_{Outcome}(I,\, T_w) = \emptyset$$

3.3 Soundness

The scheme is sound [66] if the $Search_{Outcome}$ does not provide any false positives. This implies that all the phases are correct and thus the search phase always provides sound results.

- If *kw* belongs to D_i, then the following must hold with an overwhelming probability:

$$Search_{Outcome}(I,\, T_w) = 1$$

- If *kw* belongs to D_i, then the following must hold with an overwhelming probability:

$$Search_{Outcome}(I,\, T_w) = 0$$

3.4 Security Definitions

In this section, we propose definitions for the access pattern leakage reduction in our scheme. A scheme is secure with privacy guarantee if it meets the definitions mentioned in this section. In section 3.4, we analyze how our scheme works in compliance with these definitions.

3.4.1 Keyword-Trapdoor Indistinguishability

Keyword-Trapdoor indistinguishability [66] is termed as the process of searching over encrypted text in order to hide any information that may cause the whole process to reveal information related to the plain-text of the corresponding cipher-text. This will help in achieving the security level in which when the same word is searched multiple times, the scheme will generate a unique trapdoor each time. Considering the fact that the 3rd party adversary is trying to maintain a complete history of the encrypted keywords that are being searched against the corresponding keywords, it would not be able to perform further searches in future. In our scheme, the adversary given polynomial time would still get the wrong trapdoor because the hidden operation that is performed on the trapdoor before sending it to the cloud server, is completely hidden from it.

Description

First, the challenger *C* would generate the encrypted index table I for all the words against the documents of the whole dataset. The Adversary *A* is going to select a word *kw* belongs *KW* and send this word to the *C*. *C* will generate the encrypted trapdoor of the word that it has received from the adversary. This encrypted trapdoor will be sent to *A*. This process continues until *A* has received polynomial-many encrypted query keyword trapdoors. Later on, A chooses two keywords kw_0, kw_1 belongs to *KW* and sends them to the challenger. In response, the challenger will send a trapdoor corresponding to the keyword kw_i, where *i* is the outcome of a fair coin toss. If the adversary somehow in polynomial time has probability greater than *1/2* for guessing the keyword, then it will be established that the adversary has succeeded and the scheme will completely lack the property of keyword-trapdoor indistinguishability. However, if the adversary fails to do so, then the challenger will be successful in achieving the keyword-trapdoor indistinguishability property. Let scheme = (*Keygen*, $Keyword_{Extraction}$, *Encryption*, *Encoding*, $Build_{Index}$, $Search_{Outcome}$, *Decoding*, *Decryption*) be an unranked Searchable Symmetric Encryption Scheme over the dictionary of keywords $KW = \{kw_1, kw_2, ... kw_i\}$, documents set $D = \{D_1, D_2..., D_i\}$ with λ being the security parameter and *A* being the adversary tapping into the communication channel.

Considering a probabilistic function of $Index_{Trap_{(SE,A)}}$ (λ):

```
    $Index_{Trap_{(SE,A)}}(\lambda)$:
            $K \leftarrow Keygen\ (\lambda)$
            $I \leftarrow Build_{Index}(En(D), kw_{list})$
            $for\ 1 \leq i \leq m$
                $(st_A, kw_i) \leftarrow A_i(st_A, T_{kw_1 \dots} T_{kw_i})$
                $(T_{kw_i}) \leftarrow Build_{Trap_K}(kw_i)$
        $\boldsymbol{c} \overset{\$}{\leftarrow} \{0,1\}$
        $(st_A, kw_0, kw_1) \leftarrow A_0(\lambda)$
        $(T_{w_c}) \leftarrow Build_{Trap}(K, num, kw_c)$
        $c' \leftarrow A_{m+1}(st_A, T_{kw_c})$
        $T'_{w_i} \leftarrow Build_{Trap}(kw_j);\ j \in N$
```

$$if\ c' = c, output\ 1$$
$$Otherwise\ output\ 0$$

st_A

st_A is used for string representation which records the state of the A. The property of keyword-trapdoor indistinguishability will hold for polynomial time adversary A.

$$Pr\left[Keyword_{Trapdoor_{(SE,A)}}(\lambda) = 1\right] \leq \frac{1}{2} + negl(\lambda)$$

The probability depends on the selection of c which is a fair coin toss.

3.4.2 $Trapdoor_{Index}$ Indistinguishability

This definition [66] is related to the complexity offered by the SE scheme. The trapdoor, index table and keyword must be complex and work in such a manner that they do not reveal any information related to index table entries prior to the search. This implies that for a keyword appearing more than one time, the corresponding trapdoor must be indistinguishable even if the adversary gets hold of some history. Furthermore, if a slight change occurs in the keyword i-e a bit or a character change, then the whole trapdoor and index table must reflect this change.

Description:

The challenger *C* begins by generating an index table *I* against a collection of Document set *D*. The challenger will create an encrypted index table and will send this table and the encrypted dataset to the cloud server. The *A* would choose two keywords which he would want *C* to encrypt i-e kw_0, kw_1. *C* tosses a fair coin *b* and encrypts the keyword based on the output of *b*. This trapdoor is later on sent over to *A*. *A* would try to guess the keyword which is the output of *b*. If the guess of *A* has a probability of more than ½ then *A* having more than average advantage would be considered a winner otherwise *C* would win and it would establish that the scheme provides the trapdoor-index indistinguishability.

Let scheme = (KeyGen, Encoding, Encryption, $\text{Build}_{\text{Index}}$, $\text{Build}_{\text{Trap}}$, $\text{Search}_{\text{Outcome}}$, Decoding, Decryption) be an unranked multi-keyword Searchable Symmetric Encryption Scheme over the dictionary of keywords $KW = \{kw_1, kw_2 ...... kwi\}$, documents set $D = \{D_1, D_2 ..., Di\}$, and λ be the security parameter and A be the adversary tapping into the communication channel.

Consider the following experiment Trapdoor_Index $_{(SE, A)}$ (λ):

```
        $K \leftarrow Keygen\ (\lambda)$
        $I \leftarrow Build_{Index}(En(D), kw_{list})$
        $for\ 1 \leq i \leq m$
            $let\ I' = I[0][x]$
            $let\ kw = (kw_{1, \ldots}, kw_i)$
            $(st_A, kw_i) \leftarrow A_i(st_A, T_{kw_1 \ldots} T_{kw_i})$
            $(T_{w_c}) \leftarrow Build_{Trap}(K, num, kw_c)$
      $\boldsymbol{c} \overset{\$}{\leftarrow} \{0, 1\}$
        $(st_A, kw_0, kw_1) \leftarrow A_0(\lambda)$
        $(T_{w_c}) \leftarrow Build_{Trap}(K, num, kw_c)$
         $c' \leftarrow A_{m+1}(st_A, T_{kw_c})$
         $(T_{kw'}) \leftarrow Build_{Trap(kw_j)}; j \in N$
      $if\ c' = c, output\ 1$
      $otherwise\ output\ 0$
```

st$_A$ is used for string representation which records the state of the *A*. The property of keyword-trapdoor indistinguishability will hold for polynomial time adversary *A*.

$$Pr\left[Trapdoor_{Index_{(SE,A)}}(\lambda) = 1\right] \leq \frac{1}{2} + negl(\lambda)$$

The probability depends on the selection of *c* which is a fair coin toss.

3.5 Scheme Construction

The details of all the phases described previously are given below:

- ***(K) ← KeyGen** (λ):* Given a security parameter *λ*, generate a master key *K*; such that $K \leftarrow \{0, 1\}^{\lambda}$

```
Algorithm 1: KeyGen Phase
Input: A security parameter λ;
Output: Master Key K;
Keygen: Generate key K ← {0, 1}^λ
```

- ***Enc(D) ← Encryption (K, D):*** In this phase, the original documents are encrypted using the master key. Encrypted document set would be returned as a result of this phase.

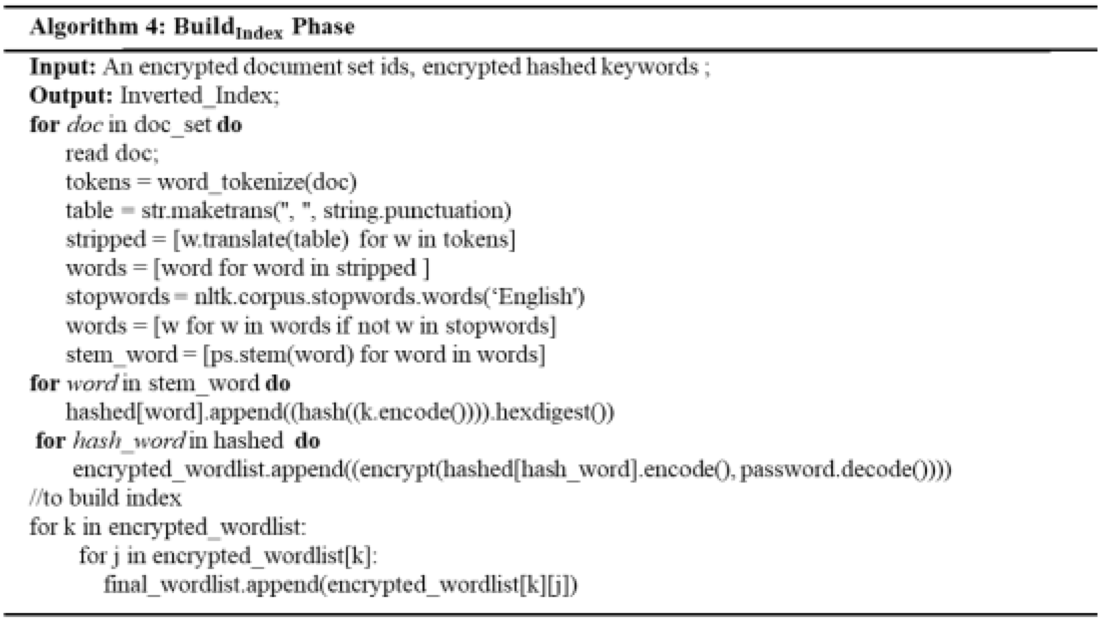
```
Algorithm 2: Encryption Phase
Input: A plaintext document set;
Output: Encrypted documents;
encrypted_docset[]
for doc in doc_set do
    read doc;
    encrypted = encrypt(doc, K)  where K is the Master key
    encrypted_docset.append(encrypted)
```

- ***En (Enc(D)) ← Encoding (k, m, ec_type, fragments, Enc (D_i)):*** In this phase, the encrypted documents are encoded by taking *k*, which is no. of data elements *m*, which is the no. of parity elements, the algorithm used to use *ec_type* and no. of fragments using fragments for file encoding. The output would be a list of (*k*+*m*) fragments against each document.

```
Algorithm 3: Encoding Phase
Input: An encrypted document set, encoded_data_frag, encoded_parity frag, ec_type;
Output: Encoded documents;
encoded_docset[]
encoded_data_frag = 3
encoded_parity_frag = 2
for i in range(0, len(encrypted_docset) do
    encoded_docset.append(sorted_frag[frag_per_file * i: frag_per_file * (i + 1)])
```

- ***(I) ← $Build_{Index}$ (En (Enc(D), kw_{list}):*** Initialize two 1-D List *A* and kw_{list}. Read all documents from document set $D = \{D_1, D_2 ..., D_n\}$. We then encrypt and encode each document and build a List A which contains all the identifiers of the encrypted fragments of respective files of the dataset. *A ← id (En (Enc(D))).* For List

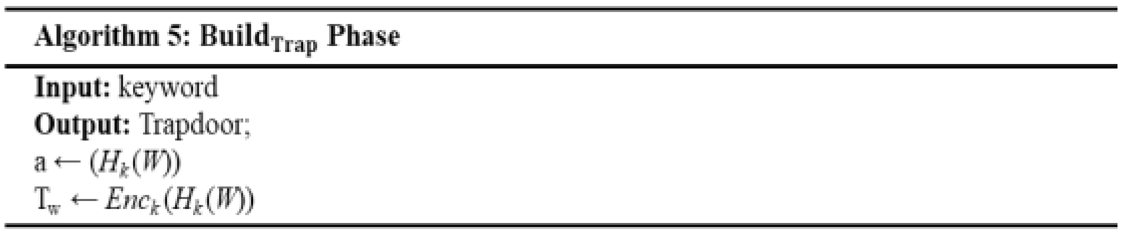
kw_{list}, firstly tokenization is used to separate each word including the punctuation marks (if any). A collection of words is extracted from each respective document to build $kw = \{kw_1, kw_{2...}, kw_n\}$ a set of words against each document. The punctuation marks are removed as they are not marker to be used for searching a particular document. All words are converted into lower-case for generalizing the search query. All the stop words from the List kw_{list} are also removed because they are not required for the search query as all documents contain these words and thus, we cannot use them as search queries. Stemming is used to retrieve the base/root of the word. This helps to map a single trapdoor to a word originating from the same stem. For each document D_i, build a kw_{list} containing specific no. of unique words. After performing all the text analysis, we generate a list C containing the hashes of the words that were generated previously. $C \leftarrow H_k(B)$. Each hash value is encrypted using AES-256, $E \leftarrow Enc\ (H_k(kw))$. Finally, we have a List E which contains a set of unique and distinct hashed + encrypted keywords occurring in D.



Algorithm 4: Build$_{\text{Index}}$ Phase

```
Input: An encrypted document set ids, encrypted hashed keywords ;
Output: Inverted_Index;
for doc in doc_set do
    read doc;
    tokens = word_tokenize(doc)
    table = str.maketrans('', '', string.punctuation)
    stripped = [w.translate(table) for w in tokens]
    words = [word for word in stripped ]
    stopwords = nltk.corpus.stopwords.words('English')
    words = [w for w in words if not w in stopwords]
    stem_word = [ps.stem(word) for word in words]
for word in stem_word do
    hashed[word].append((hash((k.encode()))).hexdigest())
 for hash_word in hashed  do
     encrypted_wordlist.append((encrypt(hashed[hash_word].encode(), password.decode())))
//to build index
for k in encrypted_wordlist:
      for j in encrypted_wordlist[k]:
         final_wordlist.append(encrypted_wordlist[k][j])
```

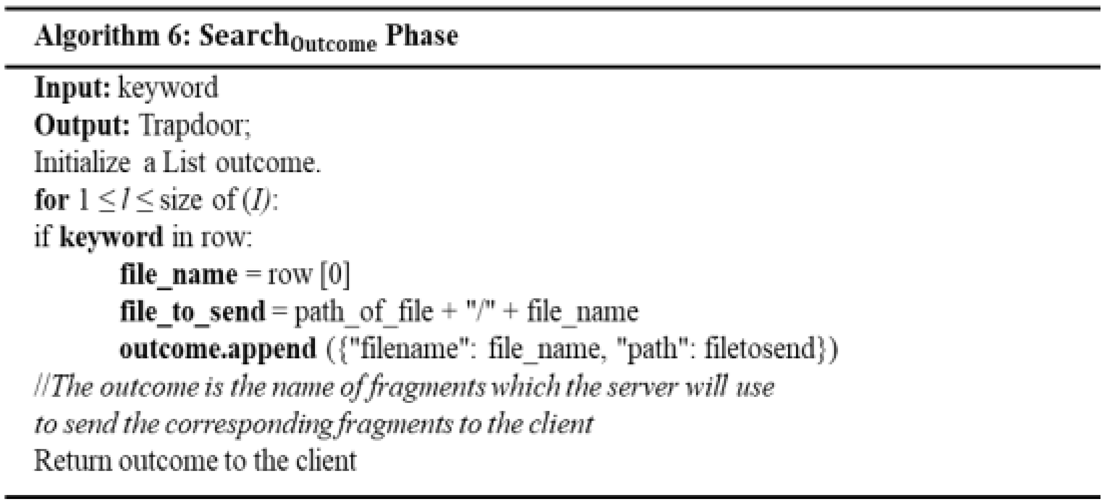
- $\boldsymbol{T_w \leftarrow Build_{Trap}}$***(K, kw, IV):*** let a ← (H_k (kw)) and b ← Enc_k (H_k (kw)). Generate a random number π of arbitrary length in hexadecimal format. Concatenate the random number π with b. This provides the randomization needed for probabilistic trapdoor generation in the case of 3rd party adversary. Set Trapdoor $T_w \leftarrow (b||\pi)$.



Algorithm 5: Build$_{Trap}$ Phase

```
Input: keyword
Output: Trapdoor;
a ← ($H_k(W)$)
$T_w$ ← $Enc_k(H_k(W))$
```

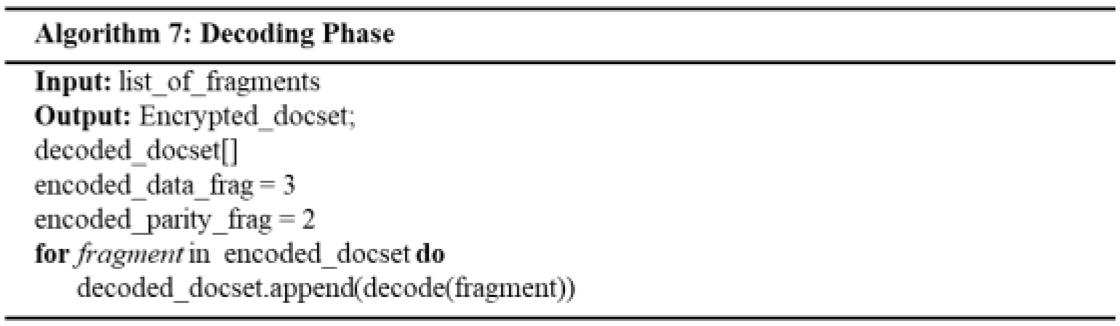
- ***d_i* ← *$Search_{Outcome}$(I, T_w):*** The trapdoor T_w would be matched in the index table *I.* The first column in the table contains all the fragment identifiers. Other column entries against the fragment identifier contains all the encrypted keywords. A scan would be initiated by the server against the provided T_w. The server checks against each entry of columns to find a match against the T_w and constantly saves the fragment identifiers in a list to later on use these identifiers for relevant fragment retrieval.



Algorithm 6: Search$_{Outcome}$ Phase

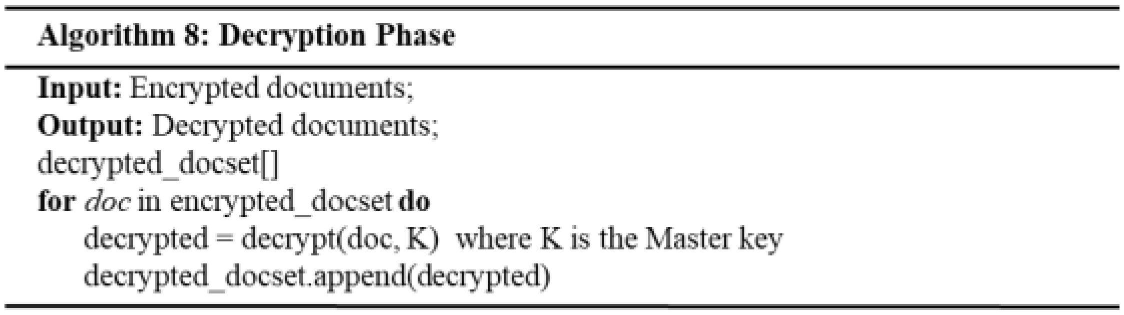
```
Input: keyword
Output: Trapdoor;
Initialize  a List outcome.
for $1 \leq I \leq$ size of ($I$):
if keyword in row:
        file_name = row [0]
        file_to_send = path_of_file + "/" + file_name
        outcome.append ({"filename": file_name, "path": filetosend})
//The outcome is the name of fragments which the server will use
to send the corresponding fragments to the client
Return outcome to the client
```

- ***Enc(D) ← Decoding (k, m, ec_type, fragments, D_i):*** In this phase, the encoded fragments are decoded by specifying *k*, which is no. of data elements, *m*, which is the number of parity elements, the algorithm used *ec_type* and number of fragments using *fragments* for file encoding. The output would be a list of *(k+m)* fragments against each document. A list is used to append all the fragments of the same document. Finally, a list within a list is used which contains all the list of fragments lists against all the retrieved documents.



```
Algorithm 7: Decoding Phase
Input: list_of_fragments
Output: Encrypted_docset;
decoded_docset[]
encoded_data_frag = 3
encoded_parity_frag = 2
for fragment in  encoded_docset do
    decoded_docset.append(decode(fragment))
```

- ***D ← Decryption*** *(K, Enc (D)):* Given a set of encrypted documents, decrypt using the master key K for final output of the search.



```
Algorithm 8: Decryption Phase
Input: Encrypted documents;
Output: Decrypted documents;
decrypted_docset[]
for doc in encrypted_docset do
    decrypted = decrypt(doc, K)  where K is the Master key
    decrypted_docset.append(decrypted)
```

Remark 1: The index table *I* needs to be regenerated whenever any file is added or updated but work fine if any file needs to be removed. This can be avoided using dynamic searchable encryption rather than static searchable encryption.

Remark 2: The keywords stored on the *CS* can be changed using frequency Distribution function of *Natural Language Tool Kit (NLTK)* in our scheme. Using the text analysis steps described earlier, limits the information that an adversary can successfully utilize in launching an attack on the scheme. As selective words will be chosen and stop words will be removed, it would become difficult to guess the words in the documents which are already encrypted. The attacks which make use of access pattern leakage use statistical analysis or file injection attacks for guessing the underlying words. If the frequency distribution is changed by the client and stemming is being used then the statistical attack would require a huge amount of data to uncover the keyword. As words are encrypted, thus we can avoid the frequency analysis attack as well while providing efficient searching. With the use of encoding, we can obfuscate the original size of the file as well and maintain privacy as well.

For enhanced security, one may use Order Preserving Hashing [64] or Order Preserving Encryption (OPH) [65].

3.6 Security Analysis

Most of the existing searchable encryption schemes are unable to resolve the issue of access pattern leakage. The authors in [26] have provided a way that might help to hide the access patterns. The most important of all are the underlying assumptions of the authors and high computational overhead. Other schemes that might have used ORAM to hide the access patterns are theoretically sound but the lack of practicality of their schemes has stopped the authors from using ORAM in newer schemes. The time required to retrieve a single file using any ORAM scheme is more or less equal to the retrieval of all dataset to the user. The searchable schemes work by using a secure index which is sent over to the server for performing search on server side on behalf of client. The index contains the information about the documents associated with a keyword. Thus, in any index-based approach the access pattern leakage on the server will become almost unavoidable. The probable solution that our scheme presents is the avoidance of such attacks that make use of the access pattern leakage e-g IKK attack, frequency attack or file injection attack. As discussed before, our proposed scheme is probabilistic for the 3rd party adversary. We will map our scheme to the definitions stated in Section 5 and we will also highlight any leakage that is leaked during the whole process. While highlighting the leakages, we assume that an adversary *A* in a standard model and we do not replace our scheme with any weak construction. The leakage is based on polynomial time and with adversary having limited computational capacity. Following are the results of the security analysis.

3.6.1 Leakage Profiles

The leakage profiling helps in identifying any leakage significant or insignificant based on certain assumptions. We will discuss the leakage for the server and 3rd party separately because the impact of this leakage may be catastrophic or may be ignored based on the assumptions and scenarios. The leakages being discussed against all the algorithms discussed during the polynomial time algorithms we use in our proposed scheme. While profiling, it must be noted that we are considering the adversary to be malicious while the server to be honest but curious. The security analysis of scheme reveals the following results:

i. Leakage $L_{3.1}$

Description: This leakage is linked with the index table I. It is assumed that the index table is revealed to the server and the client only.

$$L_1(I) = \{Enc_K(H(kw)\ ||\pi, Enc_K(id(D_i)\}$$

ii. Leakage $L_{3.2}$

Description: This leakage is linked with the trapdoor Tw generated for a particular keyword by the client. We assume that the trapdoor Tw is revealed to the cloud server and the client but the adversary *A* gets a different trapdoor than the actual one. The leakage L_2 is defined as.

$$L_2(T_w) = \ \{Enc_K(H(kw)\ ||\ \pi\}$$

iii. Leakage $L_{3.3}$

Description: This leakage is linked with the output of the searching phase against a particular trapdoor. It is assumed that the search outcome is visible to all the stakeholders which include client, *CSP* and the adversary. The leakage L_3 is defined as.

$$L_3(SO) = \{SO(kw), Enc(id(I_i)\ \forall\ T_w \in\ I\}$$

where *SO* is the Search Outcome.

3.6.2 Discussion on Leakages

The trapdoor is generated using a probabilistic encryption algorithm along with a keyed hash function, therefore the leakage associated with the trapdoor can be regarded as meaningless, even when it is revealed to the *CSP*. Now considering the trapdoor in case of 3rd party adversary *A*, the addition of the random number makes it difficult for him to associate a particular trapdoor with the search outcome. This means that if we give access of trapdoor oracle to the adversary accidentally, then the future searches will still be secure against the adversary. Hiding the access pattern is not possible in any index generated scheme but with some extra algorithms added in our scheme, we are able to reduce the attack surface. Also, it is worth mentioning here that the IKK attack performs well when the dataset is dependent and performs very poorly for an independent set of documents.

Considering the capability of *CS* in leakage $L_{3.1}$ scenario, we realize that a curious server has access to the main index table, which is supposed to be an index map for document retrieval. If the server is only curious, then it would only observe the communication as the whole conversation is encrypted. The leakage of Index table is of no concern unless the *CSP* itself turns malicious instead of semi-trusted. But that would waive the client because the fault will be considered on the server side.

The leakage $L_{3.2}$ and $L_{3.3}$ will lead to security and privacy concerns, but we will prove that these leakages do not reveal any information related to the outsourced data. The leakages and assumptions are interdependent and interrelated, therefore to maintain security, all the assumptions must be met.

3.6.3 Formal Security Proofs

Lemma 3.1: The Searchable Symmetric Encryption Scheme present above is "privacy preserving" because it is ($L_{3.1}$, $L_{3.2}$, $L_{3.3}$) secure and according to the Definition *3.1* and *3.2* we have defined above, $L_{3.1}$ is associated with the index table I which leaks the information related to the number of encrypted hashed keywords considered in a document to the cloud server. The leakage $L_{3.2}$ is associated with the leakage of trapdoor information to the *CS*. The leakage $L_{3.3}$ is linked with information made available to all the entities involved in the scheme in terms of search outcome.

Proof Sketch

The adversary A would first send the challenger C a keyword and the response from C is in the form of encrypted trapdoor to A. This keeps on going till the adversary gets all the responses of the keywords and a complete history of all the queried trapdoors is maintained by the adversary.

Later on, the adversary chooses two keywords kw_0, kw_1 belonging to kw and sends them to the challenger. In response, the challenger will send a trapdoor corresponding to the keyword kw_i, where i is the outcome of a fair coin toss. If the adversary somehow in polynomial time has probability greater than ½ for guess the keyword, then it will be established that the adversary has succeeded and the scheme will completely lack the property of keyword-trapdoor indistinguishability. However, if the adversary fails to do so, then the challenger will be successful in achieving the keyword-trapdoor indistinguishability property.

As the adversary gets the trapdoor concatenated with a completely random number each time for a trapdoor for the same word, thus the probability of guessing the keyword becomes less than ½ in polynomial time and this comes in correspondence to the security definitions. From this, we can establish that the challenger will win and the proposed technique will provide the respective indistinguishability property.

3.7 Updating Index Table

Whenever a new file is added or an existing file is removed/deleted from the dataset, it would be required to modify the existing index table. The deletion is a simple process while the addition of the documents some more manual work to be done by the client. Both of these are explained below:

3.7.1 Addition of New Documents

For the addition of new documents in the existing dataset, the client would need to append the document name against the encrypted keywords extracted from the document and append it to the existing index table. The newer index table will be sent to the *CS*, which will use the newer index table to perform searching.

If there are *N* documents that need to be added, the client would place them in folder and point the directory location in code towards this specific location. From there, the client will encrypt and encode the documents. Later, using keyword extraction, all the keywords from the newly added documents will be extracted, hashed and encrypted and will be append to the existing index table and sent over to the *CS*.

3.7.2 Deletion of Existing Documents

For the deletion one or more documents from the dataset, the client will simply instruct the *CS* to delete the documents. The entry of such documents will be removed from the index table along with the keywords that were extracted previously. For deletion of the files, it is to be assumed that the *CS* will perform the operation honestly i-e the *CS* will perform operations honestly but will remain curious and will only observe the whole process and not perform any sort of analytics.

3.8 Correctness and Soundness

This section proves the correctness and soundness defined in Section of the proposed scheme. The scheme comprises of eight polynomial time functions i-e $\mathbb{III}$ = *(Keygen, Encoding, Encryption,* $Build_{Index}$, $Build_{Trap}$, $Search_{Outcome}$, *Decoding, Decryption).*

3.8.1 Correctness

The proposed scheme is correct if for the security parameter λ, the master key K generated using the KeyGen (λ), for the keywords identified by the $\mathrm{Build}_{\mathrm{Index}}$, the search using the encrypted query keyword generated using $\mathrm{Build}_{\mathrm{Trap}}$ and search using the $\mathrm{Search}_{\mathrm{Outcome}}$ returns the correct encrypted documents.

Let K represent the output of the Keygen phase, where the master key is $K \leftarrow \{0, 1\}^{\lambda.}$ Suppose kw_0, $kw_1 \in kw$, we can verify that the below mentioned properties hold.

- Given T_w = $Build_{Trap}$ *(K, kw),* the following equality hold:

$$T_{kw_0} = Enc_K(H(kw_0))$$

- Given T_w = $Build_{Trap}$ *(K, kw`), and* $kw_0 \neq kw_1$ the following inequality holds:

$$T_{kw_0} \neq Enc_K(H(kw_1))$$

This indicates that the property can only hold when $H(kw_0) = H(kw_1)$ which has very negligible probability.

From the above discussion, we can conclude that a unique trapdoor is mapped to a unique keyword. As the index table contains all the encrypted hashed document keyword list against the respective document.

3.8.2 Soundness

The proposed scheme is correct if for the security parameter λ, the master key K generated using the KeyGen (λ), for the keywords identified by the $\mathrm{Build}_{\mathrm{Index}}$, the search using the encrypted query keyword generated using $\mathrm{Build}_{\mathrm{Trap}}$ and search using the $\mathrm{Search}_{\mathrm{Outcome}}$ returns the correct encrypted documents.

Let K represent the output of the Keygen phase, where the master key is $K \leftarrow \{0, 1\}^{\lambda}$. Suppose kw_0, $kw_1 \in KW$, we can verify that the below mentioned properties hold.

- If kw belongs to D_i, then the following must hold with a huge probability:

$$Search_{Outcome}\ (I, Tkw) = 1$$

- If kw does not belong to D_i, then the following must hold with a huge probability:

$$Search_{Outcome}\ (I, Tkw) = 0$$

3.9 Defense against attacks

In this section, we will discuss how our scheme is able to counter some of the attacks which utilize the access pattern leakages for information gathering. The focus is on three attacks which are discussed mostly in the literature against access patterns.

3.9.1 Defense against frequency analysis attack

The frequency analysis attack relies on the analysis of dataset with respect to the keywords extracted against each document present in the document set. It is assumed that all the words are a keyword in the document which will be included in the index table as well. This would help to establish which keywords occur most in the document and using the frequency of the words of the underlying language, it would become easy to identify what is the underlying word against the encrypted trapdoor. Defense against this kind of attack becomes difficult if the scheme is incorporating every word of the document present in the dataset. This not only would not increase the size of the index table but would also increase the searching time as well. To mitigate this kind of threat, our scheme removes almost all the stopwords from the dataset completely. These include the words which are necessary for building sentence structure. Otherwise, it becomes impossible to make sense of the underlying text. Next, we have fixed the number of keywords which can be included against each document in the dataset. Specifically in our scheme only *50* keywords are being added against each document at most. These are randomly selected which makes it more difficult to reconstruct the document using these limited words.

3.9.2 Defense against IKK attack

The IKK attack heavily relies on some very unrealistic assumptions. One such assumption is that the adversary knows the whole unencrypted dataset. Also, this attack works for dependent dataset only which means that each document in the dataset would have several keywords which are presented in more than one document and thus would somewhat become linked with one another. The authors claim that a successful attack has a about 80% recovery chance. Which means that 80% of the trapdoors can be mapped to correct keywords. To eliminate this kind of attack, a probable solution is to add redundant data to the original dataset. Furthermore, it seems that splitting of documents into smaller fragments would also help against this kind of attack because this would increase the hardness for the adversary having limited resources. For defense against IKK attack, we have used erasure encoding which tries to created several fragments of the original document file along with some redundant fragments as well. Also, we have added some false positives in the dataset as well. These simple yet effective technique helps to increase the time taken by the adversary to launch a successful attack on the scheme. Complete mitigation of this attack is difficult because it can only be eliminated if the access patterns are completely obfuscated.

3.9.3 Defense against File Injection attack

In file injection attack, the adversary would attempt to add his personally selected documents. After adding personalized document, when he would query the server, he would be returned with some documents of the actual dataset and maybe he would get one of his own documents as well. When he receives his personalized file, he would pinpoint that the query word actually resides in his own document file as well. Then, he would split that document file in to several smaller fragments and add it to the dataset. Now, when he again queries the server, he would get a document which would be the smaller fragment containing lesser words from the bigger document. After several attempts, he would actually get to the exact keyword. The author in [] claims that with 10,000 such documents he would be able to decrypt all the underlying keywords. This attack seems to be one of the most dangerous attacks which uses the access patterns to generate 100% result but the author has made a major assumption here. Firstly, the author assumes that the index table is being updated out of nowhere and the client himself is adding these files to the index table which seems unrealistic. Normally, in any static SE scheme, the index table would need to be updated whenever a file is to be added in the dataset otherwise the index table would contain no entry of such file and therefore it would never be returned. In our scheme, whenever a file is added, then the whole index table would need to be regenerated and then it would be stored again on the server while overwriting the original one. Now the adversary can try to inject the file in the dataset but if the user has not added any new file in the dataset, he would not generate the index table again. Suppose, later on somehow, he has injected the files successfully in the dataset and has also updated the index table accordingly. Even then it becomes a very hard problem to identify the keyword because his document would obviously several other words besides this keyword. As we are using a static single reader single writer scheme, therefore the user would only regenerate the index table only if he himself is adding some file to the dataset. But if some file from adversary is injected in the dataset, the index table would not be reconstructed and thus this attack would not work.

3.10 Summary

In this chapter, we have explained about the problem that we are solving by formally defining a problem statement. Threat model along with the system model has also been discussed in detail in this chapter. The use of deterministic and probabilistic encryption in different phases of the algorithm is also a part of this chapter. The details about each phase have been presented and discussed in detail. In order to prove the security guarantee of the algorithm, we have presented two security definitions. The security analysis of the scheme provides the mapping of the definitions against the proposed SE scheme. The correctness and soundness of the proposed work is also presented in this chapter. A brief discussion on how the scheme is providing defense against some of the attacks has also been presented in this chapter. In Chapter 4, we will discuss about the performance analysis of the proposed SE scheme.

Chapter 4

Implementation and Performance Analysis

4.1 Description

This chapter presents the computational analysis and performance of the proposed technique. The performance analysis is divided into various parts. For the first part, a discussion on the algorithmic analysis of the proposed scheme is presented. After that a discussion on the storage overhead and computational analysis is provided in detail. The computational overhead of each section of the scheme is discussed in detail.

4.2 Algorithmic Analysis

In order to check the complexity of the scheme, a brief analysis about each phase with respect to time is discussed in this section. The complexity is based on the number of documents present in the dataset. For self-convenience, the number of keywords present in a document is denoted by m, the dataset by *D*, and the number of documents by *n*. The number of keywords depends upon the size of the document file and the number of words other than the stop-words in the document. There are approximately *851* stop-words such as and, the, they that, we, he, she, it etc., which are not going to be considered in the keyword extraction phase of the code. They don't provide any valuable information and thus cannot be interpreted as a valuable keyword for any document. Dataset consists of *2,438* files with variable size. The maximum keywords that will be considered for the datasets is *50* per document to eliminate the threat of frequency analysis attack. The complexity of hash function will be denoted as *h* and complexity of *AES-256* encryption will be represented as *e.* The proposed scheme consists of *9* phases which include KeyGen, Encryption, Encoding, $\text{Keyword}_{\text{Extraction}}$, $\text{Build}_{\text{Index}}$, $\text{Build}_{\text{Trap}}$, $\text{Search}_{\text{Outcome}}$, Decoding and Decryption. The complexity of each phase is discussed as follows:

The key generation and the encryption phases are constant functions and they would require the amount of time to be performed. Being constant, the time complexity for Keygen is *O (1).*

The Encoding phase would depend upon the number of chunks that are created for each document of the dataset. Some fragments contain the required encrypted information while some include redundant data. Thus, the complexity of Encoding phase turns out to be $O\ (u+r)$ where u indicates data fragments and *r* represents redundant fragments. For the $\text{keyword}_{\text{extraction}}$, the algorithm takes in all the words of the document. After opening each document for reading in a loop, another loop is used to save each keyword by removing all the punctuation marks, stop-words, and stemming. As a nested loop is being used here, so the time complexity comes out to be $O\ (n^2)$.

For the $\text{Build}_{\text{Index}}$, phase, the function takes in a list of keywords and document names to construct an encrypted index *I,* which is used later for retrieval of fragments. This indicates that the complexity of this phase depends upon the number of keywords against each document in the dataset and the number of documents. The maximum number of keywords that are considered are *50* while the lowest will be *1*. So, the number of columns will be *51*, where first column consists of document identifiers and the rest of the columns consists of the keywords. This indicates that the number of columns is fixed while the number of rows will vary the number of documents in the dataset. The index generation involves *AES* encryption and Hash functions which gives linear complexity. It can be established that with the increase in number of documents, the time complexity will also increase linearly. Therefore, the complexity of the $\text{Build}_{\text{Index}}$ is *O (n)*.

For $\text{Build}_{\text{Trap}}$ function, the user provides a word as input which would be used to search using the index by the *CS*. The input would be firstly hashed and then encrypted. This means that one Hash function and one AES encryption function is being utilized for building trapdoor. This means the time complexity for this function would be *O (h + e)*.

For the $Search_{Outcome}$ function, the time complexity is fairly straight forward. As keyword matching is to be done against each document of the dataset, thus the complexity turns out to be $O\ (n_d)$ where n_d is the number of documents in the dataset.

As each document is going to be explored for finding a keyword match, and the number of chunks required for reconstruction of the file would exclude the redundant files that were created earlier and only the useful files will be used for reconstruction of the original encrypted document. So, the decoding function would have a complexity of *O(u)* where u represents the useful chunks of a document. After reconstruction from these important chunks, the document will be decrypted using AES which returns data in the hash form for each returned document and at the end, a hash function would be used to get the original plaintext document from this whole algorithm. Thus, the complexity of decryption function turns out to be, *O (n)* where *n* represents the number of documents to be decrypted.

The complexity of the proposed scheme is presented in tabular form in Table below.

Table 3: Complexity of proposed scheme.

ALGORITHMS	**COMPLEXITY**
Keygen	*O (1)*
Encryption	*O (n)*
Encoding	*O (n)*
$\text{Keyword}_{\text{Extraction}}$	$O\ (n^2)$

Build$_{\text{Index}}$	*O (n)*
Build$_{\text{Trap}}$	*O (h + e)*
Decoding	*O (n * u)*
Decryption	*O (d + h)*

4.3 Computational Analysis

The proposed scheme is further tested and analyzed based on the implementation and testing. This particular section provides the implementation analysis in detail. The scheme is implemented in Python language using Anaconda as primary code editor, on a Switchboard-1 Telephone Speech Corpus. Before moving any further, some preliminary details which are necessary for understanding the computational analysis are provided below:

4.3.1 System Specification

The code of the scheme is done mainly in Windows but due to installation problem with *Pyeclib* library, which is needed for encoding of the documents, the environment for this phase was shifted to Linux. The version of *Python 3.8.5* was used and the software used for managing the code was *Pycharm 2020*. To represent the encrypted index in a human readable form, it was generated and written to a *MS Excel 2016 file*. The specifications of the Hardware used are mentioned below in Table 1.

Table 4: System Specifications.

Component	**Specification**
Operating System	Windows 10/Ubuntu 18.04.4 LTS
Memory	12GB
Processor	Intel Core i5-7500U CPU AT 1.90GHz × 4
OS type	64-bit
Virtualization	VMware Workstation 15.5.6 Pro

The coding part required the usage of some pre-built libraries for efficient and less computationally expensive code. The details about the libraries are provided in Table 2.

Table 5: Libraries with versions.

Library Name	**Version**
Python	3.7.9
PyCharm Community	2020.1
NLTK	3.5.0
Hashlib	2.008
CSV	0.0.13
Collections	0.3.0

4.3.2 Dataset Specification

Switchboard-1 Telephone Speech Corpus (LDC97S62) dataset is originally collected by Texas Instruments in 1990-1991 under the sponsorship of DARPA. The Switchboard-1 speech database is aggregation of impulsive conversations addressing the growing need of larger multi-speaker databases of telephone bandwidth speech. The collection consists of 2430 conversations with 6 minutes average length. This means over 240 hours of recorded speech and about 3 million words of text which are spoken by over 500 people. The speakers are of both male and female and dialects are from every major American English dialect. The dataset consists of 120,000 unique keywords but the keywords in this dataset will be reduced to 3,828 because the ignored keywords will be removed during the keyword filtering i-e stop-words removal. This dataset constitutes a realistic dataset of telephonic conversation which helps in understanding the functionality of SE presented in this thesis.

4.3.3 Implementation Details

To analyze the execution time of the proposed technique, the implementation was done in Python language on a Windows 10 and a Linux. For the testing of our scheme, we considered all the files of the corpus which would also give the practicality of the technique in real world scenario as well.

4.4 Storage Overhead

The proposed scheme consists of eight phases. The keygen phase would generate a master key which is used both for encryption and hashing. It is generated before the start of the whole communication between *Client* and *CS*. The length of the master key is *256-bit* in length which indicates that the client would need to store $\frac{256}{8}bits$= *32 Bytes*. Typically, the file size in the dataset considered is not greater than *30000 Bytes* and the total files in the dataset are *2,438*. So, the total storage overhead for catering the whole dataset becomes approximately *56,000,000 Bytes*. Next the client needs to generate an encrypted index from the whole document dataset of *2,438* documents. With *50* being maximum considered words from each document and going through the phases of stop-words removal, punctuation marks removal, lemmatization etc., and then taking hash of each word and encrypting each hash with *AES*, the final storage size for the encrypted Index turns out to be 16,506,000 *Bytes*. At the server side, the *CS* would store the encrypted index, the chunks of the documents and some storage for processing the query received from the client side. Suppose that in the scheme, the client generates *9* fragments against each document. Considering the fact that the *CS* will perform the search for the client, the server will also store the encrypted index as well. The dataset has about *2,438* files. The size of dataset is 56.6MB or 56600KB. Thus, the total storage required on the server side will be (size of index + chunks of dataset + extra space for processing query) i-e (16,506,000 + 56,000,000 + 100,000,000) Bytes = 172,506,000 Bytes or 172.506 MB.

4.5 Computational Overhead

The running time of each of the functions involved in the schemes are provided in this section. The overhead in terms of time is analyzed and present in graphical form by generating graphs in MS Excel. To analyze the running time, the algorithm first takes in a single document file and then the files are doubled subsequently. The testing was done multiple times in order to find whether there was any observed error in the analysis. The error was not more than 0.001 sec even after the code was executed multiple times. The computation overhead of each individual phase for each dataset is provided below:

4.5.1 Keygen Phase

In this phase, the client will first select a secure password. The plaintext password would be encoded to convert it in bytes form. A hash of the secret password would be generated and padding would be done for the data because CBC requires you to pad your data to make sure the final block is filled with some data. A randomly generated IV will be generated which is going to be used for the encryption of the data. As all the operations are being done in bytes, thus the generated IV will also be encoded to convert it into bytes. The AES encryption would be done by taking private key, selecting AES mode (CBC) and the IV as parameters.

4.5.2 Encryption Phase

For the telephonic dataset, the encryption starts with a single file and ends with the encryption of all the files of the dataset. All the documents are first hashed and later encrypted to ensure that not a single bit of information can be leaked from them in terms of confidentiality. For a single file, the execution/encryption time comes out to be 3.1 sec. The linear trends of this algorithm indicate that the encryption time increases as the number of files of dataset increase. For the whole dataset, the encryption time takes about 37.52 sec. Figure 6 shows the graph of encryption phase.

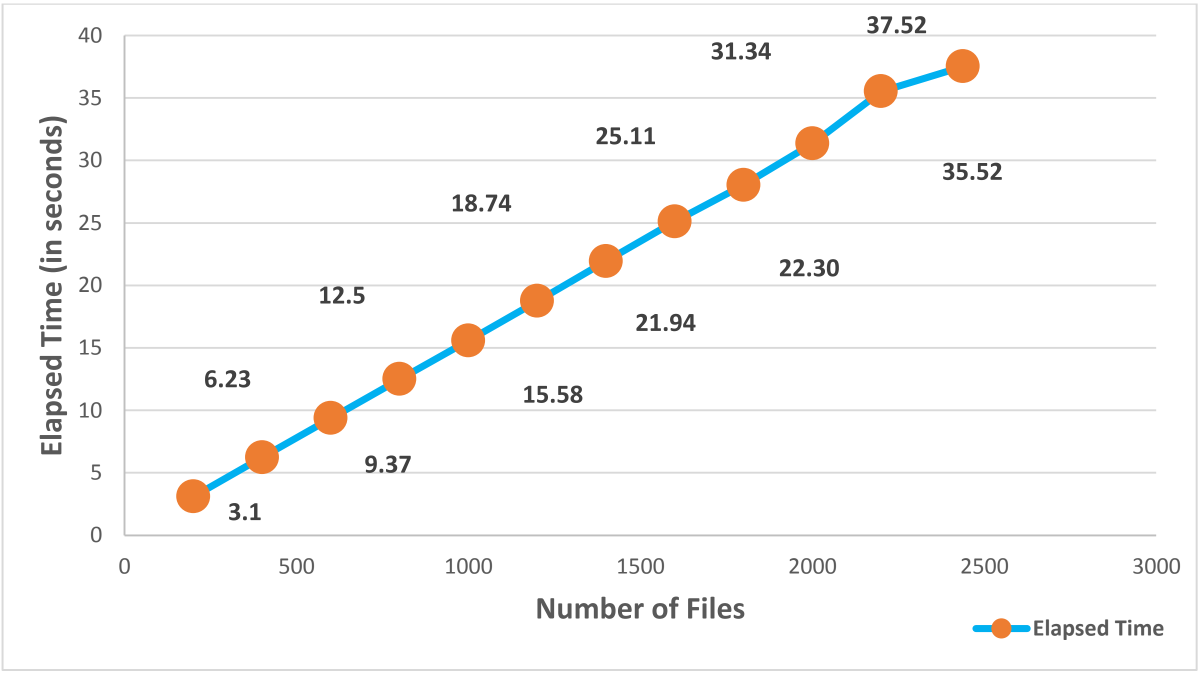


Figure 6: Computational Time for Encryption.

4.5.3 Encoding Phase

For encoding of the dataset, the value selected for k is *5* and for m is *4* (k is the number of fragments that contain the actual encrypted data, while m is the number of fragments that contain redundant data). The total fragments that would be generated from this function would be $k + m = 5 + 4 = 9$. Execution time for encoding of a single file for the dataset comes out to be *0.09 sec*. The total time taken to encode all *2,438* documents is *2.18 sec*. The encoding time varies with the fragments that are generated against each document**.** Figure 7 shows the graph of the encoding phase

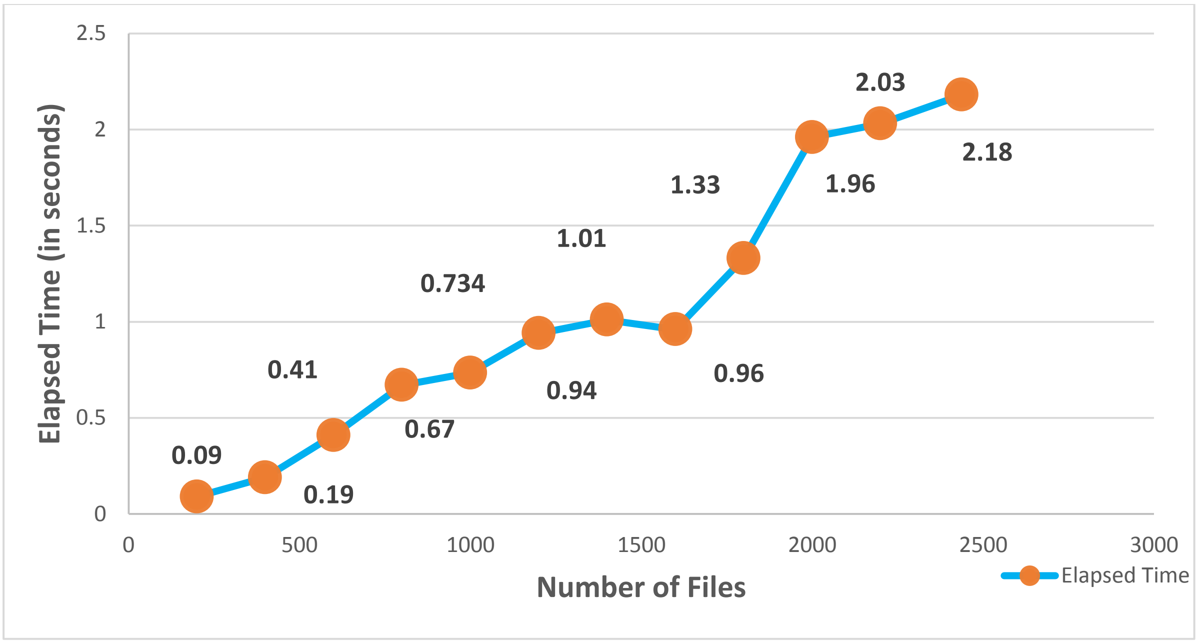


Figure 7: Computational Time for Encoding.

4.5.4 Keyword Extraction Phase

For the keyword extraction phase, firstly *200* files are taken to extract their keywords and with the same gap the graph is mapped to *2,438* files. The keyword extraction phase takes about *13.34 sec* for the first *200 files* and for the complete dataset it takes about *122.6 sec*. The graph in Figure. 8 provides the computational time foe the keyword generation. In order to measure the time interval difference for keyword extraction, we have plotted it with and without keyword analysis phase. It can be observed that it takes almost double the time when extra functions of keyword analysis are being used. For *2,438* files, it takes *61.02 secs* without the keyword analysis functions while with those functions it takes about *117.78 secs*. Figure 8 shows the computational time analysis graph for the keyword generation.

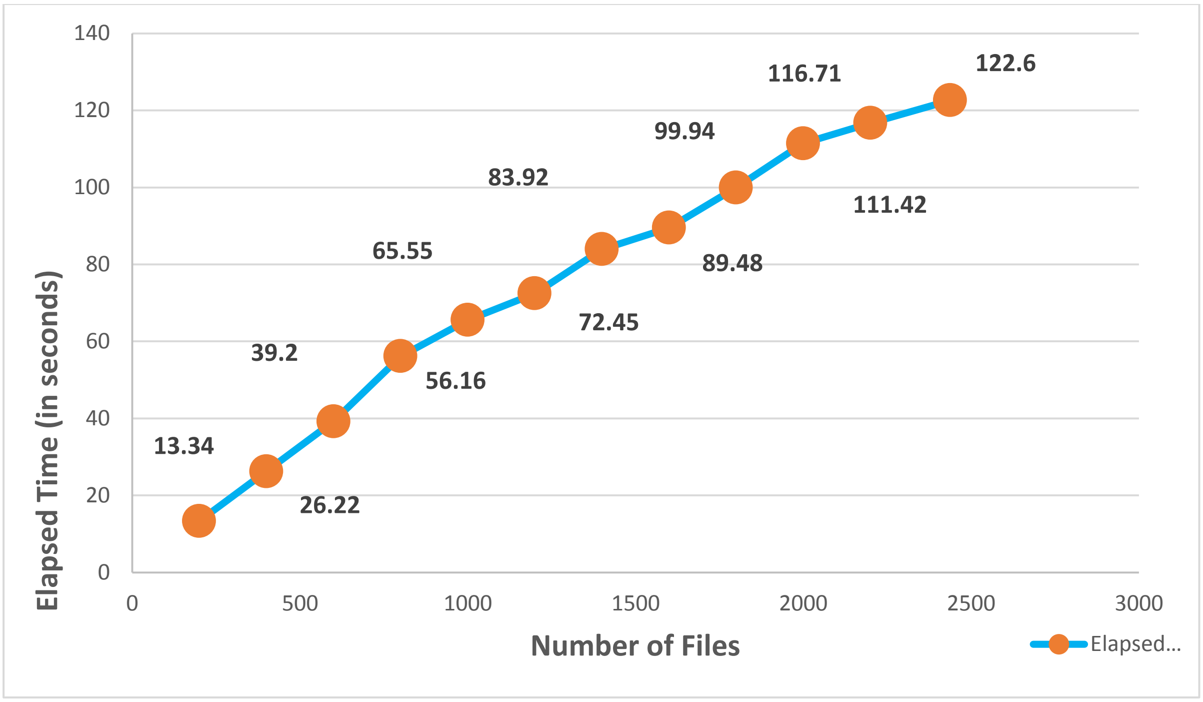


Figure 8: Computational Time for Keyword Generation.

4.5.5 Build$_{\text{Index}}$ Phase

The build index phase consists of index generation against the whole document set when it is neither encrypted nor decoded. The index table will be transmitted to the *CS* where it will be utilized as a lookup table. To analyze the execution time of the code on the whole keyword set identified and extracted from the dataset, we will use construct the index with and without keyword analysis. This would provide a comparison between the time for all the words of the document set against the distinct keywords of the documents. The keyword analysis does pose an overhead in the scheme but the benefit against the overhead is far more and the analysis of the graphs proves it as well. Figure 9 below represents a comparison between index generation with keyword analysis and without keyword analysis. The comparison shows that the overhead of keyword analysis is almost double in terms of time complexity but in terms of storage overhead of index it provides a much smaller and easy to use index table. The comparison starts initially with *200* files and moves up to *2,438* files i-e total files of the dataset. The blue line represents the time required with keyword analysis while the orange line represents time required without keyword analysis. The total time taken to generate the circles (with keyword analysis) is *117.78 sec* while the triangles (without keyword analysis) is *61.02 sec*.

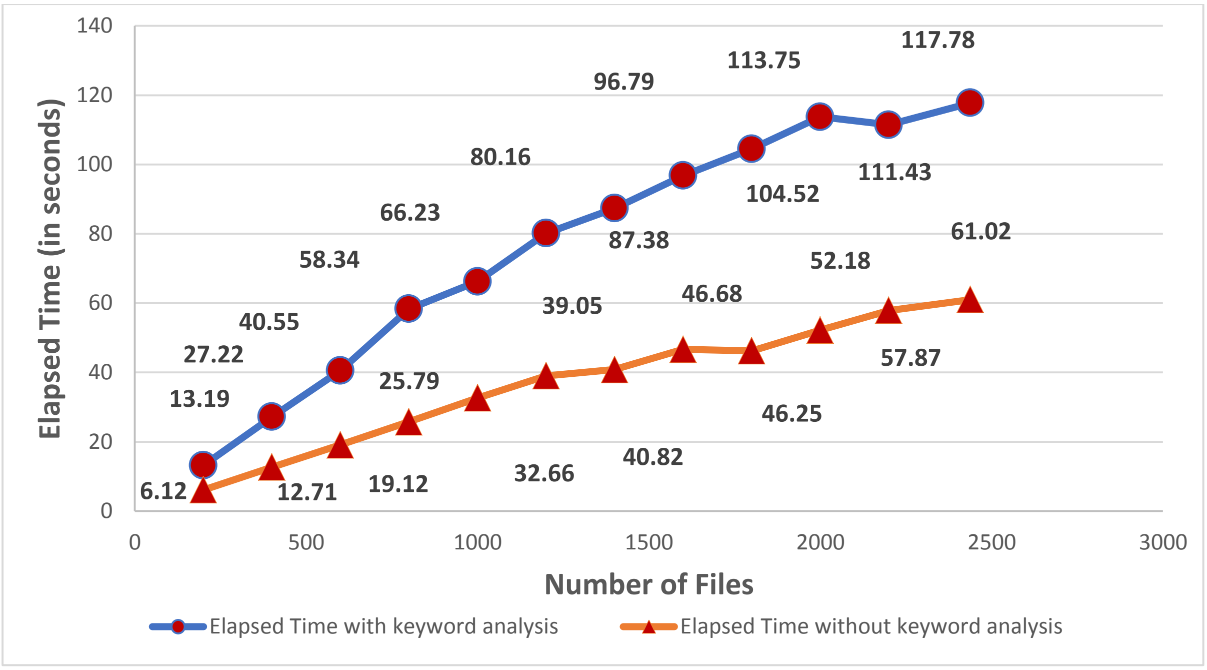


Figure 9: Computational Time for Index Generation with and without keyword analysis.

4.5.6 Build$_{\text{Trap}}$ Phase

The trapdoor is a search token that is generated on client side and is sent over to the *CS* to perform search over the encrypted dataset using the index table. The trapdoor function does not take any specific parameters due to which the time for the trapdoor generation is almost the same and it does not matter which keyword will be used in trapdoor generation function. The time taken to generate trapdoor is about *0.0112* seconds.

4.5.7 Search$_{\text{Outcome}}$ Phase

As discussed before, the search outcome phase in our case is divided into two sub phases. After the search has been perform by the *CS* against the trapdoor provided from client, the *CS* will use the index table to find the match for documents that contain the trapdoor and it will return the encoded documents to the client. The encoded documents will be decoded to get a single encrypted file. The encrypted file will be later decrypted to get the original plain-text file. Figure represents the graph generated against the dataset for the keyword *"threaten"*. The searching takes about *12.06 sec*. Most of the time here is consumed in saving the documents names in a list which is to be sent to the client after search. The labels on the nodes represents the time in seconds against the partition of dataset. The lowest time we get is *1.01* sec against *200* files. Figure 10 shows the computational graph for keyword searching phase.

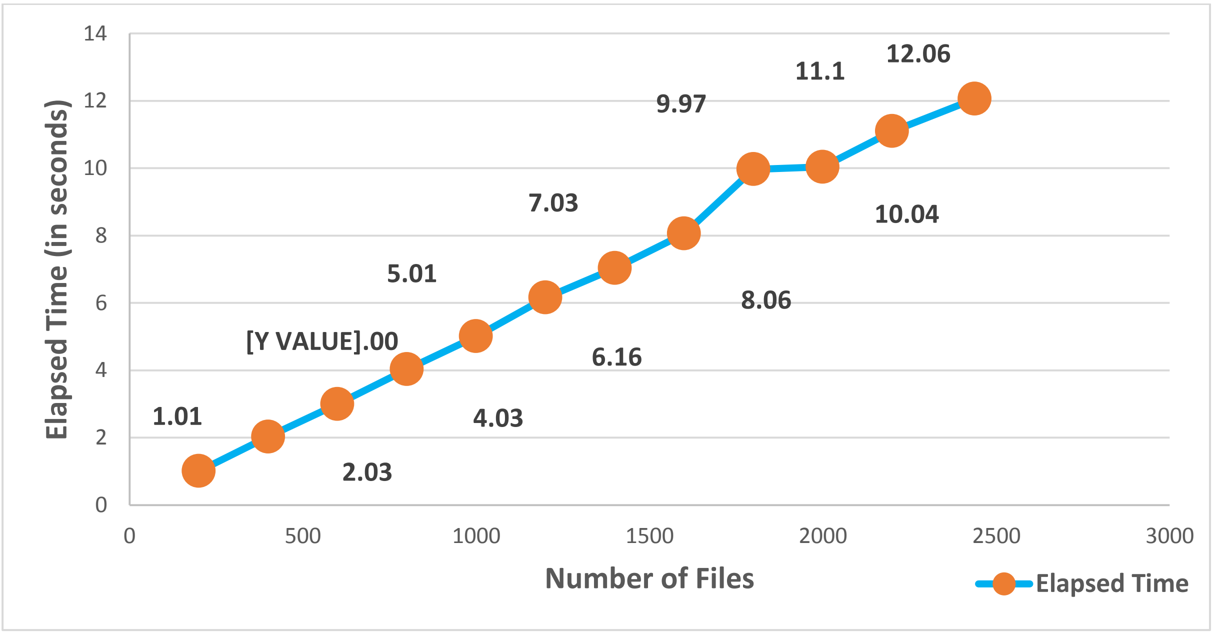


Figure 10: Computational Time for Keyword Searching.

4.5.8 Decoding Phase

After the fragments have been received on the client side, the client would run the decoding algorithm that would help in reconstruction of the original encrypted file. The time taken to reconstruct *200 files* for the dataset is *0.11 sec* while it takes about *5.71 secs*. The graph of Figure suggests an almost linear trend, because the time of reconstruction increases as the number of files in the dataset are increased. The decoding phase would take all the fragments of the files retrieved from the server after query phase. The parameters needed for the proper reconstruction include fragments of file, library used for encoding, number of fragments that contain actual data, and number of fragments that contain redundant data. After the relevant parameters have been passed to the decoding function, the graph in Figure. 11 is obtained.

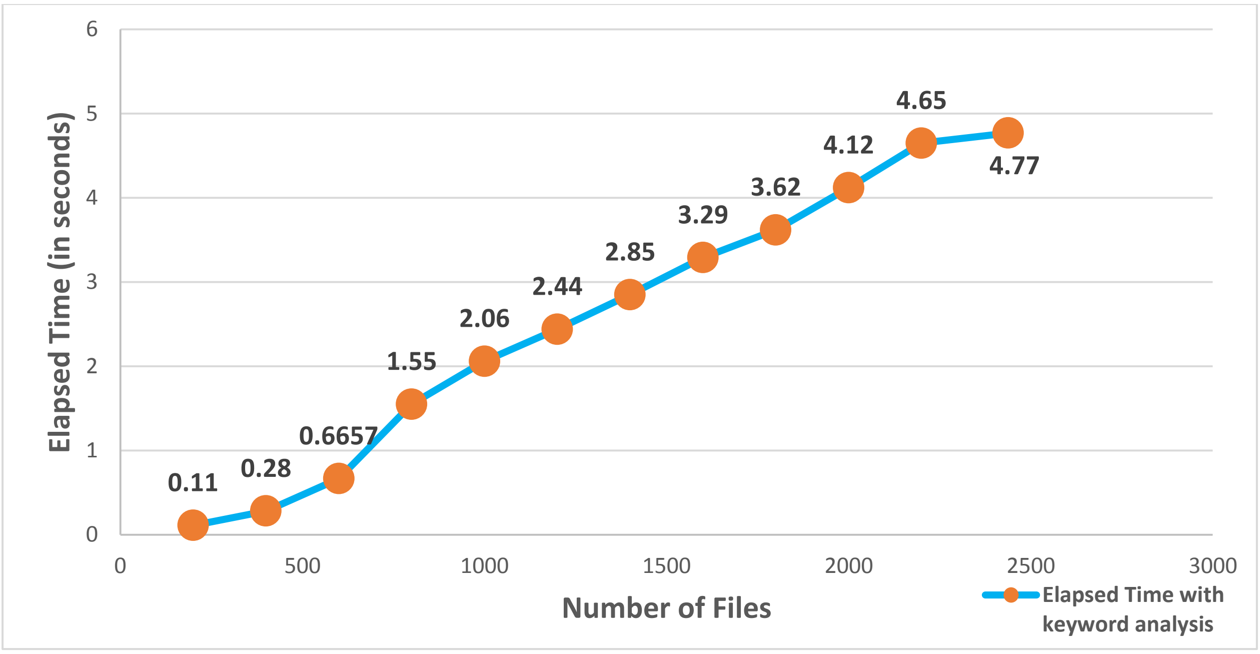


Figure 11: Computational Time for Decoding.

4.5.9 Decryption Phase

The decryption phase takes private key, IV, and the decoded fragments to return the actual plaintext file to the client. The decryption of *200* files for the dataset takes about *3.1 sec* while considering the case where all files are returned to the client, the decryption time would be *39.1 sec*. The retrieval of *2,438* for any query might not be possible but just to provide the efficiency and complexity analysis, we are assuming that a particular keyword is present all the documents. Figure 12 shows the computational time graph for the decryption phase.

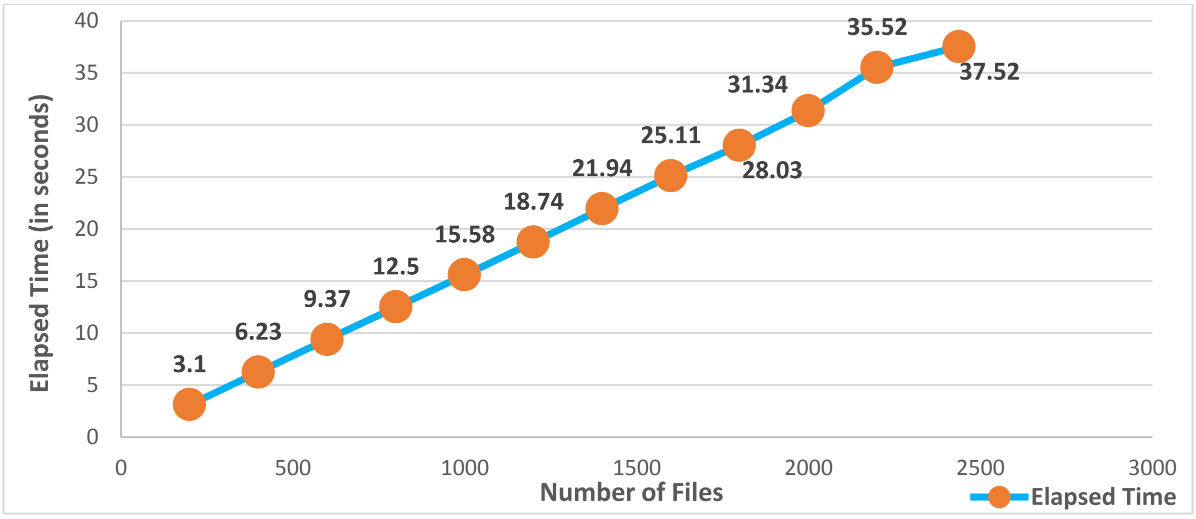


Figure 12: Computational Time for Decryption.

4.5.10 Updating Database

In order to add new documents to the existing dataset and upload them to the *CS*, the client would be required to process five major phases which include Keygen, Encryption, keyword extraction, Encoding, and Build$_{\text{Index}}$. After performing all of the above mention functions, the client is required to just add the filename and extracted encrypted keywords to the existing index table *I* and upload the updated index table to the *CS* along with the encrypted file.

For deleting any existing file in the dataset, the client would simply delete the relevant entry in the index table which include the extracted keywords along with the filename. Along with this, the encrypted file is also deleted from the dataset and the client doesn't need to perform any function on its side. The whole task of deletion is performed on the *CS* and thus all the computation would be performed there.

In order to update a file, which may include addition of text in the existing files, then the client would have two options. Either use a temp file to add the words in it, take their hashes and encrypt them and then just add those values to the index table but it would be required to encrypt the updated file, encode it and then upload it to the *CS*.

4.6 Summary

This chapter presents a detailed analysis in terms of computation and execution time of the proposed scheme. The algorithm analysis identifies the time complexity required to process each phase. The storage overhead analysis provides the analysis in terms of space that is needed to store the encrypted files, encoded files, encrypted index tables, keywords extraction, and counter parts of the some of these algorithms. The computational analysis provides the system specifications and the details of the implementation that are performed in this thesis. A discussion on the execution time of each phase is individually discussed and presented in graphical form. In Chapter 6, a conclusion of this thesis along the future directions for this thesis would be discussed. The challenges that are still present will also be discussed in detail in the next chapter.

Chapter 5

Conclusion and Future Directions

Cloud computing marks the beginning of a new era in the field of the information and communication technology as it brings with an evolution paradigm that has the potential to shift the way of computing which is never done before. The reliance of clients on a cloud server is based on a relationship of "trust". Although the cloud storage offers a great deal of solutions to the clients that include huge storage space, virtual machines, instances of GPUs etc. It provides to both individuals and corporate sector with varying charges per use. However, the security threat to the cloud is still a persistent issue that even after so much advancement has yet to be regarded as "absolutely secure". Clients and especially corporate sector are always concerned with the valuable data they are outsourcing to an entity which they can only trust. Privacy preservation and the security of the personal information on the cloud server is still an open challenge for the researchers. To overcome these issues, the encryption of the documents was considered to be a viable solution earlier on. The ability to process the encrypted documents without needing to decrypt them on either cloud server or over the communication channel has provided the users with a sense of security them enables them in believing that their valuable information is not going to misused by any non-trusted entity.

Although research in this field has been gaining more and more attention and it is advancing with time but for the privacy of documents, especially access patterns; the solutions are very limited and they are not easy to deployed considering the restrictions of computation, storage, and power on both the client and server end. Now, modern techniques like Fully Homomorphic Encryption do provide the solution for hiding the access patterns by performing search over encrypted data without the need of an index table, but the research work is still a "work in progress" and would take time to mature because the computation for huge datasets is too much to be handled by a normal user of cloud. A lightweight version of ORAM is still nowhere to be seen in the near future. Therefore, it is needed to devise newer schemes that would help in hiding the access patterns completely or at least they should protect the information that is available to the malicious users which are trying to get valuable insight about the data of the clients and corporate sector. Theoretical and experimental analysis suggests that the proposed technique provides adequate security and performance against a 3rd party adversary. In this chapter, we have presented an overview of our research, a summary of the contributions along with challenges that still exist and a future pathway in this field of study.

5.1 Research Overview

As the amount of data is increasing, a secure storage space that is always available to the users is becoming an increasing necessity. In order to store huge amount of data and at the same time providing a searching capability on this data is becoming a challenge itself given the resource constraints of devices. The cloud servers provide "storage as a service" to their customers. Thus, it becomes easier to store data on a place where the client doesn't need to use in his own storage space. But outsourcing one's confidential data requires to trust an entity that may or may not be trustworthy i-e semi-trusted. The user on the other side is losing control over its data and thus confidentiality of the data is at stake for the user. The user needs to perform normal operations over the data to keep the confidentiality of the data intact. This need of user introduced the concept of searchable encryption schemes. The current literature of searchable encryption has not given much attention to the leakage of access patterns due to which certain attacks have successfully recovered the encrypted information that is being exchanged between the client and the cloud server. The use of simulated annealing technique along with some modern probability-related algorithms, the adversaries have somehow revealed information that leaks almost all of the information about the dataset to an unknown entity observing the channel communication.

In this thesis, we have presented a privacy preserving search over encrypted document technique which tries to reduce the information that is leaked when the access patterns are leaked by performing encoding of documents and concatenating a random number to the keyword trapdoor in order to deceive the adversary into believing that he is getting the actual trapdoor for the corresponding keyword. A novice user can implement this algorithm as it provides a complete client server environment where a user is able to interact with a cloud server which is also part of the implementation of this thesis. The proposed technique is completely practical and can be used in any real time scenario.

5.2 Summary of Contributions

The research presented in this thesis discusses the security issues associated with the existing SE techniques which either try to hide the access patterns or reduce the information leaked by the leakage of access patterns. A review of the existing techniques is provided in Chapter 2. From there, it is evident that most of techniques are focused towards revealing of the access patterns by utilizing attacks like IKK attack or file count attack. The authors seem to be more interested in devising attacks that would find a loophole in the technique or use some sort of probabilistic technique to retrieve the plaintext information from the encrypted information. We have introduced a technique that would mitigate these attacks and would preserve the user's privacy by issuing a randomized trapdoor to the adversary, who is always treated as a malicious actor in any SE scheme as discussed in Chapter 3. The security analysis of the proposed technique in terms of leakage profiles defined in Chapter 3 and security provided in terms of keyword-trapdoor indistinguishability and keyword index indistinguishability. The performance analysis is given in Chapter 4, which shows that proposed technique is efficient and it can be deployed in real world applications.

5.3 Challenges and Future Work

This section discusses the challenges faced during the research of the proposed technique. The challenges will be addressed in the future work.

5.3.1 Cloud Server

With the outsourcing of data to some other entity, the users including both individuals and corporate, are actually giving that entity control over their data and the *CS* would come under a trust domain. Although major companies don't necessarily mess with the data of the users, but it is still a possibility. E-g an administrator of the cloud service provider might turn rogue or malicious. If that's the case, then the trust between the two parties would take a big hit and the users would refrain themselves from using the services of the cloud service providers because of such mishaps. To counter these types of issues is not an easy task because the it is always difficult to identify an internal threat actor. The companies do have strict policies but the types of actors still find a way to perform malicious activity including but not limited to selling of user's data on dark web. This issue is a prevalent problem for all SE schemes that are available today. The issue of a malicious cloud server is still one of the open challenges and is left as a future work.

5.3.2 Multiple Readers and Multiple Writers

Currently this scheme is limited to a single reader and single write to present a Proof of Concept for the users and researchers. The is a definite need to have a mechanism which not only multiple readers but multiple writers as well. As cloud is an open ground for all organizations because it offers services according to the pricings to any user. This means that the services are shared and accessed by different entities which can outsource their data on the cloud. Thus, it is necessary to accommodate the multi-user environment in the technique. The requirement of multi-user environment and handling of different entities based on their roles or access controls is essential. The proposed scheme needs to be altered in some ways in order to support architectures like S/M, M/S or M/M.

5.3.3 Searching without Index Table

A problem that was identified at a later stage of this thesis, is the use of index table. The index table basically gives complete access to the cloud server for it to at the very least view the table and see what’s actually inside. The problem becomes a major issue because as soon as the server would open the index table, it would recognize which files are associated with which keywords. Now, although the keywords are encrypted and the server wouldn’t get any information because decryption is not going to be possible. But according to the definition of access pattern, when an entity knows which files are returned as a result of a query, then the access pattern is leaked. This issue is not easy to resolve with the current scheme because the only probable solution is to remove the index table from the cloud server and somehow perform the searching without it. Solutions like Homomorphic encryption do provide the solution against this major issue, but as it is another domain with another solution, so we are leaving this as a future work.

References

1. M. G. Avram, Advantages and challenges of adopting cloud computing from an enterprise perspective, Procedia Technology 12 (2014) 529:534.
2. M. Nakayama, C. Chen, C. Taylor, the effects of perceived functionality and usability on privacy and security concerns about cloud application adoptions, Journal of Information Systems Applied Research 10 (2) (2017) 529–534.
3. https://www.darkreading.com/attacks-breaches/cloud-customers-faced-681m-cyberattacks-in-2018/d/d-id/1333721
4. https://www.howtowpblogging.com/popular-file-sharing-websites/
5. Bösch, Christoph & Hartel, Pieter & Jonker, Willem & Peter, Andreas. (2014). A Survey of Provably Secure Searchable Encryption. ACM Computing Surveys. 47. 1-51.

6. https://us.norton.com/internetsecurity-emerging-threats-2019-data-breaches.html
7. Gavin, G. (2016). An efficient somewhat homomorphic encryption scheme based on factorization. In International Conference on Cryptology and Network Security, pages 451–464. Springer.
8. Gentry, C. and Boneh, D. (2009). A fully homomorphic encryption scheme, volume 20. Stanford University Stanford.
9. Belland, M., Xue,W., Kurdi, M., and Chu,W. (2017). Somewhat homomorphic encryption.
10. R. Zhang, R. Xue and L. Liu, "Searchable Encryption for Healthcare Clouds: A Survey," in IEEE Transactions on Services Computing, vol. 11, no. 6, pp. 978-996, 1 Nov.-Dec. 2018, doi: 10.1109/TSC.2017.2762296
11. Riccardo Coppola and Maurizio Morisio. 2016. Connected Car: Technologies, Issues, Future Trends. ACM Comput. Surv. 49, 3, Article 46 (December 2016), 36 pages
12. Kamara, S., Papamanthou, C., and Roeder, T. (2012). Dynamic searchable symmetric encryption. In Proceedings of the 2012 ACM conference on Computer and communications security, pages 965–976. ACM
13. Curtmola, R., Garay, J., Kamara, S., and Ostrovsky, R. (2011). Searchable symmetric encryption: improved definitions and efficient constructions. Journal of Computer Security, 19(5):895–934.
14. Wang, J., Ma, H., Tang, Q., Li, J., Zhu, H., Ma, S., and Chen, X. (2013a). Efficient verifiable fuzzy keyword search over encrypted data in cloud computing. Computer science and information systems, 10(2):667–684.
15. Emil Stefanov, Marten Van Dijk, Elaine Shi, T.-H. Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. 2018. Path ORAM: An Extremely Simple Oblivious RAM Protocol. J. ACM 65, 4, Article 18 (August 2018), 26 pages.
16. Muhammad Naveed, Manoj Prabhakaran, and Carl A. Gunter. 2014. Dynamic Searchable Encryption via Blind Storage. In Proceedings of the 2014 IEEE Symposium on Security and Privacy (SP ’14). IEEE Computer Society, USA, 639–654.

17. M. Naveed, The fallacy of composition of oblivious ram and searchable encryption, Cryptology ePrint Archive, 2015.

18. M. S. Islam, M. Kuzu, and M. Kantarcioglu, Access pattern disclosure on searchable encryption: Ramification, attack and mitigation, NDSS, 2012.

19. Scott Kirkpatrick, D. Gelatt Jr., and Mario P. Vecchi. Optimization by simulated annealing. Science, 220(4598):671–680, 1983.

20. D. X. Song, D. Wagner, A. Perrig, Practical techniques for searches on encrypted data, in: Proceedings of IEEE Symposium on Security and Privacy, 2000. S&P 2000., 2000, pp. 44–55.

21. D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In Advances in Cryptology – EUROCRYPT '04, LN*CS* vol. 3027, pp. 506–522, 2004.

22. Advances in Cryptology – CRYPTO '08, Lecture Notes in Computer Science vol. 5157, D. Wagner ed., Springer-Verlag, 2008

23. M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In Advances in Cryptology– CRYPTO '07, LN*CS* vol. 4622, pp. 535–552, 2007.

24. Tahir, S (2018), Privacy Preserving Search in Large Encrypted Databases (PhD thesis), School of Mathematics, Computer Science and Engineering.

25. Miller, F. P., Vandome, A. F., and McBrewster, J. (2009). Advanced encryption standard.

26. Bao, F., Deng, R. H., Ding, X., and Yang, Y. (2008). Private query on encrypted data in multi-user settings. In International Conference on Information Security Practice and Experience, pages 71–85. Springer.

27. Stallings, W. (2006). Cryptography and Network Security, 4/E. Pearson Education India.

28. Securosis, L. (2013). Understanding and selecting a key management solution. Technical report, Technical Report, Feb.

29. Chang, Yan-Cheng & Mitzenmacher, Michael. (2004). Privacy Preserving Keyword Searches on Remote Encrypted Data. IACR Cryptology ePrint Archive. 2004. 51.

30. Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. In NDSS 2013, volume 71, pages 72–75, 2014

31. Raphael Bost. Σοφος: Forward secure searchable encryption. In SIGSAC 2016, pages 1143–1154. ACM, 2016.

32. Bost, R., Minaud, B., Ohrimenko, O.: Forward and backward private searchable encryption from constrained cryptographic primitives. In: C*CS* 2017. pp. 1465– 1482. ACM (2017)

33. Cui, S.(2019). *Secure and Efficient Searchable Encryption with Leakage Protection* [Doctoral dissertation, The University of Auckland].

34. Muhammad Naveed, Manoj Prabhakaran, and Carl A. Gunter. 2014. Dynamic Searchable Encryption via Blind Storage. In Proceedings of the 2014 IEEE Symposium on Security and Privacy (SP '14). IEEE Computer Society, USA, 639–654.

35. Panagiotis Rizomiliotis and Stefanos Gritzalis. 2015. ORAM Based Forward Privacy Preserving Dynamic Searchable Symmetric Encryption Schemes. In Proceedings of the 2015 ACM Workshop on Cloud Computing Security Workshop (C*CS*W '15). Association for Computing Machinery, New York, NY, USA, 65–76.

36. Ishai, Y., Kushilevitz, E., Lu, S., Ostrovsky, R.: Private large-scale databases with distributed searchable symmetric encryption. In: Sako, K. (ed.) CTRSA 2016. LN*CS*, vol. 9610, pp. 90–107. Springer, Cham (2016)

37. Cao, N., Wang, C., Li, M., Ren, K., Lou, W.: Privacy-preserving multi-keyword ranked search over encrypted cloud data. IEEE Trans. Parallel Distrib. Syst. 25(1) 222–233 (2014).

38. Wang, B., Song, W., Lou, W., Hou, Y.T.: Inverted index based multi-keyword public-key searchable encryption with strong privacy guarantee. In: INFOCOM 2015, pp. 2092–2100. IEEE (2015).

39. Hongwei Li, Dongxiao Liu,Yuanshun Dai, Tom H. Luan, And Xuemin (Sherman) Shen “Enabling Efficient Multi-Keyword Ranked Search Over Encrypted Mobile Cloud Data Through Blind Storage”, December 2014.

40. R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In Proc. ACM Conference on Computer and Communications Security (C*CS*), pages 79–88, 2006.

41. R. Curtmola, J. Garay, S. Kamara, R. Ostrovsky, Searchable symmetric encryption: improved definitions and efficient constructions, Journal of Computer Security 19 (5) (2011) 895–934

42. Thang Hoang, Attila Altay Yavuz, and Jorge Guajardo. 2016. Practical and secure dynamic searchable encryption via oblivious access on distributed data structure. In *ACSAC 2016*, pages 302-313. ACM, 2016.

43. G. Chen, T. Lai, M. K. Reiter and Y. Zhang, "Differentially Private Access Patterns for Searchable Symmetric Encryption," IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, Honolulu, HI, 2018, pp. 810-818.

44. Bösch, Christoph & Tang, Qiang & Hartel, Pieter & Jonker, Willem. (2012). Selective Document Retrieval from Encrypted Database. IEEE Journal on Selected Areas in Communications - JSAC. 10.1007/978-3-642-33383-5_14.

45. Chang, Yan-Cheng & Mitzenmacher, Michael. (2005). Privacy Preserving Keyword Searches on Remote Encrypted Data. 442-455.

46. Ferretti, L., Pierazzi, F., Colajanni, M., Marchetti, M.: Scalable architecture for multi-user encrypted SQL operations on cloud database services. IEEE Trans. Cloud Comput. *2*(4), 448–458 (2014).

47. Hang, I., Kerschbaum, F., Damiani, E.: ENKI: access control for encrypted query processing. *In SIGSAC 2015*, pages 183-196, ACM, 2015.).

48. Sun, W., Yu, S., Lou, W., Hou, Y.T., Li, H.: Protecting your right: attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud. In: INFOCOM 2014, pp. 226–234. IEEE (2014)

49. Shangqi Lai, Sikhar Patranabis, Amin Sakzad, Joseph K. Liu, Debdeep Mukhopadhyay, Ron Steinfeld, Shi-Feng Sun, Dongxi Liu, and Cong Zuo. 2018.

Result Pattern Hiding Searchable Encryption for Conjunctive Queries. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (C*CS* '18). Association for Computing Machinery, New York, NY, USA, 745–762

50. Hwang, R.-J., Lu, C.-C., and Wu, J.-S. (2014). Searchable encryption in cloud storage. World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering, 8(7):1080–1083] [Du, M., Wang, Q., He, M., and Weng, J. (2018). Privacy-preserving indexing and query processing for secure dynamic cloud storage. IEEE Transactions on Information Forensics and Security, 13(9):2320–2332.

51. Wu, L., Chen, B., Choo, K.-K. R., and He, D. (2018). Efficient and secure searchable encryption protocol for cloud-based internet of things. Journal of Parallel and Distributed Computing, 111:152–161.

52. Fuhr, T. and Paillier, P. (2007). Decryptable searchable encryption. In International Conference on Provable Security, pages 228–236. Springer.

53. Prasanna, B. and Akki, C. (2015). A comparative study of homomorphic and searchable encryption schemes for cloud computing.

54. Goh, E.-J. *et al.* (2003). Secure indexes. IACR Cryptology ePrint Archive, 2003:216

55. Wang, C., Cao, N., Ren, K., and Lou, W. (2012a). Enabling secure and efficient ranked keyword search over outsourced cloud data. IEEE Transactions on parallel and distributed systems, 23(8):1467–1479

56. Yanzhi Ren, Yingying Chen, Jie Yang, Bin Xie "Privacy-preserving Ranked Multi-Keyword Search Leveraging Polynomial Function in Cloud Computing" Globecom Communication and Information System Security Symposium 2014.

57. H. Huang, J. Du, H. Wang and R. Wang, "A Multi-keyword Multi-user Searchable Encryption Scheme Based on Cloud Storage," 2016 IEEE Trustcom/BigDataSE/ISPA, Tianjin, 2016, pp. 1937-1943.

58. D. D. Rane and V. R. Ghorpade, "Multi-user multi-keyword privacy preserving ranked based search over encrypted cloud data," 2015 International Conference on Pervasive Computing (ICPC), Pune, 2015, pp. 1-4.

59. Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. In SIGSAC 2015, pages 644–655. ACM, 2015.

60. Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In USENIX Security 2016, pages 707–720. USENIX Association, 2016.

61. David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In SIGSAC 2015, pages 668–679. ACM, 2015

62. Samanthula, Bharath Kumar & Jiang, Wei & Bertino, Elisa. (2014). Privacy-Preserving Complex Query Evaluation over Semantically Secure Encrypted Data. 400-418.

63. Curino, C., Evan, J. P. C., Popa, R. A., Malviya, N., Wu, E., Madden, S., Balakrishnan, H., and Zeldovich, N. (2011). Relational cloud: A database-as-a service for the cloud. In 5th Biennial Conference on Innovative Data Systems Research, pages 235–240.

64. Wang, J., Wang, J., Yu, N., and Li, S. (2013b). Order preserving hashing for approximate nearest neighbor search. In Proceedings of the 21st ACM international conference on Multimedia, pages 133–142. ACM.

65. Boldyreva, A., Chenette, N., Lee, Y., and O'neill, A. (2009). Order-preserving symmetric encryption. In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pages 224–241. Springer.

66. S. Tahir, M. Rajarajan and A. Sajjad, "A ranked searchable encryption system for encrypted data hosted on the Public Cloud," 2017 International Conference on Information Networking (ICOIN), 2017, pp. 242-247.

67. Wang J., Yu, N., and Li, S. (2013b). Order-preservation hashing for approximate nearest neighbor search. In Proceedings if the 21st ACM international on Multimedia, pages 133-142. ACM.