Precision and Privacy Preserving Multi-Keyword Search over Encrypted Data

Thesis

Submitted in partial fulfilment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

D VENKATA NAGA SIVA KUMAR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

SURATHKAL, MANGALORE - 575 025

June, 2020

DECLARATION

by the Ph.D. Research Scholar

I hereby declare that the Research Thesis entitled **Precision and Privacy Preserving Multi-Keyword Search over Encrypted Data** which is being submitted to the **National Institute of Technology Karnataka, Surathkal** in partial fulfilment of the requirements for the award of the Degree of **Doctor of Philosophy** in Department of Computer Science and Engineering is a bonafide report of the research work carried out by me. The material contained in this Research Thesis has not been submitted to any University or Institution for the award of any degree.

D Venkata Naga Siva Kumar, CS15FV04

Department of Computer Science and Engineering

Place: NITK, Surathkal.

Date: June 12, 2020

CERTIFICATE

This is to certify that the Research Thesis entitled **Precision and Privacy Preserving Multi-Keyword Search over Encrypted Data** submitted by **D Venkata Naga Siva Kumar** (Register Number: 155119 CS15FV04) as the record of the research work carried out by him, is accepted as the Research Thesis submission in partial fulfilment of the requirements for the award of degree of **Doctor of Philosophy**.

Dr. P. Santhi Thilagam
Research Guide

(Signature with Date and Seal)

Chairman - DRPC
(Signature with Date and Seal)

To my Grand Father (late), Grand Mother, and Parents
for their love and support.

ACKNOWLEDGEMENTS

It is a great pleasure to thank the people who have supported and helped me so much throughout the period of my research. First and foremost, I would like to express my sincere gratitude to my research supervisor Dr. P. Santhi Thilagam, Professor of the Department of Computer Science and Engineering, for constantly supporting me throughout the journey of my research with her knowledge, experience, and patience. I owe my deepest gratitude to her for the valuable guidance, timely suggestions, and advices. Beyond the subject area of this thesis, I have learnt a lot from her that, I am sure, will be useful in the later stages of my career as well as my life.

I would like to extend my sincere thanks to the members of my research progress assessment committee Dr. Alwyn R. Pais, and Dr. Dr. R. Madhusudhan, for their insightful feedback and constructive suggestions throughout the period of my research work. I also extend my sincere thanks to the entire faculty of the Department of Computer Science and Engineering for rendering their help and support throughout the period of my research. Nevertheless, I am also grateful to the technical and administrative staff of the department for their timely help and co-operation to carry out the research work.

I would like to express my sincere thanks to NITK for providing the necessary infrastructure and facilities required for the successful completion of this research work. I would also like to thank the Ministry of Electronics and Information Technology (MeitY), Government of India for providing me an opportunity to pursue Ph.D. through Visvesvaraya Ph.D scheme. I also thank Rashmi, Deepthi and Sankhalika for the useful discussions and the support they extended during this period.

Special thanks are due to Ambikesh. G, who helped me a lot in improving my writing skills during the tenure of my PhD. I would also like to thank Dr. Deepa G, Dr. Bindu P V, Amith Praseed, K. Srinivasa, Uma Priya, Bheemappa, Raghavan and Srinu for their help extended during the course of present study.

I am deeply indebted to my late grand father, grand mother, parents, sister, brother-in-law and maternal uncle for being with me all through, and encouraging me for the successful completion of the research work. Thanks to my friends and family members for their support.

Finally, I thank the Almighty for granting me the health and strength for carrying out this research work.

D Venkata Naga Siva Kumar

ABSTRACT

A growing number of data owners are increasingly using cloud storage services of various Cloud Service Providers (CSPs) for storing and managing their information/documents. The cloud storage services provide numerous benefits to the data owners that include cost savings, greater reliability, ubiquitous access, and better performance. In spite of these benefits, the stored data at the remote cloud servers are vulnerable to the attacks initiated by the untrustworthy CSPs. Such data risks and associated privacy concerns have recently been in the limelight from revelations of extensive surveillance by several national security agencies and other government entities. The primary concerns of storing documents at cloud servers are confidentiality and privacy due to the loss of control over who accesses and manages the outsourced documents. In order to address these concerns, sensitive data is required to be outsourced in encrypted form to the cloud servers. Although the encryption guarantees confidentiality, it makes the retrieval process more complex.

Searchable Encryption (SE) is a technique that guarantees confidentiality and privacy by storing documents in encrypted form at the cloud servers and allows search over encrypted data without decrypting it. In SE, the data owners store their documents, and the corresponding indexes in encrypted form, and the data users retrieve the documents by sending encrypted queries (trapdoors) to the cloud server. Despite its privacy and confidential guarantee, the privacy of trapdoor keywords and index keywords could be compromised due to the information leakages that are caused by the vulnerabilities in the adopted schemes used for encrypting indexes and queries. The information leakages include frequency of ciphertext values, rank-order information, and search pattern. The cloud servers exploit these leakages to infer plaintext information through various information disclosure attacks such as frequency analysis attack, rank-order exploitation attack, and scale analysis attack. Hence, this work aims at preventing these leakages and thereby mitigates the attacks.

The cloud server uses frequency analysis attack to infer index keywords based on the frequency leakage of ciphertext values (repetition of the same encrypted keywords' relevance scores) in indexes. This leakage occurs due to the insufficient randomness in the order preserving encryption (OPE) schemes that are used for encrypting keywords' relevance scores in indexes. The existing OPE schemes leak frequency information when there are same plaintext scores for two or more keywords within the same document. In this work, an Enhanced One-to-Many order-preserving mapping technique

is developed with improved randomness to mitigate the frequency leakage. The experimental study confirms that the proposed technique reduces not only the frequency leakage of keywords but also the co-occurring keywords.

The cloud server returns the relevant documents in descending order for a given trapdoor based on the ranks of the documents. However, the cloud server uses the rank-order exploitation attack to infer the plaintext information of frequently issuing query keywords or frequently occurring keywords of the dataset based on the rank information leakage. Scale analysis attack also occurs when the users issue the same trapdoor again and again to retrieve the same documents. This attack enables the cloud server to infer plaintext keywords of trapdoors based on the search pattern leakage, which can be determined from the given set of trapdoors. The existing approaches prevent search pattern leakage by adding random keywords to a list of query keywords in a trapdoor generation approach, but precision gets affected due to the random keywords. These approaches cannot prevent rank-order information leakage completely since the random values of random keywords follow the distribution of actual keywords' relevance scores. Therefore, it is highly essential to prevent these attacks by preserving the privacy of both rank information and the search pattern. In this work, a Pseudo-Ranking approach is developed to address this issue with the help of two servers, i.e., cloud server (CS) and the intermediate server (IS). The CS assigns pseudo-ranks to the documents instead of actual ranks, and the IS would nullify the impact of random keywords of a trapdoor for achieving higher precision. The experimental results confirm that the proposed approach preserves the privacy of both rank information and search pattern without affecting precision.

Besides preventing these attacks, it is also essential to provide the latest relevant documents to the users to enable them to choose timely decisions based on updated information. To provide the latest relevant documents, there should be a provision in SE for allowing the data owners to perform the dynamic updates efficiently over the existing encrypted indexes. The existing tree-based indexing schemes cannot perform dynamic operations efficiently since the trees' size is larger in terms of height and breadth. This causes a delay in performing dynamic updates and retrieving top-k relevant documents. In this work, a Max-heap tree based index structure is developed to address this issue. The experimental results of the proposed tree index confirm that it improves the time efficiency of top-k document retrieval and dynamic updates.

Keywords: Searchable Encryption, Frequency of ciphertext values, Rank-orden information, Search pattern, and Dynamic updates.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviations	Expansion
ABE	Attribute Based Encryption
BS	Boolean Search
CS	Cloud Server
CPA	Chosen Plaintext Attack
CSPs	Cloud Service Providers
DF	Document Frequency
DO	Data Owner
DU	Data user
ERI	Encrypted Randomized Index
FHE	Fully Homomorphic Encryption
FHEI	Fully Homomorphic Encryption over Integers
FOCPA	Frequency of Ordered Chosen Plaintext Attack
GDFS	Greedy Depth First Search
IDF	Inverse Document Frequency
HVE	Hidden Vector Encryption
IS	Intermediate Server
KBB	Keyword Balanced Binary
KRB	Keyword Red Black
LSH	Locality Sensitive Hashing
MKBS	Multi-Keyword Boolean Search
MKRS	Multi-Keyword Ranked Search
OPE	Order Preserving Encryption
OXT	Oblivious Cross Tags
PBF	Privacy Bloom Filter
PE	Paillier Encryption
PEKS	Public Key Encryption with Keyword Search
PIR	Private Information Retrieval
RI	Randomized Index
RFC	Request for Comments
RS	Ranked Search
SCU	Secure Computing Unit
SE	Searchable Encryption
SHA	Secure Hash Algorithm

Abbreviations	Expansion
SKBS	Single Keyword Boolean Search
SKRS	Single Keyword Ranked Search
SSE	Searchable Symmetric Encryption
TF	Term Frequency
VB	Virtual Binary

CHAPTER 1

INTRODUCTION

Due to the proliferation of web, mobile, cloud, IoT and connected technologies, the vast amounts of data are being generated due to many activities such as business transactions, mobile devices, sensors, social media, high definition videos, share markets, and web clicks. This generated data may belong to public or private organizations or even individuals. The generation of such huge data has become indispensable for the organizations and individuals to use third-party cloud storage services. Cloud storage has become a prevalent storage option in recent years, where users can store their data of any size and share it with others. Many individuals, government agencies, medium, and large scale organizations are increasingly using cloud storage services of various Cloud Service Providers (CSPs) like Google Cloud Platform, Amazon S3 (Simple Storage Server), Microsoft Azure, DropBox. Examples of stored data include medical records, personal tax documents, government policy documents, corporate personal communications, and financial data. The data owners store their data at cloud servers to avoid the cost of both upfront investment involved in setting up their own data center, and its maintenance and also to get other benefits such as ubiquitous access, and unlimited access to hardware/software resources. Despite such benefits provided by the cloud, the major challenge that remains is the concern over the confidentiality and privacy of uploaded data onto the cloud servers (Kamara and Lauter 2010).

The confidentiality and privacy concerns hinder the adoption of cloud services by many organizations in real world due to the following reasons:

1. The data owners lose complete control on their data once uploaded onto the cloud servers (Yan et al. 2017). They cannot control who can access and retrieve once uploaded. The cloud servers will try to know about about the information they have stored.

2. CSPs can misuse the stored data for their own business benefits and even can leak this data to the business competitors of actual data owners (Cash et al. 2015; Shahzad 2014).

Therefore, the data owners' unencrypted data stored at the remote cloud server would be vulnerable to the attacks initiated by the untrustworthy CSPs (Subashini and Kavitha 2011). CSPs usually enforce users' data security through mechanisms like firewalls and virtualization. However, these mechanisms do not protect users' privacy from the CSPs themselves since they possess full control of the system hardware and lower levels of the software stack, and, therefore, can misuse the stored data for their personal or financial benefits. At cloud servers, there may exist disgruntled, profiteered, or curious administrators and other employees who can access users' sensitive information for unauthorized purposes. For example, Facebook corporation has allowed its US users' data to be used by the Cambridge Analytica firm, which later used it to influence US voters towards their favourable candidate in the 2016 US presidential elections[1]. In this context, both the confidentiality of stored documents and the privacy of users could be compromised. Confidentiality (Tari 2014) refers to the protection of outsourced data from being shared with unauthorized users other than the intended users. While the privacy refers to the protection of data from being misused and leaked (Dawoud and Altilar 2017; El Makkaoui et al. 2016). Several surveys indicate that the confidentiality and privacy concerns are significant barriers to the adoption of cloud storage services. To address these concerns, it is required to outsource the data in encrypted form, but it makes the retrieval process more complex. Searchable encryption is a technique (Song et al. 2000) that guarantees confidentiality and privacy by storing documents in encrypted form at the cloud servers. It also enables search over encrypted

[1] https://www.scmp.com/news/world/united-states-canada/article/2139327/explainer-how-cambridge-analytica-exploited-facebook

documents without decrypting with the help of searchable encrypted documents, which corresponds to the documents. In general, confidentiality could be achieved through encryption, which in turn assures privacy so that the cloud servers cannot come to know anything about the plaintext data from encrypted data and cannot misuse the stored data for unauthorized purposes. It is only the data owners or users with whom they share the secret keys can perform search and decrypt the retrieved documents. Therefore, with the help of searchable encryption, cloud servers cannot come to know any information about the stored data. The guarantee of confidentiality and privacy depends on the type of encryption approach that would be used for encrypting indexes and the documents. The encryption approaches should be chosen so that users can search over encrypted documents, and they do not leak any information to the cloud servers while searching and retrieving required documents.

1.1 SEARCHABLE ENCRYPTION

Searchable Encryption (SE) is a technique that enables the cloud server to perform search operation over encrypted data without knowing anything about plaintext information. There are two types of searchable encryptions such as SSE (Searchable Symmetric Encryption) (Song et al. 2000) and PEKS (Public Key Encryption with Keyword Search) (Boneh et al. 2004a) or it is also called as ASE (Asymmetric Searchable Encryption). SSE approaches use the same secret key for encryption and decryption, and they are computationally efficient because they involve arithmetic operatins such as addition and multiplication in encryption and decryption. While the ASE approaches use different key for encryption and decryption, and they are computationally expensive because they involve exponentiations and pairing operations in encryption and decryption. SSE approaches (Yunling WANG 2016) are primarily meant to enable data access only by data owners with the corresponding secret key, which is used to generate encrypted indexes and encrypted documents. To enable the data access by other users in SSE, the data owners have to provide them with either the secret keys for encrypting queries or trapdoors that can be sent to the cloud servers to retrieve the documents. Broadcast encryption should be used along with the SSE to enable search by other users (Curtmola et al. 2006). On the other hand, PEKS approaches (Boneh et al. 2004a) allow any user

to search over the encrypted data of multiple owners with his/her corresponding secret key. The PEKS with encryption schemes such as the Ciphretext-Policy Attribute-based encryption (Wang et al. 2014b) enables the data owner to allow search operation by multiple users. In SE, whether it is SSE or PEKS, the cloud server adopt either Boolean or Ranked search approaches to process the trapdoors and return the documents to the users. In Boolean search approaches, the documents are returned only if they contain one or more query keywords of the trapdoor. whereas in ranked search approaches, the cloud server returns top-k documents in decreasing relevance order of the users' trapdoors.

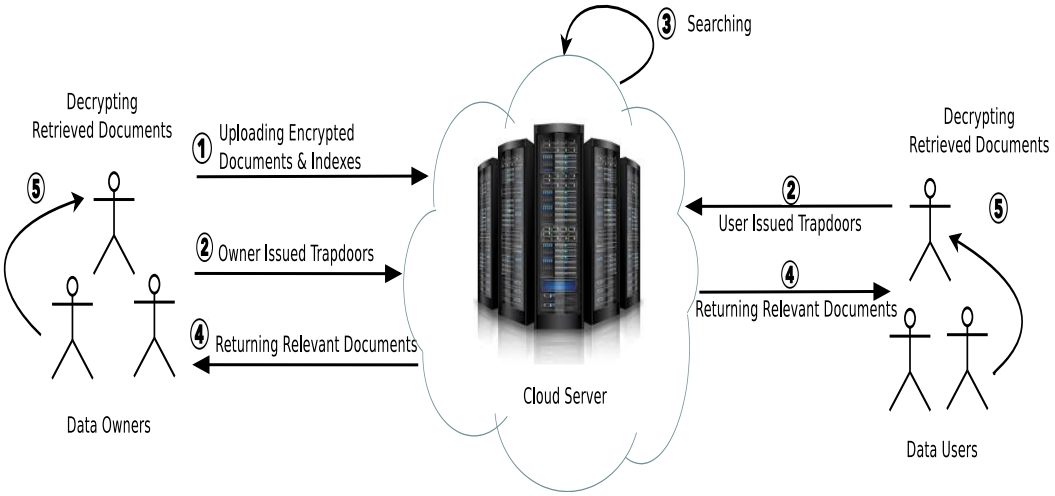


Figure 1.1: Different stages of searching over encrypted data.

Searching over Encrypted Data: The process of searching over encrypted documents is shown in Figure 1.1. It depicts the sequence of steps to be performed by the data owners and users to enable the cloud server to perform search and retrieve the encrypted documents of users' interests through trapdoors.

The data owners are first required to generate plaintext indexes for all his/her plaintext documents. There are two types of indexing schemes, such as Forward indexing, in which there exists an index for each document (Jin et al. 2016) and Inverted indexing in which, for each keyword, it contains a pointer to document identities that they contain (Curtmola et al. 2006). Each of them can be represented by different data structures, that results in various indexing methodologies, i.e., tree-based indexing (Xia et al.

2016), Bloom filters (Bloom 1970), and Bucketization (Agrawal et al. 2004). These indexing techniques aim at performing search operations that are either linear or sub-linear to the total number of the documents stored. These indexes are then encrypted with any one of the encryption schemes,i.e., Deterministic Encryption (DE) (Bellare et al. 2007, 2008; Boldyreva et al. 2009), Non-deterministic encryption (Yang et al. 2017), Functional encryption(FE) (Boneh et al. 2011). To return the documents in decreasing relevance order of the given trapdoors, the uploaded encrypted indexes, also referred to as searchable indexes, should include ranking information, i.e., keywords' relevance scores, which are determined by using various keyword weight measures such as Term-Frequency (TF), and Term-Frequency-Inverse Document Frequency (TF-IDF) (Lee et al. 1997). The keyword relevance scores also need to be encrypted by using any one of the encryption schemes such Order Preserving Encryption (OPE) (Boldyreva et al. 2009), Fully Homomorphic Encryption (FHE) (Wang et al. 2015; Wu 2015), and schemes. These schemes help the cloud server to determine ranks of the documents for given trapdoor from the encrypted scores. Each of these encryption schemes and indexing techniques have certain advantages and limitations with respect to time, privacy, and precision. Hence, they are selectively chosen for encrypting the indexes especially as per the data owners' privacy and precision requirements. After the indexes are encrypted, the data owners then encrypt the plaintext documents using either a public key or secret key encryption schemes with a sufficiently large key. The data owner then uploads both the encrypted documents and their corresponding searchable indexes onto the cloud servers (Goh et al. 2003).

The data owners or the authorized data users can issue trapdoors, i.e., queries in encrypted form along with a value 'k' to the cloud server to send only the top-k relevant documents. Upon receiving the trapdoor, the cloud server uses various similarity measures such as Coordinate matching (Witten et al. 1994), Cosine similarity (Manning et al. 2008), Jaccard coefficient (Wong and Kim 2013), Locality sensitive hashing (Zhang et al. 2016a), and Inner product operation (Li et al. 2017) to determine the scores, i.e., ranks of the documents for a given trapdoor. The cloud server then sorts the documents in decreasing order and sends top-k of them to the data users or owners.

Returning the top-k relevant documents reduces the network traffic and also fulfills the users' information needs quickly.

1.2 LEAKAGES

In SE, the data owners hold complete control on the security keys used for encrypting indexes and documents. Therefore, neither the adversaries nor the cloud servers can directly obtain any information about the documents since they have no access to the keys. However, there are two aspects, wherein information leakages are more likely to occur, i.e., while generating trapdoors and while processing trapdoors using Boolean and Ranked search approaches. Therefore, although the indexes and trapdoors are in encrypted form, the cloud servers could still infer plaintext information due to the leakages caused by the vulnerabilities in the adopted encryption schemes of indexes and queries. The possible leakages in SE are given in Figure 1.2. The leakages in SE are categorized into two types (Kumar and Thilagam 2019a): 1) Direct ciphertext leakages, these leakages are directly extracted from the available encrypted information, i.e., uploaded encrypted indexes, and trapdoors. The leakages involved in this category are Distribution information, Association information and Size pattern. The distribution information can be obtained by observing the frequency of encrypted ranking information, i.e., the repetition of same encrypted TF or TF-IDF values in indexes (Cash et al. 2015). This leakage is caused especially when TF or TF-IDF values are encrypted by deterministic encryption schemes, which encrypts the same plaintext to the same ciphertext. Association information leakage can be obtained while processing trapdoors with the help of encrypted indexes, e.g., the cloud server can come to know which document contains which keywords of the trapdoor (Zerr et al. 2008) and how many documents have common keywords of the trapdoor. Size pattern leakage can be obtained by observing the number of keywords present in indexes of the documents and trapdoors (Goh et al. 2003). This is caused when sizes of each index and each trapdoor are different. 2) Indirect ciphertext leakages, which are not directly dependent on the available encrypted encrypted information, but on the user information requirements. The leakages of this category are Rank-order information, Search pattern and Access pattern. Rank-order information is leaked when the cloud server returns the documents

in descending order as per their scores, i.e., ranks. Search pattern leakage information can be obtained when a user sends the same trapdoor again and again. Access pattern leakage refers to outcomes (search results) of the trapdoors. The same sequence of documents may be returned to the same trapdoors. The leakages of these two categories can be exploited by the cloud server for inferring plaintext through various attacks which are explained in Section 1.3 .

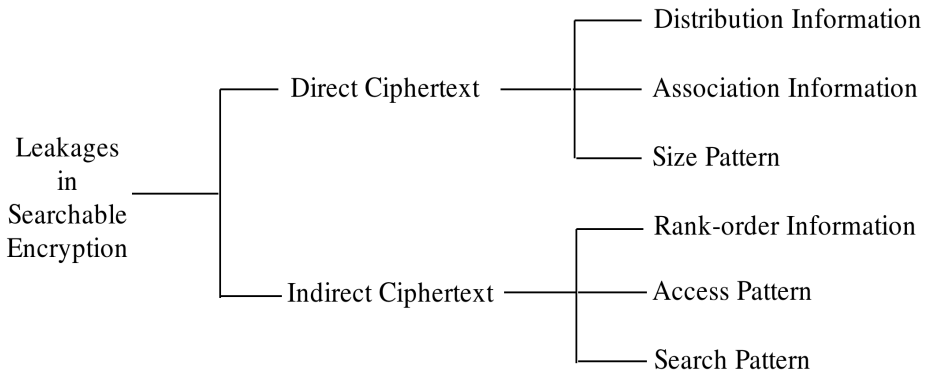


Figure 1.2: Leakages in searchable encryption.

1.3 THREAT MODEL AND ATTACKS

In this section, the formal definitions of privacy, confidentiality, and security in the context of searchable encryption are provided first, and then a threat model and its various possible attacks are presented.

Privacy: It refers to the protection of sensitive data from being leaked and misused (Dawoud and Altilar 2017; El Makkaoui et al. 2016) while enabling the cloud server to search over encrypted data.

Confidentiality: It refers to the protection of data owners' outsourced data from being known by the cloud server and thereby disclosing it with unauthorized users other than the intended users (Tari 2014).

Security: It refers to guaranteeing Confidentiality, Integrity, Authenticity, and Non-repudiation of outsourced data at the cloud server while ensuring the availability of data for access by the data users (Tang et al. 2016; Tari 2014). This work focusses on assuring confidentiality and privacy.

Communication channels: All communication between the cloud server and the data owner/user is done using a public channel (Diffie and Hellman 1976; Wang et al. 2014a). Although it is public, the adversaries (both external adversaries and the cloud server) do not get any information from it because all communication between the user and cloud server using this channel is done in encrypted form. A secure channel is assumed to exist between the data owner and the data users for ensuring secure communication with the help transport layer security protocol. The data owners use the secure channel for sharing the secret keys that are necessary for generating trapdoors and decrypting the retrieved documents (Yu et al. 2010).

Honest-but-Curious (HBC) Threat Model: In this threat model, the cloud server is considered as an adversary and its behavior is assumed to be honesh-but-curious (Ghosh Ray et al. 2018; Wang et al. 2012). It means that the cloud server follows the protocols as agreed between the cloud server and data owner with respect to storing, searching and retrieval of the documents for the given users' trapdoors. However, it attempts to infer as much information as possible from the available information. The available information includes uploaded encrypted indexes, encrypted documents, previous trapdoors, and their corresponding search results. In this honest-but-curious threat model, the cloud server is also assumed to have some background knowledge about the dataset, i.e., knowledge about what data is stored, and distribution information of specific keywords of the dataset, users search behavior.

In this threat model, the cloud server initially mounts various attacks to exploit the information leakages (explained in Section 1.2) that are observed in the available information for obtaining statistical information. The cloud server then relates this statistical information to the background knowledge information (e.g., keyword specific distribution information) for inferring plaintext information. There exist information disclosure attacks that exploit the leakages to infer sensitive information from the encrypted data. The objective of these attacks is to infer as much plaintext information as possible by exploiting the leakages. The details about each of these attacks are explained as follows.

1. **Frequency analysis attack:** This attack aims at deducing the index's plaintext

keyword from the encrypted keywords' relevance scores, e.g., TF values of keywords present in the uploaded encrypted index (Naveed et al. 2015). This attack is based on the intuition that for some specific keywords, the distribution information of plaintext TF values would be the same as the distribution information of encrypted TF values. The cloud server infers plaintext information from the encrypted TF values by relating the frequency information of uploaded dataset to the frequency information of publicly available dataset (auxiliary dataset) or even by guessing the most probable occurring keywords of the uploaded dataset by observing the frequency information. This attack is more likely to succeed with the help of some background knowledge of the uploaded dataset. The background knowledge could be about what data the data owners stored at the cloud server, and what probable keywords that are most likely to occur that the cloud server could guess easily. For example, assume that the data owners have stored Request for Comments (RFC 2016) dataset in encrypted form at the cloud server. It is widely known that this dataset is all about how information can be communicated from one host to another in the internet using various protocols of "computer network". Then, it can be assumed that some of the specific keywords like "computer," "network" and "communication" may appear in most of the documents of this dataset. The cloud server then infers one or all of these specific keywords by generating histograms (showing distribution information) for all encrypted TF values of each encrypted index keyword. The cloud server observes the histograms and identifies the ones whose frequency (i.e., the repetition of the same encrypted TF value) is much higher than the frequency of other encrytped TF values in other histograms. The corresponding encrypted keywords of those histograms are noted. These encrypted keywords may be related to one or all of those specific keywords. The cloud server thus infers plaintext keywords through frequency analysis attack. Frequency information can be exploited to infer plaintext information through correlation attack (Bindschaedler et al. 2018; Durak et al. 2016). Therefore, it is necessary to prevent the leakage of frequency information from the encrypted TF values. The existing OPE schemes are prone

to frequency analysis attack, especially when two or more keywords have the same TF value in one or more documents of the dataset. Hence, there is a need for a privacy preserving encryption scheme that either prevents or mitigates the leakage of frequency information than the frequency leakage of the existing OPE schemes. This research work aims to mitigate this attack by reducing frequency leakage.

2. **Rank-order exploitation attack:** For any given trapdoor, the cloud server returns the documents in descending order as per their scores, i.e., ranks. The users' information needs can be met quickly with the by going through top k documents only (e.g., k=10 in Google search results) rather than going through all the returned documents. However, when two or more related trapdoors are provided in a given time period, this rank order information of documents can be exploited by the cloud server to infer the plaintext information of frequently issuing query keywords or frequently occurring keywords by observing the difference in scores of the returned documents for the given two or more trapdoors. For example, assume that there exist 5 documents $\{d_1, d_2, d_3, d_4, d_5\}$ and three keywords {"computer", "network", "communication"}, which are the most frequently occurring keywords in these documents. There exists a user who issues two related trapdoors, T_1 and T_2, both of which differ by just one keyword. T_1 contains query keywords {"computer", "network"} and T_2 contains {"computer", "network", "communication"}. In addition, assume that documents d_3, d_4, d_5 contain the keywords {"computer", "network"} while the documents d_1, $d2$ contain all the three occurring keywords. When trapdoors T_1 and T_2 are issued, the cloud server returns the top-5 documents with their scores for T_1 and T_2 are $\{d_1 : 2, d_2 : 2, d_3 : 2, d_4 : 2, d_5 : 2\}$, and $\{d_1 : 3, d_2 : 3, d_3 : 2, d_4 : 2, d_5 : 2\}$ respectively. The cloud server observes the difference in scores of the documents for both the trapdoors and relates it to its background knowledge of the database (Rahman et al. 2015). This difference is easily observable as the scores of the documents for trapdoor T_2 is higher or equal to each document's score of T_1. From this difference of scores, relativenss of trapdoors can estimated and the doc-

uments that have high scores tend to convey that they match one or more extra keywords of trapdoor T_2 than T_1. These extra keywords are the most likely to be the frequently occurring keywords of the dataset or frequently searching query keywords. Thus, the rank-order exploitation attack allows the cloud server to infer plaintext keywords of the trapdoors or most occurring keywords of documents (Rahman et al. 2014). The existing approaches cannot prevent the rank-order information leakage (Guo et al. 2019; Xia et al. 2016). Hence, it is important to prevent the exploitation of rank order information, i.e., higher rank privacy is required. This research work aims to preventing this attack by assigning pseudo-scores to documents instead of actual scores so that the cloud server does not come to know the actual scores of the documents.

3. **Scale analysis attack:** The cloud server uses this attack to infer plaintext keyword of trapdoor (Cao et al. 2014). The success of this attack depends on the the leakage of the search pattern, which could be determined by comparing one trapdoor information with other trapdoors or may also be determined from the similarity scores of the documents because same scores are assigned to the same documents for the same trapdoors. Search pattern leakage conveys whether two or more issued trapdoors are generated from the same query keyword set or not. If they are same, then the search pattern is leaked. Then the cloud server infers trapdoor's plaintext keyword by defining the equivalence relationship among the similarity scores of the documents for the given trapdoors. With this equivalence relationship, the cloud server deduces two possible values of document frequency (DF). The DF of keyword refers to the number of documents that contain the given keyword. The cloud server then relates the inferred DF values of keywords to the DF values of frequntly occurring keywords of the dataset. Thus, the cloud server infers the plaintext keyword of the trapdoors through this attack. This attack is more likely to succeed when the inferred DF value of the keyword is either equal to the size (number of documents) of the dataset or more than half of the entire dataset size. For example, consider the dataset related to RFC (RFC 2016) and keywords set {"computer", "network"}. In addition, as-

sume that one of the keyword appears in all the documents and the other one does not appear in any document. When a data user issues trapdoors T_1 and T_2 containing keywords {"computer",} and {"computer", "network"} respectively, the cloud server can infer one of the trapdoor's plaintext keyword by inferring DF value from the equivalence relationship, which is tested on documents' similarity scores of both the trapdoors. The cloud server then relates the derived DF value to the DF values of some specific keywords. Thus, it derives the plaintext keywords of the trapdoors. Hence, it is required to prevent the search pattern leakage information. The existing approaches prevent search pattern leakage by adding random keywords to a list of query keywords in a trapdoor generation approach, but precision is affected with those random keywords. This research work aims at preventing this attack by nullifying the impact of added random keywords.

4. **Access pattern exploitation attack (or also referred to as IKK Attack):** The cloud server uses this attack to infer the plaintext keyword of the trapdoors (Cash et al. 2015). The success of this attack depends on the leakage of access pattern, which is exploited to infer the plaintext keyword of the trapdoor. The access pattern conveys whether the sequence of the returned documents for the given two or more trapdoors are same or not. If they are same, the cloud server assumes that the data user is accessing those documents frequently, so he/she issues the same trapdoor. Plaintext keyword of trapdoor can be known by using IKK attack by exploiting the leakage of access pattern. For example, let us assume that a set of documents stored at cloud server are about "computer science" related information. The keyword of the trapdoor can be inferred using this attack with the help of some background knowledge, i.e., keyword distribution information of stored documents. Based on the documents stored, let us assume a scenario, wherein the keywords 'computer', 'academic', 'network' are more likely to appear in most of the given documents. Let us also assume that the data user sends trapdoors for these keywords interspersed among other keywords in a particular period. After all communications are finished between the data user and the cloud server, the cloud server sees a list of trapdoors and the corresponding returned documents in

which those keywords present. The cloud server could then calculate the probability of appearance of any two of these keywords in any document by noticing the number of same documents returned for those corresponding trapdoors. By observing these probabilities, the adversary can isolate these three trapdoors representing keywords 'computer', 'academic', 'network' because of their high probabilities. Further, it is also possible that the pair of keywords 'computer', and 'network' are more likely to appear together than the probabilities of the pair of keywords 'computer', and 'academic', or 'academic' and 'network'. This observation enables the cloud server to uniquely identify the trapdoor of the keyword 'academic'. Furthermore, if the cloud server knows the trapdoor for 'computer', it is also possible for it to know the trapdoor for 'network'. Therefore, it is quite evident that the access pattern leakage information along with some background knowledge about the dataset is sufficient enough for the cloud server to infer the plaintext information of the trapdoors. Hence, it is required to return a unique set of documents for the given trapdoors to prevent the access pattern leakage information and thus to prevent IKK attack. But efficiency, precision, and security contradict each other; therefore, most of the existing SE schemes leak access pattern due to efficiency reasons. SE schemes using Oblivious RAM (Goldreich and Ostrovsky 1996a), Blind storage (Wang et al. 2015), Private Information Retrieval (PIR) (Boneh et al. 2007) can accomplish ideal security (no information it leaks including access pattern) to the cloud server. However, these approaches are practically infeasible to adapt in the real-world due to the poly-logarithmic computation and communication overheads. Therefore, a more practical and feasible SE approaches are required to prevent access this attack. The proposed research work cannot prevent this attack but it avoids exploitation of access pattern information.

Of all the attacks discussed above, the proposed research work aims to mitigate frequency analysis attack by mitigating frequency leakage, prevents rank-order exploitation attack by preventing the rank-order information leakage and also prevents scale analysis attack by preventing search pattern leakage.

1.4 PRIVACY REQUIREMENTS

As described in the above Section 1.3, the cloud servers infer plaintext information from the encrypted information through various attacks based on the leakages. Hence, it is required for SE schemes to prevent the cloud server from the exploitation of leakages. The SE schemes are required to meet both the following security and search accuracy criteria to ensure the full confidentiality and privacy of stored documents at the cloud server while allowing users to retrieve documents of their interest.

Data Confidentiality of Documents: The content of the uploaded encrypted documents should not be revealed to the cloud server. Almost, existing SE schemes (Cash et al. 2013; Karame et al. 2011) used semantically secure encryption schemes like AES with CTR or CBS mode for encrypting the documents (Bellare and Goldwasser 2008). Hence, they preserve the data privacy. The privacy of the documents' content is said to be guaranteed by a SE scheme if it satisfies the data privacy.

Index privacy: In Boolean search approaches, the indexes contain the document identities and the corresponding encrypted keywords that they contain. The indexes in Ranked search approaches also include the keywords' relevance scores, i.e., Term Frequency (TF) or Term Frequency-Inverse Document Frequency (TF-IDF) values for each keyword of every document of entire dataset. In SE, even after encrypting the index entries, i.e., document identities, keywords, and their corresponding keywords' relevance scores are encrypted, the cloud server could try to infer the keywords of the index based on the leakages such as distribution information, association information, and size pattern through various information disclosure attacks. Hence, protecting the privacy of index content is significant since any leakage from index compromises the complete privacy of the stored documents. Privacy of index keywords can be guaranteed if a SE scheme prevents these leakages with the help of suitable methodologies. A SE scheme is said to satisfy index privacy if the cloud server does not distinguish the index of one document from the index of another document in a polynomial time (Feng and He 2018; Li and Liu 2017). Therefore, the index privacy can also be referred to as Index Indistinguishability.

Trapdoor privacy: A trapdoor is an encrypted query sent by the data user to the cloud server to get the relevant documents. The objective of the trapdoor privacy is to prevent the cloud server from knowing which keywords are encoded in a trapdoor. If the cloud server knows the keywords, it tries to generate another trapdoor for the subset of the known keywords. Hence, the cloud server should not be able to determine any information from the given trapdoor and the indexes other than the ability to say that the trapdoor matches with encrypted indexes of some documents. A SE scheme is said to satisfy trapdoor privacy if the cloud server does not distinguish the trapdoor of one query from the trapdoor of another query in a polynomial time. Trapdoor privacy can also be referred to as Trapdoor Indistinguishability (Tahir et al. 2017). If trapdoor privacy is satisfied, the cloud server cannot infer query keywords directly from the given trapdoor other than through keyword guessing attacks and brute force attacks, which are computationally expensive to carry out (Arriaga et al. 2014). The privacy of the encoded keywords of both trapdoor and indexes can be guaranteed if trapdoor privacy is satisfied.

Rank privacy: The cloud server must be provisioned with ranking functionality to send the top-k relevant documents for the users' trapdoors by using various similarity measures such as Coordniate matching, Cosine similarity, and Jaccard Coefficient. For the given trapdoor, the rank information of each document is determined using various similarity measures with the help of encrypted TF, or TF-IDF values of trapdoor's keywords. This rank information conveys how significant a document is to the cloud servers, and the rank-order information of the retrieved documents is more likely to be exploited if two or more related trapdoors are sent to the cloud server over a given time period. Therefore, it is required to preserve rank privacy, and at the same time, the users should get the top-k relevant documents for their trapdoors. Hence, SE schemes are required to return the top-k relevant documents without affecting ranking privacy (Zhang et al. 2016b), (Chen et al. 2016). The privacy of both the keywords of trapdoors and indexes can be guaranteed if rank privacy is satisfied.

Search pattern privacy: Search pattern conveys what information the user is frequently looking for. Whenever the user wants to issue the same query again and again,

the same trapdoor may have to be generated by the data user each time for the same query. Hence, it leaks some pattern (search pattern) that conveys to the cloud server that two or more trapdoors are generated from the same keyword set. Search pattern privacy can also be referred to as Trapdoor unlinkability (Cao et al. 2014), i.e., prevents the cloud server from deducing the relationship between two or more trapdoors if all of them belong to the same query keyword set. If search pattern is leaked, keywords of the trapdoor can be known through scale analysis attacks (Cao et al. 2014). Search pattern leakage is caused due to the lack of randomness in issued trapdoors. Hence, sufficient randomness needs to be ensured while generating the trapdoors so that trapdoors cannot be linked even when the user issues the same query. The privacy of trapdoor keywords can be guaranteed if search pattern privacy is satisfied.

Access pattern privacy: Access pattern refers to information about whether a sequence of search results for the given two or more trapdoors is the same or not. It conveys the cloud server about what documents the user is frequently accessing. If access pattern is leaked, keywords of the trapdoors can be inferred through access pattern exploitation attack (Cash et al. 2015). Hence, secure SE schemes are required that cannot leak access pattern even if the users access the same documents again and again. There are various approaches like PIR (Chor et al. 1998), Oblivious RAMs (Goldreich and Ostrovsky 1996b), and Blind Storage technique, for preventing access pattern leakage. However, these techniques are computationally expensive. Hence, these protocols are practically infeasible for implementation in the real world environment. Therefore, efficient, feasible and secure searchable schemes are required for preventing access pattern leakage. The proposed research leaks access pattern, but it will avoid the exploitation of access pattern information. Keywords of trapdoors can be guaranteed if the access pattern privacy is satisfied.

Search accuracy: It is required for SE schemes to provide relevant documents to the users for their trapdoors to satisfy their information requirements (Liu et al. 2019; Xu et al. 2012). Besides guaranteeing the privacy, it is also required to ensure higher search accuracy to fulfill the users' needs and it will lead to the enhanced utilization of cloud storage services. However, search accuracy and security contradict with each

other, i.e., search accuracy is affected if security is met and vice versa. For example, assume that search pattern privacy is preserved by adding random keywords to a list of actual query keywords in trapdoor generation approach to ensure that different trapdoor is generated each time for the same query. But search accuracy may be affected as the cloud server might return some non-relevant documents, which are in response to the random keywords of the trapdoor. Therefore, it is equally important for the SE schemes to meet search accuracy besides meeting the above privacy requirements.

1.5 APPLICATIONS

There are some of the real-world application domains where the searchable encryption has scope for adoption. They are provided below.

1. Health care systems: The patients are concerned about their health records stored at the hospitals. The patients may lose their jobs if their sensitive health information is leaked to the owners of their organizations. Sometimes the patients may also visit another hospital on recommendations of regular doctor for better medication, where the patients may need, to share health records with the new doctor. In these scenarios, searchable encryption can guarantee the privacy of health records by storing them in encrypted form at cloud servers (Fabian et al. 2015), (Gardiyawasam Pussewalage and Oleshchuk 2016). Thus it benefits the hospitals in terms of saving money by uploading their data onto the cloud servers and also benefits the patients in terms of privacy as their data is accessible only by the recommended doctors but not by the administrative staff of the cloud servers and hospitals.

2. Computer forensics: The digital evidence such as cell phone records, log files, and email messages related to civil and felony crimes are stored in systems at forensic labs. The evidence could be targeted for destruction or alteration by the guilty involved in those crimes through administrative staff of these labs. Therefore, this evidence is significant, and is required to prevent the disclosure of this information to the unauthorized personnel to convict the guilty. Searchable encryption could be a viable solution in this scenario, since entire information would

be in encrypted form so that the administrative staff cannot come to which document belongs to which person and at the same time, it also allows the investigating agencies like FBI, Interpol, and CBI to retrieve the required information from forensic labs through trapdoors (Armknecht and Dewald 2015).

3. Email routing systems: The email service providers normally have access to all the emails that is a concern for many users since the mails may contain sensitive information. The email users want their mails to be secured against adversaries and even email service providers from knowing anything about their mails (Tang 2012). Searchable encryption can be used to store only encrypted mails at the email service providers while enabling retrieval of the mails through trapdoors.

4. Secure audit logging: Audit logs of any system are an important as they represent current and past activities of the system users. The audit logs can be used to detect unauthorized past activities carried out by different people in an organization. The contents of the audit logs contain sensitive information. Therefore, it needs to be protected from the adversaries who try to tamper those audit logs. Hence, searchable encryption can be used to preserve the privacy of audit logs contents (Waters et al. 2004).

5. Encrypted Search Engines: Search engines like Google, Yahoo, and Bing. allow the users to search for plaintext information retrieval. The cloud servers act as the search engines, which are aimed to retrieve the relevant information from the plaintext data. In SE context, retrieving relevant information from encrypted data without leaking any information is not yet fully realized. Even though CSPs like Amazon, Microsoft Azure offer support for SE, but they store some auxiliary information about the secret key, which is enough to infer all plaintext information from the encrypted data. SE has a scope in fulfilling the privacy and security requirements of the medium and large-scale organizations and also the universities. These institutions may have some sensitive information, i.e., regarding the current employees who are working on the ongoing projects, documents about the past and current projects, and future plans. This information is generally

stored in plaintext form on the servers of these institutions. As this information is highly sensitive, the organizations wish that their information is not be known even to the administrators of those servers. SE can fulfill their needs by developing encrypted search engines, wherein any data owner within the organization can encrypt their own data and store it at cloud servers while ensuring it is searchable by the intended recipients.

1.6 RESEARCH GAPS AND MOTIVATION

Data owners are interested in utilizing cloud servers to store and manage their data for achieving the benefits of lower cost, higher reliability, ease of access, and better performance. However, confidentiality and privacy (Kamara and Lauter 2010) prevent the data owners from excessive usage of cloud servers for storing their documents that contain sensitive information. Although searchable encryption guarantees privacy and confidentiality through encryption, privacy issues are not yet resolved completely because of the vulnerabilities in the adopted schemes that are used for encrypting indexes and queries. These vulnerabilities cause information leakages that could be exploited by the cloud servers to infer plaintext information through various informatino disclosure attacks, which are described in Section 1.3. The information leakages include frequency of ciphertext values, rank-order information, and search pattern. This dissertation focuses on addressing the research gaps involved in preventing these leakages besides supporting dynamic updates efficiently. This dissertation focuses on addressing the research gaps involved in preventing these leakages besides supporting dynamic updates efficiently. Each of them is briefly explained here in the context of encrypted search engines, while the full details are provided in Section 2.4.

- The frequency leakage of ciphertext values leads to the disclosure of index keywords. This leakage occurs due to the insufficient randomness in the order preserving encryption (OPE) schemes (Boldyreva et al. 2009; Wang et al. 2012). It is shown in (Pan et al. 2020); that this leaked frequency information even enables the attackers to infer plaintext information of not just a single keyword, but multiple keywords' values from the OPE encrypted values. Due to the frequency leak-

age, the existing OPE schemes are not applicable for usage in encrypted search engines, in which these schemes are used for encrypting index content of the documents, i.e., relevance scores of keywords. Probabilistic encryption schemes like Fully Homomorphic Encryption (Wang et al. 2015; Wu 2015) prevent frequency leakage, but the ciphertext values do not preserve the plaintext order due to which the relevant documents cannot be identified by the cloud server from the encrypted values. Hence, encryption schemes that preserve the plaintext order without leaking frequency information should be used. To fill this research gap, it is highly essential to improve the randomness of the OPE schemes; thereby, frequency leakage can be minimized or prevented. The first objective of our thesis focuses on mitigating the frequency leakage.

- The exposure of rank-order information to the cloud server leads to the disclosure of frequently issuing query keywords of trapdoors or frequently occurring keywords of the dataset. The search pattern also would be leaked to the cloud server when the user issues the same trapdoor again and again to retrieve the same documents. The search pattern leakage leads to the disclosure of plaintext keywords of the trapdoor. In encrypted search engines, the data users do not prefer to retrieve the documents in relevance order of his/her trapdoors due to the leakage of rank-order information, and they also cannot issue the same trapdoor to retrieve the same documents due to the leakage of the search pattern. The existing SE approaches proposed in (Cao et al. 2014) prevent the search pattern leakage by adding some random keywords to a list of actual keywords in queries. However, precision is affected in these approaches as the cloud server would return some non-relevant documents because of the random keywords. They also cannot prevent the rank-order information leakage completely as the cloud server could still determine the ranks of the documents because the random values of random keywords in indexes follow the standard deviation (σ) of actual keywords' values. To fill this research gap, it is highly essential to preserve the privacy of both rank-order information and search pattern without affecting precision. The second objective of our thesis focuses on meeting this requirement.

- After the documents are uploaded on to the cloud server in the context of encrypted search engines, the data owners would carry out their day-to-day business activities due to which the stored documents undergo some changes. It is, therefore, essential to reflect the updates in the already existing encrypted indexes efficiently. Incorporation of the updates facilitates the data users to get the latest top-k documents, which are very important in today's digital era, wherein, having access to the latest information makes the data users choose timely decisions. The existing SE approaches incorporate the updates using tree-based indexing schemes (Kamara and Papamanthou 2013; Wu and Li 2019; Xia et al. 2016). However, as the size of the tree in terms of height and width is higher, the existing approaches do not support all the dynamic operations (Insert, Delete, and Modify) efficiently. To fill this research gap, an efficient index structure is developed in this work to support dynamic updates efficiently.

1.7 THESIS CONTRIBUTIONS

The proposed research work aims at guaranteeing the privacy of documents stored at the cloud servers and also enables the data users to retrieve the latest top-k relevant documents more efficiently than the existing SE schemes. The major contributions of this work are provided below.

- A comprehensive literature of a single and multi-keyword search over encrypted data using Boolean and Ranked Search approaches (Kumar and Thilagam 2019a). Each of the existing SE approaches is analyzed with respect to the privacy of indexes, trapdoors, ranking information, search pattern, and access pattern along with search accuracy.

- An Enhanced One-to-Many Order Preserving Mapping technique to map the same plaintext keywords' relevance score to different ciphertext values. This technique mitigates the frequency leakage of same ciphertext values than the frequency leakage (Kumar and Thilagam 2019b) caused by the existing order preserving mapping techniques.

- A Pseudo-Ranking approach to retrieve top-k relevant documents securely without leaking rank-order information and search pattern to the cloud server. This approach also guarantees higher precision while assuring the privacy of both rank-order information and search pattern.

- A Max-heap tree based index structure for supporting dynamic updates efficiently and also for retrieving top-k documents efficiently. Also, a secure keyword dictionary expansion approach is proposed for adding new keywords to the existing dictionary without allowing the cloud server to know the location of the added keyword.

- A secure keyword dictionary expansion approach has been proposed to add a new keyword to the index without allowing the cloud server to know the location of the added keyword (Rashmi et al. 2018).

1.8 THESIS ORGANIZATION

The rest of the thesis is organized as follows: Chapter 2 presents the literature review on searchable encryption with respect to Boolean search, and Ranked search. Additionally, this chapter discusses dynamic updates supported by the existing searchable encryption schemes. Chapter 3 presents the problem description and objectives. Chapter 4 illustrates the proposed approach for mitigating frequency analysis attack and Chapter 5 for preventing rank-order exploitation attack and scale analysis attack. Chapter 6 presents the proposed tree based index structure for supporting dynamic updates. Chapter 7 summarizes the contributions of the research work presented in this thesis and discusses future research directions.

CHAPTER 2

LITERATURE REVIEW

Searchable Encryption (SE) approaches are classified as SSE (Searchable Symmetric Encryption) (Curtmola et al. 2006) and PEKS (Public Key Encryption with Keyword Search) based on the type of the key used in it. The above two approaches are used to retrieve either all relevant documents or top-k relevant documents based on the user issued trapdoors. The trapdoors may contain single or multiple query keywords, there exists various privacy issues involved in processing these trapdoors. This chapter reviews existing literature on SE approaches based on i) the type of search (Boolean or Ranked search) performed on the encrypted documents and ii) the number of keywords present in the trapdoors, which are given below:

1. Boolean Search
    - Single Keyword Boolean Search
    - Multi-Keyword Boolean Search
2. Ranked search
    - Single Keyword Ranked search
    - Multi-Keyword Ranked search

In this chapter, a comprehensive review of searching over encrypted documents using Boolean and Ranked search approaches are presented in Sections 2.1 and 2.2

respectively with respect to the precision and privacy requirements. In addition, Section 2.3 presents a review of dynamic updates supported by various searchable encryption approaches.

2.1 BOOLEAN SEARCH

Boolean search approaches return the documents that contain the user issued query keywords. The Boolean search systems are used in social networking sites, e.g., LinkedIn, Monster, Facebook, Mail Servers and many online shopping cart sites like Amazon, eBay, Flipkart. The Boolean searchable encryption approaches are simple, efficient, and easy to implement. The relevance in Boolean search can be interpreted using a matching between query keywords and document keywords. Two types of Boolean searches are possible based on the number of keywords present in Boolean search requests: i) Single Keyword Boolean Search, and ii) Multi-Keyword Boolean Search.

2.1.1 Single Keyword Boolean Search (SKBS)

SKBS approaches return the documents only if they contain the given keyword of the query. The SKBS approaches can be categorized further based on the indexing technique they have used: i) Forward-index based SKBS approaches, and ii) Inverted-index based SKBS approaches. In forward index, there exists an index for each document (Jin et al. 2016). It is suitable for updating documents but searching time is proportional to the number of documents in the dataset. Whereas, in Inverted index, for each keyword, it contains a pointer to document Ids that they contain (Curtmola et al. 2006). In this approach, searching time is constant but performing update operation is costly as it may require adjustment of n positions for each update. Hence, SE approaches of SKBS are grouped into Forward-index based SKBS approaches and Inverted-index based SKBS approaches. The following sections present the SE approaches under these two categories with respect to the privacy requirements.

2.1.1.1 Forward-index based SKBS approaches

A deterministic and sub-linear efficient public key searchable encryption scheme is proposed in (Bellare et al. 2007) with a hash-based indexing method. It achieves the

sub-linear search time by tagging the hash of plaintext keyword to encrypted index keyword in order to give the exact location for the server to determine the corresponding encrypted documents for the given queries. The proposed approach guarantees search accuracy due to hashing but it is prone to false positives as many ciphertext messages are associated with a single hashtag. It leaks the search pattern because the trapdoor is generated using hash functions. Hence, the same hash value is generated if the same query is issued again. The privacy of indexes is not preserved as they are vulnerable to dictionary attacks because of the low min-entropy in encryption and restriction in length of the encrypted keywords in the indexes.

The public key based SE scheme (Boneh et al. 2004b) was first proposed using bilinear map. Index privacy is preserved due to the usage of the bilinear map with prime order groups for encrypting index keywords. Trapdoor privacy is preserved due to the usage of hash functions and random number in its trapdoor generation approach. However, due to the usage of same hash functions and same random number, search pattern privacy cannot be preserved.

The first storage efficient fuzzy keyword search is proposed in (Li et al. 2010) to improve the user searching experience. This approach returns the closest possible matching documents for the given query using wildcard '*' based edit distance approach. The privacy of both index and trapdoor is preserved due to the usage of one-way function for encryption. Search pattern privacy is not preserved because of the usage of the same one-way function and the same fuzzy keyword set for the same query in generating trapdoors. For example, for the given query keyword, and distance 'd', which specifies the number of operations required to transform from one keyword into another, the wildcard '*' can be used to generate fuzzy keyword set by placing it at any position within the keyword. '*' indicates any alphabet character can be substituted in place of '*'. The number of substitutions allowed is limited by a distance 'd'. The resulting keywords set after substitution is called fuzzy keywords set. This fuzzy keyword is generated again when the user reissues the same query. Hence, search pattern privacy is not preserved. The search accuracy is not high as the adopted wildcard approach misses some of the other possible fuzzy keywords that are close to the actual keywords of the dataset.

A two-round communication protocol is proposed in (Chang and Mitzenmacher 2005) to preserve index and trapdoor privacies by masking the content of the indexes and queries with pseudo-random functions. The index of each file is represented using a ' n' bit string, where n indicates the number of keywords in the dataset. If a keyword w_i belongs to file F_i, the corresponding bit in index string is set to 1 using pseudo-random functions. Similarly, the trapdoor is also an n-bit string, and the corresponding bit of query keyword is set to 1. This approach also allows the data owners to update their documents securely and retrieves the latest documents for their queries. It meets the forward privacy, i.e., earlier issued trapdoors do not match the updated documents since the indexes of newly added documents are encrypted with different keys. Even though the precision is high in this approach, it is more prone to search pattern leakage since the same bit of the keyword has to be set if the same query is issued.

PEKS with different variations is proposed for achieving time constraint based privacy (Abdalla et al. 2005). The authors have come up with such a searchable encryption method by converting hierarchical identity based encryption into public key encryption with temporary keyword search. Index privacy is preserved due to the usage of the bilinear pairing operation with prime order group elements for encrypting keywords and time period. The trapdoor privacy is preserved due to the inclusion of a time period and usage of the hash function and random number in generating trapdoor. After the validity period expires, it cannot be used for searching. Search accuracy is preserved due to the usage of the same random number in encrypting document keywords and the queries. But, this approach leaks the search pattern since it generates the same trapdoor for the same query because of the usage of the same hash function and the same random number in its trapdoor generation function.

The secure Z-IDX index technique has been proposed in (Goh et al. 2003) for faster retrieval of the documents. The proposed Z-IDX index is built using Bloom filters and pseudo-random functions. Index privacy is preserved due to the usage of pseudo-random functions twice for encrypting index keywords of the documents and also due to the incorporation of random keywords in indexes such that the adversaries cannot distinguish the document indexes by their sizes. Trapdoor privacy is preserved in

Table 2.1: Security analysis of Forward-index based Single Keyword Boolean Search approaches.

		Privacy Requirements						
Ref.	**Approach**	**Data Confidentiality**	**Index Privacy**	**Trapdoor Privacy**	**Search Accuracy**	**Search Pattern**	**Access Pattern**	**HBC**
Bellare et al. (2007)	PEKS	✓		✓	✓			
Boneh et al. (2004b)	PEKS	✓	✓	✓	✓			
Li et al. (2010)	SSE	✓	✓	✓				
(Chang and Mitzenmacher 2005)	SSE or PEKS	✓	✓	✓	✓			
(Abdalla et al. 2005)	PEKS	✓	✓	✓	✓			
Goh et al. (2003)	SSE	✓	✓	✓		✓		
(Boneh et al. 2007)	PEKS	✓	✓	✓		✓	✓	✓

this approach as it uses pseudo-random functions for encrypting keywords of the query. This approach also supports occurrence search in which the data users can specify the occurrence of particular query keyword in documents. Due to the presence of occurrence number in trapdoors that may change each time as per the users' requirements, it will generate different trapdoor for the same query. Hence, search pattern privacy is preserved in this context. However, search accuracy cannot be achieved due to the usage of bloom filters in indexes that cause false positives.

A Private Information Protocol (PIR) protocol based keyword search is aimed to preserve the access pattern privacy (Boneh et al. 2007) using an oblivious storage approach. The privacy of access pattern is safeguarded because of the expulsion of retrieved documents of the earlier trapdoors and also due to some irrelevant documents returned to each trapdoor because of the presence of random keywords in trapdoors. The privacy of index and trapdoor is preserved as pseudo-random functions and probabilistic approaches are used for encrypting each keyword of the documents and queries. Besides this, each document index and trapdoor includes some random keywords along with actual keywords. Search pattern privacy is preserved due to the presence of random keywords in each trapdoor. Search accuracy is not preserved as it returns false positives because of the usage of bloom filters for storing each document index. This approach meets all the privacy requirements, but it involves two rounds of interaction for each trapdoor between the cloud server and users due to oblivious storage. Moreover, it is a cumbersome process due to the large post-computational cost on data users in filtering out the random documents to get the actual documents, and it also requires enormous storage for storing multiple copies of the actual documents' indexes along with random ones. It is found that PIR based approach (Boneh et al. 2007) preserves all privacy requirements specified in Section 1.4, but search accuracy is not satisfied due to the involvement of random keywords in trapdoor. The analysis of forward-index based SKBS approaches with respect to the privacy requirements is presented in Table 2.1.

2.1.1.2 Inverted-index based SKBS approaches

Block and stream cipher approaches for encrypting index keywords is proposed in (Song et al. 2000). It achieves higher search accuracy as the location of the corresponding documents for the given query keyword can be directly determined by the trapdoor itself. The privacy of index and trapdoor is not preserved since they are prone to brute force attack with the less computational cost since the intruder has to apply brute force technique to only half of the encrypted word to derive the keyword of the query. The other half of the encrypted word is meant for searching and decryption purposes. Search pattern privacy is not preserved as it generates the same trapdoor for the same query due to the usage of a deterministic encryption scheme for encrypting query keyword.

Another sub-linear searching method presented in (Curtmola et al. 2006) uses an inverted indexing technique. It introduces a new security notion called security against the adaptive adversarial model, who tries to infer information from the outcome of previous search requests. Index privacy is preserved as pseudo-random permutations are used to encrypt the index keywords and the keywords of the documents in indexes are scrambled if they are present. If keywords are not present in any document, some random values are added in order to ensure that the number of documents that match each keyword is the same. The authors extended their approach for supporting search by multiple users through a Broadcast encryption scheme. However, search pattern privacy is not preserved in this approach as the trapdoor generation function uses the same pseudo-random permutation and the same seed for the same query. The analysis of inverted-index based SKBS approaches with respect to the privacy requirements is presented in Table 2.2.

2.1.2 Multi-Keyword Boolean Search (MKBS)

MKBS approaches support multi-keyword queries by using Boolean operators, i.e., AND, OR, and NOT. The precision of MKBS approaches is higher than SKBS since each returned document contains either all the keywords or more than one keyword of the query. However, due to the involvement of multiple keywords in a query, the MKBS

Table 2.2: Security analysis of Inverted-index based Single Keyword Boolean Search approaches.

		Privacy Requirements						
Ref.	**Approach**	**Data Confidentiality**	**Index Privacy**	**Trapdoor Privacy**	**Search Accuracy**	**Search Pattern**	**Access Pattern**	**HBC**
Song et al. (2000)	Block and Stream ciphers	✓			✓			
Curtmola et al. (2006)	SSE	✓	✓	✓	✓			

approaches give more scope for the cloud server to infer meaningful information, i.e., the documents containing the common keywords and keywords of the indexes through statistical analysis from the previous trapdoors and their corresponding search results. The MKBS approaches are further classified based on the indexing technique used: they are Forward-index based MKBS approaches, and Inverted-index based MKBS approaches. The following sections present the SE approaches under these two categories with respect to the privacy requirements.

2.1.2.1 Forward-index based MKBS approaches

The first MKBS approach over encrypted data uses a forward index (Golle et al. 2004). They have proposed two approaches under two security constructions for MKBS. One is by the Decisional Diffie-Hellman (Boneh 1998) approach, wherein the communication cost of trapdoor is proportional to the number of documents, and another one is by using Bilinear Decisional Diffie-Hellman (BDDH) (Joux 2002) that achieves constant cost. This approach achieves search accuracy since the trapdoor includes the keywords that will directly match with the keyword fields of each document index. As the index of each document includes keyword fields for enabling search, the server can try to infer some specific keyword fields of indexes and trapdoors through keyword guessing attacks. Hence, the index and trapdoor privacies are not preserved in this approach.

Hidden Vector Encryption (HVE) scheme based multi-keyword search approach is proposed to support a rich set of query predicates, i.e., subset, range, and comparison queries over encrypted data (Boneh and Waters 2007). The privacy of both indexes and tokens i.e., the trapdoors, which consists of predicates of the query, is preserved due to the usage of the bilinear map operation for encrypting both keywords of indexes and predicates of queries. However, the search pattern privacy is not achieved due to the reasons that include; usage of the same secret key with the same exponent in its token generation approach to meet the search accuracy and also due to the threshold of the predicates in trapdoors that the cloud server comes to know with a simple binary search technique.

Table 2.3: Security analysis of Forward-index based Multi-Keyword Boolean Search approaches.

Ref.	Approach	Privacy Requirements						
		Data Confidentiality	**Index Privacy**	**Trapdoor Privacy**	**Search Accuracy**	**Search Pattern**	**Access Pattern**	**HBC**
Golle et al. (2004)	PEKS	✓			✓			
Boneh and Waters (2007)	Hidden Vector Encryption	✓	✓	✓	✓			
Hwang and Lee (2007)	PEKS	✓	✓	✓	✓			
Katz et al. (2008)	Predicate Encryption	✓	✓	✓	✓			
Shen et al. (2009)	Predicate Encryption	✓	✓	✓		✓		
Zhang and Zhang (2011)	PEKS	✓	✓	✓	✓			
Ryu and Takagi (2007)	SSE	✓	✓	✓	✓			
Park (2011)	Hidden Vector Encryption	✓	✓	✓		✓		
Yang et al. (2017)	Paillier Encryption	✓	✓	✓		✓		

A storage efficient conjunctive keyword search mechanism is proposed in (Hwang and Lee 2007). Compared to other public-key based conjunctive keyword search approaches, it reduces the size of ciphertexts of documents due to the usage of bilinear map operation and a single random value as a private key, this approach is highly useful when data owners upload encrypted information onto the cloud servers for multiple users. It conserves index privacy as it uses prime-order group based bilinear map operation, which uses two collision-resistant hash functions for encrypting each index keyword of the document. Hence, the server cannot infer any information from that index. The trapdoor privacy is also achieved in the same way as in index privacy. This approach also enables multiple users to search without creating separate ciphertext for each individual user by using their mPECK scheme. But, search pattern privacy is not preserved in this approach as it uses the same secret key with the same hash functions in generating trapdoors, due to which, the same trapdoor is generated for the same query.

Another predicate-only encryption scheme for multi-keyword search proposed in (Katz et al. 2008) uses forward indexing method and PEKS. In this approach, the secret keys correspond to predicates, and the ciphertext is related to attributes, i.e., attributes of the index. The predicates in this approach support a wide variety of queries such as equality, conjunctive, disjunctive, and also allow to evaluate polynomial equations. This approach also allows other intended users to search by converting it to an anonymous identity-based encryption scheme. Search accuracy is achieved due to the inner product operation, which allows the predicates to be evaluated on corresponding attributes of the encrypted documents successfully if they match. Index privacy is preserved since the bilinear map operation with composite order groups' elements are used for masking attributes of documents by multiplying them with different groups' elements. Similarly, trapdoor privacy (predicate privacy) is preserved by multiplying predicates with random elements of groups' elements. But, search pattern privacy is not preserved due to the usage of the same random elements in generating trapdoors for the same queries. An improvised symmetric-key based predicate encryption scheme is proposed in (Shen et al. 2009) using bilinear maps with composite order group, which has the product of four primes as the order compared to (Katz et al. 2008), which has

the product of three primes. Due to this extra randomness, it preserves search pattern privacy. However, this randomness may impact the search accuracy as it may generate a predicate, i.e., a token, which may not correspond to the intended documents' index attributes.

A conjunctive multi-keyword subset search approach is proposed in (Zhang and Zhang 2011) using a public key encryption approach. It returns documents for the queries when they match subsets of indexed keywords of each document. In this approach, keywords of document indexes are transformed to l-degree polynomial equations, where 'l' indicates the number of keywords of each document. The length of each polynomial equation of each document would be the same in this approach. The roots of these equations would be used as the trapdoors to retrieve the corresponding documents. It preserves index privacy as the coefficients of these polynomial equations are encrypted using bilinear map operation with prime order groups. Trapdoor privacy is preserved as the roots are encrypted using a bilinear map with a secret key corresponding to the intended documents. It achieves search accuracy since the cloud server returns the documents whose polynomial equations are satisfied by the roots of the given trapdoor. Search pattern privacy is not preserved since the same roots may have to be issued to evaluate the corresponding polynomial equations in order to retrieve the same intended documents.

An efficient multi-keyword search mechanism is introduced in (Ryu and Takagi 2007) using SSE and forward indexing method. This approach is efficient as it uses bilinear maps, which are constructed from ordinary curves for encrypting indexes and trapdoors. The privacy of both index and trapdoor is preserved because of the usage of bilinear groups of prime order and hash functions for generating encrypted indexes and trapdoors. But, search pattern privacy is not preserved as it generates the same trapdoor for the same query because of the usage of the same random number of bilinear group in its trapdoor generation approach.

An efficient Hidden Vector Encryption (HVE) approach is proposed (Park 2011) for supporting a conjunctive keyword search on encrypted data using prime-order based bilinear map operation. The privacy of the index and trapdoor is preserved using two

random exponents in generating encrypted indexes and tokens. Search pattern privacy is also preserved as it generates different trapdoor for the same query due to the usage of two random exponents in its token generation approach. The authors also proposed anonymous identity-based encryption, which is more efficient than the approach proposed in (Boyen and Waters 2006) because of the usage of less number of random elements in public keys for encryption. Search accuracy is affected for certain tokens as it uses random exponents in the token generation that may not correspond to the attributes of intended documents.

A wildcard based multi-keyword search approach is proposed in (Yang et al. 2017) to enhance the system users' searching experience. It is useful for the users who usually issue queries containing spelling errors in each keyword of the query. The wildcard '*' can be present in any position, i.e., front, middle, or at the back of each keyword in the query and up to two wildcards are allowed in this approach for each keyword of the query. If the wildcard is present at the back of the keyword, any number of characters can be substituted in place of a wildcard. Otherwise, the user specifies a maximum number of characters to be allowed for substitution. No information is leaked during wildcard based search operation due to the usage of secure ciphertext partition approach, which is required to partition the encrypted keyword at the wildcard position and also due to secure multi-bit extraction protocol, which is necessary to extract the specified number of bits up to the partitioned position. It starts substituting the characters from the wildcard position for a possible match with the index keyword using secure greater than or equal protocol. This approach preserves the privacy of indexes and trapdoors since both the index and query keywords are encrypted by using a paillier encryption scheme. Search pattern privacy is also met due to the non-deterministic nature of paillier encryption, which generates a different trapdoor for the same query. However, search accuracy would not be high since it might match other index keywords especially when wildcard appears at the end of query keyword, e.g., wildcard query keyword priv* might match with the keyword "privacy" but the user intends to give keyword "private". The users in this approach can also search over encrypted data of multiple owners by obtaining an individual authorization certificate from each data

owner. With a flexible time-period based authorization methodology, where the cloud server has information about authorized and revocated users, only the authorized data user is allowed to perform search operation during that time period and cannot search once the period expires. The analysis of forward-index based SKBS approaches with respect to the privacy requirements is presented in Table 2.3. In summary, it is found that there exists no approach that guarantees all the specified security requirements in Section 1.4.

2.1.2.2 Inverted-index based MKBS approaches

A public key based two-round communication protocol for multi-keyword search is proposed (Wang et al. 2015) to preserve access pattern privacy. It prevents the access pattern leakage due to the adoption of blind storage protocol (Naveed et al. 2014), in which, the documents are stored in the form of blocks in different locations because of which the cloud server does not come to know which documents are retrieved for the given trapdoor each time. The index is represented as a matrix of polynomial equations. Each equation represents the inverted list of each keyword. Each equation is padded with random elements and is encrypted using paillier encryption scheme. As it uses private set intersection protocol (Freedman et al. 2004), the cloud server cannot even infer the documents containing common keywords from the intersection results of index keywords correspond to the query keywords. Hence, the index privacy is preserved. Similar to the index privacy, the trapdoor privacy is preserved as the coefficients of query keywords are also encrypted using paillier encryption scheme. This approach generates different trapdoor for the same query due to the non-deterministic nature of paillier encryption scheme used for generating trapdoor and besides this, the trapdoor also contains encrypted coefficients of random numbers along with the coefficients of actual query keywords due to which search pattern privacy is also protected. However, it causes the difficulty for owners and the users to adapt it in the real-world environment since the proposed approach involves two round communications for retrieving documents for their trapdoors due to the adoption of blind storage protocol and paillier encryption scheme.

Table 2.4: Security analysis of Inverted-index based Multi-Keyword Boolean Search approaches.

		Privacy Requirements						
Ref.	**Approach**	**Data Confidentiality**	**Index Privacy**	**Trapdoor Privacy**	**Search Accuracy**	**Search Pattern**	**Access Pattern**	**HBC**
Wang et al. (2015)	Paillier Encryption	✓	✓	✓		✓	✓	✓
Drazen et al. (2015)	SSE and OXT Protocol	✓	✓	✓	✓			
Miao et al. (2017)	CP-ABE	✓	✓	✓	✓	✓		

A multi-keyword verifiable searchable encryption scheme proposed in (Drazen et al. 2015) uses the Oblivious Cross Tags (OXT) protocol (Cash et al. 2013). This approach is useful when cloud servers do not follow the protocol as specified by the data owner in returning the relevant documents for the given trapdoor. For each trapdoor, the cloud server returns the encrypted document identities and corresponding tags of trapdoor keywords. The data user verifies the returned document identities by validating the tag, which is a pseudo-random value that is generated using the corresponding pseudo-random function with the same secret key. Data user determines the actual document identities on successful verification of the results upon which the data user can request them from the cloud server. The index privacy is preserved as this approach uses the pseudo-random function and non-deterministic encryption scheme for encrypting index keywords, tags, and document identities, respectively. Trapdoor privacy is also preserved as it uses the pseudo-random function for encrypting keywords of the query. Search pattern privacy is not achieved as it uses the same function with the same key for generating trapdoor for the same query. This approach is not acceptable to the data owner and users as it involves two rounds of communication for each trapdoor between the user and the server.

Attribute-Based Keyword Search over Hierarchical Data (ABKSA-HD) for multiple users is proposed (Miao et al. 2017). Two approaches are introduced, namely, ABKS-HD-I for supporting search operation without user revocation and ABKS-HD-II with user revocation. This approach is highly suitable for searching over the data that have a hierarchical relationship with other data items in the dataset. The privacy of both Index and Trapdoor is met due to the adoption of a hash function applied to each keyword, whose result is used as an exponent of the bilinear group generator for encrypting index and query keywords. The search pattern privacy is preserved as the exponent in trapdoor is the multiplication of the hash value of each query keyword with two random numbers. Hence, the different trapdoor is generated for the same query. This approach uses the CP-ABE technique to support search operation by multiple users. It maintains a unique version number for each user at the cloud server that is generated based on the user's attributes. To retrieve data from the cloud server (CS), the user has to submit the

trapdoor along with his/her attribute set to the cloud server. The CS then first verifies the version number from those attributes and then uses the trapdoor to search the index upon successful verification. When a user is revoked, the data owner communicates the corresponding updated version number to the cloud server. Therefore, the revoked user cannot perform search operation with an old version number once he is revoked. Hence, this approach efficiently handles user revocation than the existing methods. However, this approach does not hide the privacy of the attributes of the users. Security analysis of inverted-index based MKBS approaches is presented in Table 2.4.

From the literature review, it is found that paillier encryption based approach (Wang et al. 2015) preserves all privacy requirements specified in Section 1.4, but search accuracy is not satisfied due to the presence of random coefficients in trapdoors.

2.2 RANKED SEARCH

In ranked search approaches, the cloud server returns relevant documents in decreasing relevance order of the user query. Ranked search approaches are currently being used in search engines such as Google, Yahoo, and Microsoft Bing on plaintext data. With regard to encrypted data, it can be used in various third-party cloud service providers for enabling ranked search over encrypted data, e.g., secure E-mail communication. Ranking of documents is done by including additional keywords' relevance scores into searchable indexes. The additional information, i.e., encrypted TF or TF-IDF weights of all searchable keywords of documents is stored in indexes. Due to this additional information, the precision of Ranked search approaches is higher than Boolean search approaches. This is because of the indexes in Boolean search approaches contain just 1 or 0 ,i.e., indicating presence of keyword in corresponding documents, whereas in indexes of ranked search approaches, different keyword scoring measures such as TF, and TF-IDF are stored that convey the relevance, i.e., importance of the keyword in documents and document collection respectively. However, this additional information may lead to a frequency analysis attack if the distribution information is leaked to the cloud server from the encrypted keywords' relevance scores. In addition, it may also lead to rank-order exploitation attack as the cloud server comes to which document is

Table 2.5: Security analysis of Forward-index based Single Keyword Ranked Search approach.

Ref.	Approach	Privacy Requirements							
		Data Confidentiality	**Index Privacy**	**Trapdoor Privacy**	**Search Accuracy**	**Rank Privacy**	**Search Pattern**	**Access Pattern**	**HBC**
(Kuzu et al. 2012)	Threshold-based Similarity using LSH and Paillier Encryption	✓	✓	✓	✓	✓		✓	

more relevant and which ones are not since the cloud server only returns the top-k relevant documents to the users' trapdoors. This rank order information can be exploited by the cloud server to infer the plaintext information of frequently issuing query keywords or frequently occurring keywords by observing the difference in scores (or ranks) of the returned documents for the given two or more trapdoors. To prevent this attack, SE approaches are required to preserve rank privacy. However, if rank privacy is preserved, precision would be affected. Hence, rank privacy is a very significant privacy requirement in ranked search approaches, and the SE approaches should assure rank privacy without affecting the search accuracy, i.e., precision. Two types of Ranked searches are possible based on a number of keywords present in ranked search requests: i) Single Keyword Ranked Search (SKRS), and ii) Multi-Keyword Ranked Search (MKRS).

2.2.1 Single Keyword Ranked Search (SKRS)

In SKRS approaches, documents are assigned ranks based on the single keyword of the given query. The SKRS approaches are further classified based on the indexing technique used: they are Forward-index based SKRS approaches, and Inverted-index based SKRS approaches. Each SE approach of these two categories are analyzed with respect to the privacy requirements in the following two subsections.

2.2.1.1 Forward-index based SKRS approaches

A fault tolerant and secure Ranked search approach (Kuzu et al. 2012) is proposed using Paillier Encryption (PE) and Locality Sensitive Hashing (LSH) methods. It allows queries that consist of spelling, representation errors but still provides relevant documents due to LSH indexing. In this approach, the cloud server returns the relevant documents by forwarding the given trapdoor to the corresponding buckets, where the stored document's features are much closer to the given query, hence, search accuracy is preserved. The index privacy is protected since the original bucket's content with a few added fake records is encrypted with a paillier encryption. The trapdoor privacy is preserved as the trapdoors are generated using pseudo-random functions and LSH functions. The access pattern privacy is also accomplished by using two-servers that are assumed to be not colluding, where one server stores the index information and

other server stores corresponding encrypted document collection. When a user issues trapdoor, first, it goes to the index server that finds the encrypted document identifiers with the top scores due to the homomorphic additive property of paillier encryption. Those encrypted identifiers will be sent to the documents server, which decrypts the scores of those identifiers and sends the corresponding top-k documents to the user. The ranking privacy is also satisfied as the second server has access to encrypted documents only and does not have any information about the trapdoors. Therefore, it cannot infer any information from the decrypted paillier scores. Search pattern privacy is not preserved due to the usage of pseudo-random functions, and LSH functions with the same key for generating the trapdoor.

There exists only one forward-index based SKRS approach in literature for searching over encrypted data. Security analysis of this approach is presented in Table 2.5.

2.2.1.2 Inverted-index based SKRS approaches

A ZERBER^{+R} model based approach (Zerr et al. 2009) is proposed to preserve the privacy of rank information by merging posting lists of different keywords in inverted indexing. Each merged list includes some encrypted keywords, the corresponding posting elements, i.e., document IDs and encrypted TF values. The data users in this approach retrieve the top-k documents by informing the cloud server about which posting lists he/she wants to retrieve for their query keywords. The cloud server then returns the posting lists, which are then decrypted by the user and determines the top-k document IDs based on the corresponding query keyword score present within the returned lists. The user then further requests the top-k identified documents from the cloud server. In this approach, the index privacy is preserved by merging posting lists of unrelated keywords to prevent the leakages of even document frequency (Büttcher and Clarke 2005). Due to the merging of lists, the cloud server cannot come to know how many documents contain given the query keyword. Search accuracy is achieved as ranking documents is done at the user side. Ranking privacy is also achieved because of usage of relevance score transformation function for encrypting relevance scores. This function maps each plaintext TF value of each keyword to a unique value within the given range so as to

Table 2.6: Security analysis of Inverted-index based Single Keyword Ranked Search approaches.

		Privacy Requirements							
Ref.	**Approach**	**Data Confidentiality**	**Index Privacy**	**Trapdoor Privacy**	**Search Accuracy**	**Rank Privacy**	**Search Pattern**	**Access Pattern**	**HBC**
Zerr et al. (2009)	TF relevance score with Relevance score transformation function	✓	✓		✓	✓			
Wang et al. (2012)	TF relevance score with One-to-Many OPE	✓	✓	✓	✓				
Tahir et al. (2017)	TF-IDF relevance score with Masking approach	✓	✓	✓		✓	✓		

prevent the distribution information leakage of keywords to the cloud server. Search pattern privacy could not be achieved as the users need to specify the same posting lists if the users issue the same query. This approach also does not preserve access pattern privacy, and it is not adoptable in the real world since it requires at least two rounds of interactions with the cloud server for processing each search request.

Another single keyword Ranked search approach is devised (Wang et al. 2012) to enable rank-ordering the documents for the given trapdoor using One-to-many Order preserving encryption (OPE) scheme. This scheme ensures that the same relevance scores in the index of documents are mapped to different values. Trapdoor privacy is preserved due to the usage of pseudo-random functions and hash functions for generating trapdoors. As TF values of index keywords are encrypted using One-to-Many OPE, index privacy is preserved as long as the output range of One-to-Many OPE scheme is comparatively very large than the input TF values. If not, it compromises the privacy of index keywords from encrypted TF values through the leakage of distribution information of specific index keywords. Due to the usage of One-to-many OPE for encrypting TF values, search accuracy is achieved as it maintains the order information after encryption. This order information suffices for the cloud server to determine the order of retrieved documents. Hence, ranking privacy is not preserved as the cloud server comes to know the top-k relevant documents for the given trapdoor. The privacy of the search pattern is also not preserved due to the usage of hash functions for generating the trapdoors.

Lightweight single keyword ranked search approach (Tahir et al. 2017) is proposed to preserve search pattern privacy by using one-way hash functions and a probabilistic symmetric key encryption scheme for generating trapdoor. Index privacy is preserved due to the usage of inverse one-way hash functions for encrypting each plaintext keyword. Ranking privacy is preserved by multiplying each TF-IDF value of each keyword with a pseudo-random number so that extracting the plaintext TF-IDF value from that multiplication result is computationally infeasible. Due to this random number, the cloud server does not come to the actual ranks of the documents. But, search accuracy is affected because of masking, i.e., multiplying actual TF-IDF values with the

pseudo-random numbers, which cannot preserve the order of plaintext TF-IDF values after masking. Security analysis of inverted-index based SKRS approaches is presented in Table 2.6.

In this subsection, various inverted-index based SKRS approaches are explored for searching over encrypted data. The literature review reveals that there exists no approach that guarantees all the privacy requirements as specified in Section 1.4.

2.2.2 Multi-Keyword Ranked Search (MKRS)

MKRS approaches return documents in decreasing relevance order of queries. The precision of MKRS is higher than SKBS, MKBS, and SKRS because of the presence of multiple keywords in a query, i.e., MKRS allows the user to specify all his/her search constraints in a query and also due to the availability of keywords' relevance scores. Though the indexes in SKRS contain TF or TF-IDF values, the precision of SKRS cannot be higher than MKRS since all the user search constraints cannot be specified with the single keyword of the query. The MKRS approaches are further classified based on the indexing technique used. They are Forward-index based MKRS approaches, and Inverted-index based MKRS approaches. Each SE approach of these categories are analyzed with respect to the privacy requirements such as Data, Index, Trapdoor, Ranking, Search and Access pattern privacy along with search accuracy. Review of these sections is presented in the following two subsections.

2.2.2.1 Forward-index based MKRS Approaches

The access pattern privacy preserving MKRS approach proposed in (Örencik and Savaş 2012) uses PIR protocol by which the cloud server does not come to know which documents are relevant ones for users among the returned documents for their trapdoors. This approach also preserves search pattern privacy due to PIR in which a randomized trapdoor is sent to the cloud servers that includes a set of random keywords and actual query keywords. Trapdoor privacy is preserved as the trapdoor, which is an r-bit string generated as a result of applying hash functions on randomized queries. Search accuracy is good due to the usage of hash functions for generating both trapdoors and encrypted indexes. The keywords' relevance scores in indexes are stored in n-different

levels for each document, and a query matching in higher level indicates it is the more relevant document. This n-level information is directly exposed to the cloud server, which exploits it to infer specific keywords of some indices through a frequency analysis attack.

A full-text retrieval approach (Song et al. 2017) is proposed for enabling efficient search over a massive number of encrypted documents by storing their documents in hierarchical tree form based on the similarity of the documents.. Each document index consists of sets of compound keywords that are grouped based on their similarity. For each compound keyword, membership entropy score is determined using term frequency and document length. These scores are helpful in assigning ranks to the documents for the given trapdoor. The data owners then generate a set of index bloom filters using hash functions for each group of compound keywords. These bloom filter based indexes and corresponding encrypted documents are dispatched to the cloud server, which organizes bloom filters in a hierarchical index tree form at the cloud server. Similar to the index bloom filters, the data users also generate query bloom filters for his/her actual query keywords and sends their entropy scores to the cloud server along with the query bloom filters. Search accuracy is achieved due to the usage of larger width size of bloom filters in order to avoid false positives. The privacy of index and trapdoor is preserved due to the usage of one-way hash functions applied to both index and query compound words. Rank privacy is not preserved as the entropy scores are not encrypted. Search pattern privacy is also not preserved due to the hash functions applied to queries that generate the same bloom filters if the same query is issued. Though this approach is efficient as far as searching functionality is concerned due to the Bloom filters, it incurs extra computational burden on data users and owners due to a lot of pre-processing work involved in generating compound words for each document and each query. An effective MKRS scheme proposed in (Cao et al. 2014) uses a secure inner product similarity measure and SSE. The indexes and trapdoors are vectors, which represent TF values of documents' keywords and IDF values of queries' keywords respectively. Each index and query vectors are extended from D_i, Q_i to D_i+U+1, Q_i+U+1 respectively, where U indicates number of dummy keywords added to both actual documents and

Table 2.7: Security analysis of Forward-index based Multi-Keyword Ranked Search approaches.

Ref.	Approach	Privacy Requirements							
		Data Confidentiality	**Index Privacy**	**Trapdoor Privacy**	**Search Accuracy**	**Rank Privacy**	**Search Pattern**	**Access Pattern**	**HBC**
Örencik and Savaş (2012)	TF-IDF relevance score with SSE	✓		✓	✓		✓	✓	
Song et al. (2017)	Membership entropy based relevance score with SSE	✓	✓	✓	✓				
Cao et al. (2014)	TF-IDF relevance score with Masking approach	✓	✓	✓		✓	✓		
Xia et al. (2016)	TF-IDF relevance score with Masking approach	✓	✓	✓		✓	✓		
Li et al. (2014b)	TF relevance score with Inner product operation and ABE	✓	✓	✓	✓	✓	✓		
Sun et al. (2013)	TF relevance score with Masking approach	✓	✓	✓		✓	✓		
Shen et al. (2018)	TF-IDF relevance score with Masking approach	✓	✓	✓		✓	✓		
Fu et al. (2014)	Enhanced TF-IDF relevance score with Masking approach	✓	✓	✓		✓	✓		
Orencik et al. (2013)	TF-IDF relevance score with SSE and Paillier Encryption	✓	✓	✓	✓	✓		✓	
Yu et al. (2013)	TF-IDF relevance score with HME	✓	✓	✓	✓	✓	✓		

query vectors. Random values are assigned to the dummy keywords in index vectors and value 1 is assigned to some of the dummy keywords in query vector and for others, value 0 is assigned. The privacy of index and trapdoor is preserved as both index and query vectors are masked (multiplied) with two invertible matrices and their inverses respectively based on a random vector. The ranking privacy is preserved as the ranking information is masked and randomized with random values of dummy keywords due to which some non-relevant documents will be sent to the users. Hence, the cloud server cannot determine which documents among the returned documents are more important for the user's trapdoors. The privacy of the search pattern is also safeguarded due to the random values of the dummy keywords in queries which makes the generated trapdoors indistinguishable. However, search accuracy cannot be higher due to the presence of random values of dummy keywords in trapdoors. An efficient and effective MKRS scheme is proposed in (Sun et al. 2013), which achieve the same privacy requirements in the same way as achieved in (Cao et al. 2014). It uses Multidimensional B-Tree (MDB) (Bentley 1975) based indexing method to retrieve the top-k related documents more efficiently with the help of MD search algorithm. The MD search algorithm helps in retrieving documents quickly by skipping some nodes of the tree that are not required. However, search accuracy could be affected due to the random values of phantom keywords in both trapdoors and indexes.

A novel approach for performing dynamic Multi-Keyword Ranked search method proposed in (Li et al. 2014b) handles the update operations (adding new keywords) efficiently. The search accuracy is achieved without affecting ranking privacy because of the two reasons: i) It ensures that the weight of dummy keywords in index and query vectors are restricted to be within a sufficiently large range. ii)The index also includes the popularity of queried keywords and keywords' accessed frequency information. This approach meets the privacy of indexes, trapdoors, ranking privacy, and search pattern in the same way as achieved in (Cao et al. 2014). It handles the data updates efficiently without re-encrypting the whole index with the partition matrix based approach, wherein, for the newly updated information, a new square matrix is added to the already existing index matrix at the cloud server. It also authorizes other intended

users to perform search operation on stored data using ABE. However, the only problem with this approach is burdensome that the data owner has to keep track of secret keys whenever an update operation occurs since the newly added matrix will be encrypted with a new key instead of the old key. This approach cannot handle deletion operations efficiently.

Preference-based MKRS approach proposed in (Shen et al. 2018) returns the relevant documents as per the preferences of user' s query keywords using lagrange coefficients and secure inner product similarity measure. The preferences are represented by a priority, which is a numerical value, which conveys the significance of query keyword, i.e., the higher the priority number, the more priority the user gives for these keywords. Index vector of each document in this approach represents TF-IDF values of document's keywords that are encrypted using two invertible matrices based on a random vector, due to which index privacy is preserved. The trapdoor generated for each query includes two vectors, i.e., encrypted query vector and encrypted preference vector. These two vectors are generated by first converting query keywords and their priorities into legrange polynomial equations and then the coefficients of these equations are represented as vectors. These two vectors are then encrypted using two inverses of invertible matrices based on a random vector due to which trapdoor privacy is preserved. The privacy of ranking and search pattern is preserved in the same way as it is preserved in (Cao et al. 2014). Search accuracy is also not achieved in this approach due to the presence of random values of some random keywords in trapdoors and indexes.

A semantic-based MKRS approach is proposed in (Fu et al. 2014) to support not only the given query keywords but also the synonyms of query keywords by constructing a synonym keyword set for each keyword of the document collection. It uses a balanced binary tree indexing scheme to retrieve the documents efficiently. The documents are ranked by computing cosine similarity measure between documents and query vectors. This approach uses a new keyword weighting measure called Enhanced Term Frequency-Inverse Document Frequency (ETF-IDF) to reflect keyword's significance across all categories within the document collection. The proposed approach

satisfies ranking, trapdoor, index, data, and other privacies in the same way as achieved in (Sun et al. 2013),(Cao et al. 2014) using the same methodology by extending the vectors of documents and queries with random values and their encryption using invertible matrices based on a random vector. However, search accuracy is affected because the random values of dummy keywords in index and query vectors affect the actual relevance scores of documents.

Paillier cryptosystem based MKRS approach devised in (Orencik et al. 2013) preserves the ranking privacy without affecting search accuracy using two servers. In this approach, a signature, which is generated using minhash functions, is used to represent each document. These functions map each keyword of the document into one of the λ buckets. Each bucket stores the keywords of the documents that have similar features. The search accuracy is achieved due to minhash functions, which map the signature of the query to corresponding buckets, which contain the relevant documents. The privacy of both ranking and access pattern is preserved due to paillier encryption scheme and the usage of two non-conspired servers, i.e., Search Server (SS) and File Server (FS), respectively. As TF-IDF values are encrypted using paillier encryption scheme, which maps each TF-IDF value to different ciphertext values, the frequency information will not be leaked. Also, the cloud server cannot come to know which document has a higher rank and which documents not due to the paillier encryption. Thus, ranking privacy is preserved. But, due to the homomorphic additive property of paillier encryption scheme, SS ascertains the scores of the documents straightforwardly from the encoded TF-IDF values of keywords. Then, paillier scores, their document identities, and value 'k' are sent to the File server, which decrypts the scores and sorts them in descending order; then returns the top-k encrypted documents. FS holds only document identities, their plaintext scores, and encrypted documents and no access to trapdoors. Hence, no other information can be extracted from them. SS does not know what documents were returned for earlier trapdoors of the users. Thus, ranking privacy, access pattern privacy are preserved without affecting search accuracy. However, the problem with this approach is that FS should be supplied with a secret key for decrypting paillier scores. Search pattern privacy is also not preserved due to the usage of minhash functions.

Two Round Searchable Encryption (TRSE) approach is designed (Yu et al. 2013) to prevent the statistical information leakages from encrypted TF-IDF values, i.e., inter-distribution of files and term distribution. It uses modified Fully Homomorphic Encryption over Integers(FHEI) for encrypting TF-IDF values of document keywords due to which there exists no frequency leakage, and the cloud server can assign scores to the documents directly from the encrypted TF-IDF values because of the additive property of homomorphic encryption. For any given trapdoor, the cloud server sends document Ids and their scores to the users, who decrypts the encrypted scores and then requests the cloud server actual top-k document Ids in the second round. Thus, search accuracy is achieved. Rank privacy is also preserved as rank-ordering of documents is done at the user's side and the cloud server cannot determine which document is more relevant than others. Search pattern privacy is preserved as it generates different trapdoor for the same query as it uses a different random number for the same query keywords in generating trapdoor. However, the proposed TRSE scheme is computationally expensive as it generates large ciphertext because of the usage of large prime numbers and random numbers in FHEI for encrypting each TF-IDF value. Performing any operation on such large ciphertext is expensive. This computational burden, because of FHEI incurs more cost on data owners for consuming more computational power from the resources of the cloud server. The privacy of the access pattern is not achieved since this cloud server knows the returned documents. Hence, this approach cannot hide the association between trapdoors and their search results. Security analysis of forward-index based MKRS approaches with respect to the privacy requirements is presented in Table 2.7.

Rank privacy in approaches (Cao et al. 2014), (Xia et al. 2016) is preserved by randomization of ranking information with invertible matrices in indexes, i.e., some dummy values are added to phantom keywords in each document index. Due to these dummy values, search accuracy is compromised as the cloud server may return some non-relevant documents to the users for their trapdoors. It is also found that there exists no approach that guarantees all the specified privacy requirements as given in Section 1.4 along with search accuracy.

Table 2.8: Security analysis of Inverted-index based Multi-Keyword Ranked Search approaches.

		Privacy Requirements							
Ref.	**Approach**	**Data Confidentiality**	**Index Privacy**	**Trapdoor Privacy**	**Search Accuracy**	**Rank Privacy**	**Search Pattern**	**Access Pattern**	**HBC**
Swaminathan et al. (2007)	Okapi relevance score with One-to-Many OPE	✓	✓	✓	✓				
Ibrahim et al. (2012)	TF-IDF relevance score with Paillier encryption	✓	✓	✓	✓	✓		✓	
Li et al. (2015)	TF-IDF relevance score with Masking approach and CP-ABE	✓	✓	✓	✓	✓	✓	✓	✓

2.2.2.2 Inverted-index based MKRS Approaches

The first Ranked search approach for multi-keyword query (Swaminathan et al. 2007) is proposed using okapi relevance score and One-to-Many Order Preserving encryption (OPE) scheme. Privacy of both index and trapdoor keywords are preserved as they are first encrypted using symmetric encryption scheme and they are further mapped to hash values using hash functions. It achieves search accuracy as the relevance scores of each keyword of each document are encrypted using one-to-many OPE, which maintains the order information after encryption. However, the rank privacy is dependent on an intermediate unit of cloud server called secure computing unit (SCU), which rank orders the documents based on encrypted TF values of query keywords. This SCU is a trusted entity of data owners and assumed that it does not leaks ranking information to the data server. Hence, this approach may not be adoptable by the people who wish the server to perform ranking without relying on the trust of computing units. The search pattern is also leaked as it generates the same trapdoor for the same query due to the usage of the hash functions with the same key in generating trapdoors.

A MKRS approach (Ibrahim et al. 2012) is aimed at preventing the access pattern leakage by separating the association between trapdoors and corresponding returned documents by using two servers; Search server (SS) and Document server (DS). The SS stores encrypted indexes and DS stores encrypted documents. Search accuracy without affecting rank privacy is attained due to two reasons:i) Paillier encryption scheme, which ensures all encrypted TF-IDF values will follow the uniform distribution, which prevents leakage of distribution information of relevance scores. ii) SS does not know the actual scores of document IDs because of the paillier encryption which does not maintain plaintext order information after encryption. For any given user's trapdoor, the SS calculates each document's score by adding the paillier scores of keywords of the trapdoor due to its additive homomorphic property. The SS then sends the document id's with their paillier scores to the DS, which determines the actual scores after decrypting the scores with the corresponding secret key. The SS does not know which documents are returned and the DS does not know anything about trapdoor, but the user gets the relevant documents with the help of DS. Thus, access pattern privacy and rank-

ing privacy is preserved along with search accuracy. Index privacy is also preserved since the index information contains few fake keywords with random values, and the keywords are encrypted using a one-way hash function, document id's, and keywords' relevances by paillier encryption. Hence, the SS cannot obtain any statistical information from the indexes. Trapdoor privacy is also preserved since each query keyword is encrypted with one way hash function. However, this approach does not preserve search pattern privacy because the same trapdoor is generated for the same query because of usage of hash functions in generating trapdoors.

Blind storage (Naveed et al. 2014) based MKRS approach is devised in (Li et al. 2015) to preserve access pattern privacy. It also enables multiple users to perform search operation using CP-ABE. Each trapdoor includes encrypted query, stag, which contains random numbers that give the location of the corresponding index entries for computing relevance score. Index and Trapdoor privacy are preserved as they both are encrypted using two invertible matrices and their inverses respectively based on a random vector. Search accuracy is achieved due to the usage of inner product similarity operation for computing similarity score of documents between query vector and document vector. The search pattern privacy is preserved as each stag of trapdoor contains some random numbers, and the encrypted query of trapdoor also includes two random values due to which different trapdoor is generated for the same query. After computing the relevance scores, the cloud server sends the encrypted document identifies called descriptors to the users, who decrypt them using corresponding secret key. The users then determine block identities based on the decrypted documents identities and then further requests the cloud server those block identities along with some random Ids in order to preserve the access pattern privacy. This approach preserves all the security requirements along with precision. However, the proposed approach requires two rounds of communication for processing each trapdoor. Security analysis of inverted-index based MKRS approaches is presented in Table 2.8.

Blind storage based approach (Li et al. 2015) preserves all the security requirements as specified in Section 1.4 along with search accuracy, but it requires two rounds of interaction between the cloud server and the user for each trapdoor.

In summary, as far as the privacy requirements of all the Boolean and Ranked search approaches are concerned, there exists only one approach (Li et al. 2015) that preserves all the security requirements specified in section 3 along with search accuracy. However, it requires two rounds of interaction for processing each trapdoor. Access and Search Pattern leakages in most of the approaches are prevented by approaches that are practically infeasible solutions such as private information retrieval, blind storage protocol or by storing index information and associated documents in two different servers. Hence, secure and efficient multi-keyword search approaches are needed to retrieve the top-k relevant documents securely in a single round of communication.

2.3 DYNAMIC UPDATES

In this section, a literature review of searchable encryption schemes supporting dynamic updates on documents is presented. The data users expect to get the latest top-k relevant documents for their trapdoors. In today's digital era, wherein, having access to the latest information makes the data users choose timely decisions. To provide the latest top-k relevant documents, it is required to incorporate the updates in the already existing encrypted indexes efficiently. Therefore, the SE schemes are required to incorporate the updates in the already existing indexes efficiently. The forward-index based SE schemes proposed in (Bellare et al. 2007; Boneh et al. 2004b; Goh et al. 2003; Kuzu et al. 2012) support dynamic updates straightforwardly, but they are not efficient as the update operations and searching time is proportional to the number of documents of the dataset. Hence, the tree-based indexing schemes are used in SE to address this issue. Tree-based indexing schemes are helpful in improving the time complexity of performing dynamic updates and search operation. There exists various tree-based indexing structures that are aimed to facilitate data owners to perform dynamic updates, and they enable the cloud servers to retrieve the top-k relevant documents for the users' trapdoors in sub-linear time.

A boolean search approach is proposed to improve the efficiency of search time and dynamic updates using a keyword red-black (KRB) tree-based indexing structure (Kamara and Papamanthou 2013). In this indexing structure, each node consists of

two vectors, each with m-bits (number of unique keywords in the dataset), are used to represent the index information of the document. This indexing structure is developed in a bottom-up fashion, where first leaf nodes are created from the given documents, and then the internal nodes are generated based on the content of leaf nodes. Each leaf node consists of two m-bit vectors, where a bit i in either of the vectors indicates the presence of the corresponding keyword k_i in that document. The internal nodes also consist of two vectors, where each bit in either of the vectors is the result of OR operation on the respective bits of the left and right child nodes. For a given single keyword of the trapdoor, the search process starts from the root to the leaf nodes that contain the given keyword. It involves recursively searching the corresponding bit in the vectors of the nodes starting from the root node to the leaf nodes. It stops searching at an internal node if the corresponding bit in vectors is set to 0, which indicates the keyword does not present in the below sub-tree nodes. As the height of the tree is O (log n), the search time complexity of determining a document that contains the given keyword is O(log n). Therefore, the search time complexity of determining r documents that contain the given keyword is $O(rlogn)$. Similarly, the time complexity of performing dynamic updates is O(q log n), where q the number of unique keywords of the added document. The problem with this indexing technique is that while searching and updating, it involves many internal nodes which do not represent the documents' content directly. Due to this, the height and breadth of the tree is very high , which will directly impact the efficiency of retrieving documents and performing dynamic updates.

Another boolean search approach using a Privacy Bloom Filter (PB) based tree indexing technique (Li et al. 2014a) is proposed for retrieving documents and performing updates efficiently. This indexing structure is developed in a top-down fashion, where the root node is constructed first that contains all the possible keywords of the dataset, and then the content of the root node is divided into left and right child nodes. Each of the child nodes contains half of the elements of the root node. Then, each left and right child nodes are further partitioned recursively until the leaf nodes are formed where each leaf node stores the index content of a single document. In this indexing structure, bloom filters are used to store the content of the document's index. Each index

represents the actual keywords of the document along with some random keywords, which prevent the cloud server from distinguishing one document's index from another document's index. Similar to (Kamara and Papamanthou 2013), the height of this tree-based indexing structure is also O(logn). Therefore, the search time complexity of determining $'R'$ documents for a given trapdoor is O(R log n). This indexing technique also contains internal nodes that do not represent documents directly, but they are required to be traversed while searching and performing updates. Hence, it incurs extra cost in index construction, searching, and performing updates. Another multi-keyword boolean search with an Indistinguishable binary tree structure (Li and Liu 2017) is proposed. This indexing structure is developed in bottom-up fashion (Wu and Li 2019). Each node of this tree consists of two m-bits vectors where a bit i in either of the vectors indicates the presence of the corresponding keyword k_i in that document. Bloom filters are used to store the indexed keywords of the document. In this approach, a pseudo-random function is used to determine which of the two vectors to be used to store the keyword of the document, and the corresponding keyword's position in that vector would be set to 1. This tree-based indexing is called an indistinguishable bloom filter tree index because the random numbers are used to decide which of the two vectors to be chosen in each node for storing keywords of the document in order to de-correlate the content of other nodes. The search complexity of this approach is also O(R log n) similar to (Li et al. 2014a). Both of the bloom filter based indexing structures (Li and Liu 2017; Li et al. 2014a) are optimized through their width optimization and depth optimization techniques. These optimization techniques avoid visiting some internal nodes while processing trapdoors and performing updates, but not all the intermediate nodes would be avoided. Hence, the worst-case time complexity of searching and performing updates cannot be improved unless all internal nodes are avoided. Search accuracy is also affected by these indexing techniques since they have used bloom filters to store the content of the indexes that cause false positives.

A multi-keyword boolean search approach using Virtual Binary (VB) tree-based indexing structure is proposed (Wu and Li 2019). VB tree is a complete binary tree where each leaf node represents a document, and the keywords of the corresponding

document are stored in it. The internal nodes store the union of the keywords of the left and right child, and the root node of the tree contains all the unique keywords of the dataset. In this indexing structure, the tree does not exist in real, but the each node of the tree is stored in a hash table in order to prevent the exposure of branches and nodes' information to the cloud server. Each element in this hash table consists of an encrypted keyword and the path information. The path information is stored in binary form, i.e., 0 for left child and 1 for a right child that gives the document identity $id(doc)$ after the conversion of binary information into a decimal value. For any given trapdoor, the search process involves recursively checking all the paths wherever a matching encrypted keyword of the trapdoor exists. A relevant document is found only if the length of the path is equal to the height of the tree, and then it determines the document identity from that path information. Since only the leaf nodes represent the actual documents, searching time and performing updates in this indexing structure is also not optimal. Because of the existence of many internal nodes that do not represent the documents directly, the height and breadth of VB tree are higher, which have a higher impact on searching time and performing dynamic updates.

There exists only one multi-keyword ranked search work that retrieves the top-k relevant documents and supports dynamic updates efficiently with the help of the Keyword Balanced Binary (KBB) tree indexing structure (Xia et al. 2016). This indexing structure is developed in a bottom-up fashion, where first leaf nodes are formed based on the keywords' relevance scores of documents, i.e., TF values. Then, the internal nodes are generated using the content of the leaf nodes. Each element in internal node represents the maximum TF value of either the left or right child nodes. Both the internal and leaf nodes in this tree consist of $m + m'$ values, where there are 'm' TF values of actual keywords and m' random values. The trapdoor also consists of $m + m'$ values, where each of the value in m is generated based on the IDF value of query keywords, and Other m' values are random keywords, whose values are set to either 0 or 1 randomly. For any given trapdoor, the search procedure starts from the root node to the leaf nodes using Greedy Depth-First Search (GDFS). In this technqiue, the cloud server determines top-k relevant documents by calculating the similarity score at each node,

starting from the root node. The cloud server also determines scores at the left and right child nodes and searches based on the maximum score of the child and continues the same procedure until the leaf node with the help of GDFS. At each leaf node, the document will be added to a list only if its score is greater than the scores of other existing nodes in a list, and it stops searching when the top-k relevant documents are found. As this approach uses the GDFS approach, it determines top-k relevant documents without visiting all the internal nodes. Thus, the time complexity of determining top-k relevant documents using this indexing structure is $O(m + m')^2 + xmlogn)$. However, this approach involves visiting some internal nodes that do not represent the documents directly. Hence, the search time complexity is not optimal. Due to the presence of internal nodes, it increases the height and breadth of the trees that will affect the efficiency of Insert, Delete, and Modify operations besides searching time.

All the above tree-based indexing structures such as KRB tree, PB tree, and VB tree-based approaches support only boolean search. These approaches guarantee the privacy of indexes and trapdoors. But these approaches leak search patterns since they have used either pseudo-random functions or one-way hash functions for generating trapdoors that are deterministic. Hence, they generate the same trapdoor for the same query. IBF tree based boolean search approach prevents search pattern leakage by inserting some random location hash pairs along with the actual location hash pairs of query keywords in their trapdoors. However, these random location hash pairs cause some false positives, thus affects the precision. The KBB tree-based SE approach supports ranked search and also prevents the leakages of both rank-order information and search pattern due to the presence of random keywords in trapdoors and the assignment of values to the random keywords in indexes. However, search accuracy is affected in this approach due to the random keywords. Another problem with KBB tree-based scheme is that it is required to visit all internal nodes while processing trapdoors and performing dynamic updates. Therefore, this causes an increase in tree height and width, which affects the efficiency of dynamic updates and search time. The time complexity analysis of all the tree-based indexing SE approaches is presented in Table 2.9.

Table 2.9: Comparison of SE schemes supporting dynamic updates

Ref.	**Approach**	Search Methodology	**Search time**	**Updates time**
Kamara and Papamanthou (2013)	Keyword red-black tree	Boolean Search	$O(xlogn)$	$O(mlogn)$
Li et al. (2014a)	PBtree	Range and Boolean search	$O(xlogn)$	$O(logn)$
Xia et al. (2016)	Keyword balanced binary Tree	Ranked Search	$O(mlogn+m)$	$O(m^2logn+m)$
Li and Liu (2017)	Indistinguishable binary tree	Boolean Search	$O(xlogn)$	-
(Wu and Li 2019)	Virtual binary tree	Boolean Search	$O(xlogn)$	$O(logn)$

Note: n indicates of number of documents, m indicates the number of keywords, x refers to the number of documents that are relevant to the given trapdoor's keywords.

2.4 RESEARCH DIRECTIONS AND CHALLENGES

The privacy issues of existing searchable encryption approaches are mainly involved in encrypting keywords' relevance scores, generating trapdoors, retrieval of top-k relevant documents, search pattern, and access pattern. The keyword relevance scores, i.e., TF or TF-IDF values in indexes helpful in determining ranks of the documents. The keywords' relevance scores need to be encrypted in such a way that the cloud server can perform rank-ordering the documents from the encrypted scores. Encryption schemes like Order preserving encryption, One-to-Many OPE meet this requirement. These encryption schemes should not leak any information other than order information. However, these schemes leak both frequency and order information. Order information helps in performing rank order information, but the frequency information should either be prevented or mitigated. The cloud server mounts a frequency analysis attack to exploit the frequency leakage information to infer plaintext information from the encrypted keywords' relevance scores. The reason for frequency leakage is that the lack of randomness in mapping the given plaintext values to ciphertext values. A pseudo-random number cannot be used directly for the purpose of preventing frequency, if it is so, the ciphertext values will not preserve the order of plaintext values after mapping. Therefore, it is highly challenging to balance the randomness while minimizing the frequency leakage without affecting the order of plaintext values in ciphertext values.

Another privacy issue arises when a user wants to retrieve the same documents repeatedly. The user sends the same trapdoor again and again for this purpose. Search pattern leakage can be determined from the issued trapdoors in this context that leads to the scale analysis attack. The existing multi-keyword search approaches prevent search pattern leakage by adding random keywords along with actual query keywords (Cao et al. 2014; Xiangyu et al. 2019). As these random keywords change each time, the different trapdoor would be generated for the same query. Thus, it prevents the search pattern leakage. However, random keywords of trapdoors affect precision because some non-relevant documents would be returned to the user in response to the added random keywords of the trapdoor. A privacy issue also arises when cloud servers send the top-k relevant documents in descending order based on their ranks

to the users' trapdoors. The rank information leakage leads to rank-order exploitation attack. This attack allows the cloud server to infer plaintext information from the rank-order information. To prevent this attack, rank information should not be leaked to the cloud server. Without allowing cloud server to know rank-order information, it cannot decide which documents are relevant documents and which ones are not relevant documents. The existing approaches cannot prevent rank-order information completely. It is, therefore, highly difficult to satisfy users' information requirements by sending top-k relevant documents while preserving the privacy of both rank-order information and search pattern without affecting precision.

In addition to addressing the privacy issues, it is also highly essential to fulfill the user's information needs quickly by sending the latest relevant documents efficiently for the given users' trapdoors. To meet this requirement, it is required to support dynamic updates such as Insert, Delete and Modify operations efficiently over encrypted data. Tree-based indexing techniques certainly helpful in retrieving documents and performing updates, but they are not optimal in searchable encryption. Since the tree-based index structuretures are constructed either in a top-down or bottom-up fashion, an update operation on any single node may lead to updates on many other nodes of the tree. Therefore, it is required to minimize the impact of one nodes' update operation on other nodes of the tree. To satisfy this requirement, the height and breadth of the trees must be kept as minimum as possible so that the impact of update operation on other nodes would be minimum. Thereby it helps to perform dynamic updates in near-optimal sub-linear time and also the retrieval of the documents. The existing tree-based indexing techniques do not achieve this as they involve many intermediate nodes to be visited while processing trapdoors and performing updates. It is highly challenging, and a complex task is involved in designing a tree-based indexing structure with minimum height and breadth while enabling it to be easily updated. At the same time, it is also required to make it searchable quickly for the identification of the top-k relevant documents without visiting all tree nodes.

The following major research gaps are identified from all the issues identified through our literature work:

- *Secure order preserving encryption scheme* : The existing schemes such as OPE (Boldyreva et al. 2009) and One-to-Many OPE (Wang et al. 2012) can be used to encrypt the relevance scores of keywords in indexes. These schemes enable the cloud server to assign ranks to the documents for the given trapdoors based on the encrypted scores directly. However, they both leak frequency information that leads to a frequency analysis attack. This frequency information leakage even enables the attackers to infer plaintext information of multiple keywords' values from the OPE encrypted values (Pan et al. 2020). This leakage is due to the lack of sufficient randomness in their encryption processes. Another variant of OPE (Kerschbaum 2015) also leaks the frequency information in the form of linear depth of the tree. This depth precisely corresponds to the frequency of plaintext scores. Several works proposed in (Grubbs et al. 2017; Maffei et al. 2017) demonstrated the inference of sensitive plaintext information from the encrypted values of this OPE scheme. A multi-source order-preserving encryption (MSOPE) scheme (Liang et al. 2020) is proposed to prevent the frequency leakage with the help of Paillier encryption. However, identifying relevant patients' encrypted health records require an interactive session between a doctor and a patient for each query, or it requires a secret key to be shared with a doctor to decrypt and identify the actual patient's health records. The data owners and users do not prefer to adopt the approaches that require them to share the decryption key and force them to involve in interaction with the server for each retrieval. Therefore, there is a need for an order preserving encryption scheme for encrypting keywords' relevance scores without leaking frequency information to the cloud server. This leakage can be mitigated or prevented completely by increasing the randomness of the existing schemes appropriately.

- *Precision and privacy preserving multi-keyword search approach:* The precision and privacy are two contradicting requirements of SE schemes. Precision might be compromised to some extent if privacy is more preferred, and privacy would

be compromised if precision is more preferred (Cao et al. 2014; Xiangyu et al. 2019). For example, to achieve higher precision, it is required to send top-k relevant documents to the users' for their trapdoors. But, the rank-order information might be leaked while sending the documents. Thus, the privacy of rank information would be compromised while guaranteeing higher precision. This rank-order information leakage leads to the disclosure of entire plaintext information of documents through a full reconstruction attack (Lacharité et al. 2018). Likewise, consider another contradicting example where it is required to preserve the search pattern privacy. The existing approaches prevent search pattern leakage by adding random keywords in trapdoors besides query keywords or by adding confusing keywords (Fu et al. 2020), but these random or confusing keywords could cause false positives. Thus, precision is compromised while guaranteeing search pattern privacy. However, the data user has to process the retrieved documents (that includes some non-relevant documents) again to find out the actual relevant documents. This post processing after retrieval needs to be avoided at data user side. Therefore, there is a need for a searchable encryption approach that guarantees both precision and privacy without the involvement of user in post processing of the retrieved documents.

- *Efficient indexing structure for supporting dynamic updates*: The focus of the existing tree-based SE schemes is to retrieve the relevant documents efficiently while supporting dynamic updates (Dai et al. 2020; Kamara and Papamanthou 2013; Wu and Li 2019; Xia et al. 2016). However, since the height and breadth of the existing tree-based indexing structures are higher, they cause a delay and not optimal in performing dynamic updates and retrieval of the top-k relevant documents. Therefore, there is a need for an efficient indexing structure to address this issue.

- *Feasible approach for preserving access pattern privacy:* The access pattern privacy in existing approaches is preserved through practically infeasible solutions such as Private information retrieval (Chor et al. 1998), blind storage methods Naveed et al. 2014 and Oblivious RAM Goldreich and Ostrovsky 1996b.

There exists no feasible approach that guarantees access pattern privacy (Cui et al. 2018; John et al. 2020). Some approaches preserve access pattern privacy by using two servers (search server and file server), of which one holds the secret key for decrypting the search server assigned ranks (Orencik et al. 2013) and the file server for sending the relevant documents to the users. The file server comes to know the access pattern, but cannot infer any information from it as it has no access to the trapdoors and the indexes. But, disclosing a secret key to file server causes a serious threat to the stored data. Therefore, it is required to preserve access pattern privacy without disclosing a secret key to any server in the two-server approach.

2.5 SUMMARY

In this chapter, a comprehensive review of searching over encrypted data using Boolean and Ranked search approaches is presented with respect to the precision and privacy requirements specified in Section 1.4. This chapter also provides a review of dynamic updates supported by various searchable encryption schemes. The existing SE approaches have been presented categorically based on the features that include; the number of keywords present in the trapdoor, how the search results are returned to the users' trapdoors, and their support for dynamic updates. Finally, a discussion on all research issues and challenges associated with privacy, precision, and dynamic updates is provided.

From the literature review, it is identified that the information disclosure attacks in SE exploit the leakages to infer plaintext information from the encrypted data. Some of the primary reasons for these leakages include the lack of sufficient randomness in the encryption schemes adopted in SE, functionality support for enabling the cloud server to perform some operation on encrypted data. The encryption schemes are different from each other in terms of security, efficiency and precision. Hence, they should be selectively chosen while encrypting data as per the precision and security requirements of data owners. If the concern of data owners is more about precision, then they can use deterministic encryption schemes for encrypting data. If their concern is more about the privacy, then they can use the non-deterministic encryption schemes for encrypting

data. Therefore, the adopted SE approaches can be classified based on their ability to guarantee precision. They are Deterministic encryption (DE) and Non-deterministic encryption (NDE), of which NDE is also referred to as Probabilistic encryption Goldwasser and Micali (1984). In DE schemes, the same plaintext is always mapped to the same ciphertext. Hence, they guarantee high precision, and they are also efficient, but there exists a lot of scope for the possibility of security attacks discussed in Section 1.3. Unpadded RSA and AES with ECB mode are a couple of examples of DE schemes. In NDE schemes, the same plaintext will be encrypted to different ciphertext value each time under the same key. Hence, NDE schemes are secured, but they are not as efficient as DE schemes, and the precision also won't be high. For example, Paillier encryption is NDE scheme. This scheme require a secret key to decrypt the paillier scores of the documents in order to determine actual ranks of the documents.

The existing SE approaches can also be classified on various features to allow data owners to specify multiple users to read and modify their data, functionality to verify the returned results, and the type of index that is supported, i.e., forward- index-based and inverted-index based. Based on the number of writers and readers, SE approaches can be classified as Single Writer Single Reader (SWSR), Single Writer Multiple Reader (SWMR), Multiple Writer Single Reader (MWSR) and Multiple Writer Multiple Reader (MWMR) (Bösch et al. 2014; Rao et al. 2018). Similarly, based on the ability to verify whether the returned results are correct or not, the SE approaches can be classified as Verifiable Searchable Encryption (VSE) (Drazen et al. 2015) and Non-Verifiable Searchable Encryption (NVSE). SE approaches can also be classified based on the type of index, i.e., Forward-index and Inverted-index. While the former one supports updates without much overhead but searching time is proportional to the number of documents, and the latter one is efficient at retrieving, but it has difficulty in updating the index due to the readjust of many index entries for every single update. The taxonomy of adopted SE approaches based on these features is shown in Figure 2.1.

There are other cryptographic approaches such as Private Information Retrieval (PIR), Blind Storage, and Oblivious RAM. These approaches are exclusively meant

for preventing both the search pattern and access pattern leakage information. In PIR protocol, for each query, a set of random queries are also sent to the cloud server, and similarly, a set of random documents are also stored at the cloud server along with the actual documents. Due to the random queries and random documents, the cloud server does not come to know which query is real and which original documents are returned to the users for their queries. Blind storage is another protocol, in which, the encrypted documents are stored in blocks of memory so that the cloud server only knows which blocks are uploaded and retrieved and it does not come to know which blocks constitute the same file. Similarly, Oblivious RAM is another protocol, in which, the documents would be kept re-encrypted and the locations of the stored documents in memory would be kept changing, due to which the cloud server does not come to know which documents the user is frequency accessing. These approaches have a higher scope to guarantee higher privacy, but they are not viable to adopt in the real world due to the reasons include; incurring the vast cost of post-processing search results, multiple rounds of communication between user and server, and also required to keep track of memory blocks in which the required files are stored respectively. Therefore, it is needed to come up with easily deployable SE schemes to address theses issues.

From the literature, it has been found that there exists no straight forward approach in SE that guarantees precision and all the privacy requirements in a single round of communication due to the contradictory nature of precision and privacy. Because of the lack of a single suitable encryption scheme that facilitates determining relevant documents without compromising privacy, developing a deployable multi-keyword search approach that ensures both higher precision and privacy is always highly challenging.

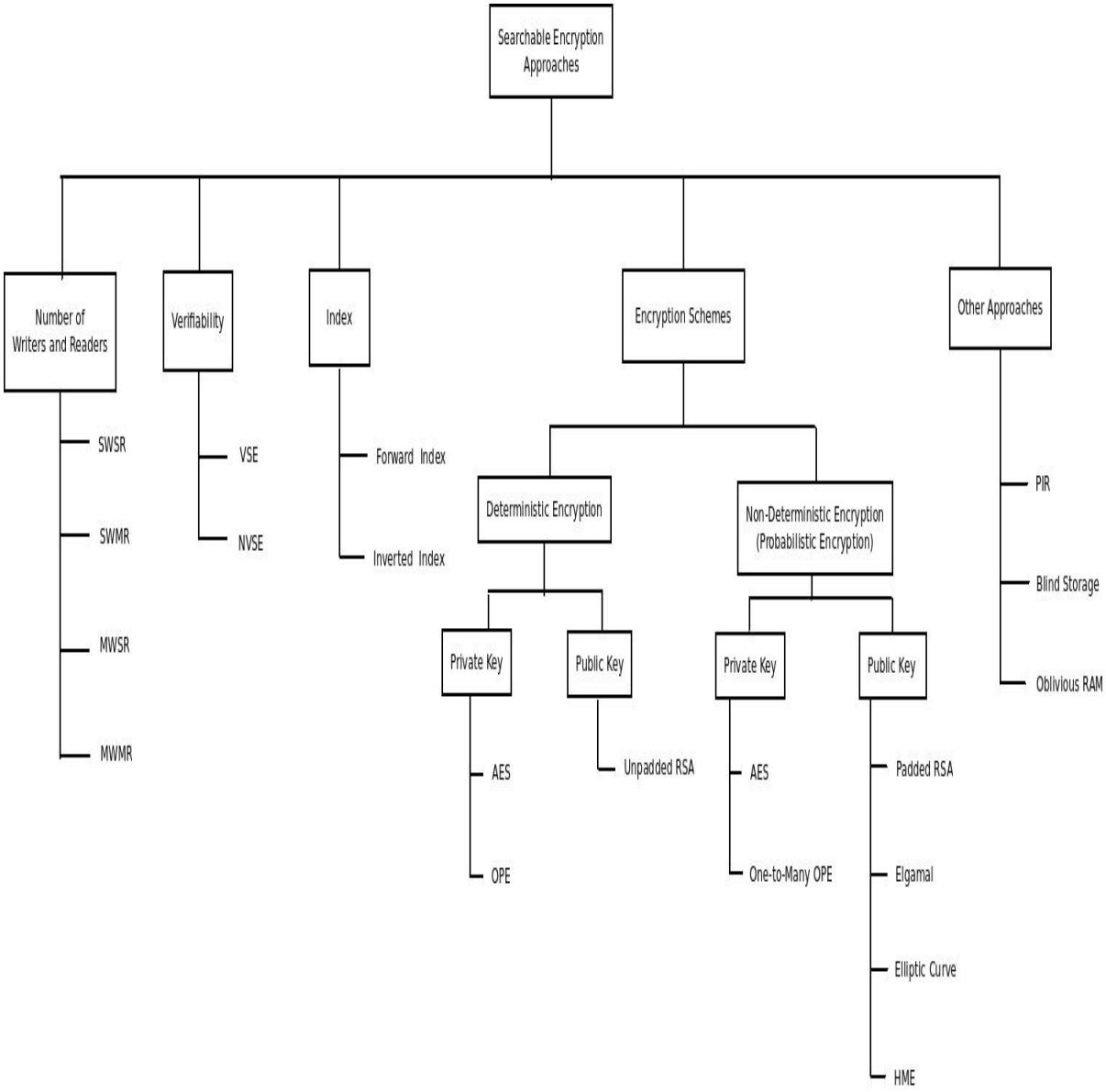


Figure 2.1: Taxonomy of searchable encryption approaches.

CHAPTER 3

PROBLEM DESCRIPTION

In this research work, the primary goal is to develop a multi-keyword search approach over encrypted documents that allows the data users to get the latest top-k relevant documents securely and efficiently for their trapdoors. It has been found from the literature review that the existing searchable encryption approaches are prone to various information disclosure attacks due to the leakages caused by the vulnerabilities of encryption schemes, which are used for encrypting indexes and queries. The leakages include frequency of same ciphertext values in indexes, search pattern information determined from the issued trapdoors, rank-order information determined from the retrieval of the top-k relevant documents. These leakages allow the cloud server to infer plaintext information from the encrypted data through various attacks such as Frequency analysis attack, Scale analysis attack, and Rank-order exploitation attack. Therefore, it is essential to prevent the cloud server from mounting these attacks in order to assure the privacy and confidentiality of outsourced documents. This research work is focused on mitigating the leakages and thereby avoids the possibility of attacks. Besides guaranteeing the privacy of stored documents from these attacks, this work is also focused on providing the latest relevant documents for the users' trapdoors in order to enable users to choose timely decisions based on the availability of updated information. The primary objective of this research work is subdivided into the following objectives.

3.1 OBJECTIVES

1. *Development of an enhanced one-to-many order-preserving mapping technique to mitigate the frequency leakage:* The aim of this work is to develop an Enhanced One-to-Many order preserving encryption (OPE) scheme that can be used to encrypt keywords' relevance scores in indexes without leaking frequency information. As this scheme preserves the order of plaintext values after encryption, the cloud server can perform rank-ordering of the documents directly from the encrypted relevance scores. There exists already some OPE schemes (Boldyreva et al. 2009; Wang et al. 2012), but they leak frequency information, especially when two or more keywords are likely to co-occur equally in the same document (Pan et al. 2020). This leakage is due to the lack of sufficient randomness in the encryption process of existing OPE schemes. The proposed scheme is focused on mitigating the frequency leakage of keywords as well as co-occurring keywords than the frequency leakage of the same caused by the existing OPE schemes.

2. *Development of a privacy-preserving search approach for preventing rank-order and search pattern leakages*: The aim of this work is to develop a privacy-preserving multi-keyword search approach that prevents the leakages of both rank-order information and search pattern without affecting precision. The existing multi-keyword search approaches (Cao et al. 2014; Xiangyu et al. 2019) prevent search pattern leakage by adding random keywords in trapdoors, but they affect precision. Also, these approaches do not prevent rank-order information leakage completely as the cloud server comes to know some of the actual relevant documents for a given trapdoor. Therefore, it is essential to develop an approach that securely retrieves the top-k relevant documents without leaking rank-order information and search pattern information to the cloud server. The proposed approach prevents rank-order information by assigning pseudo-ranks to the documents instead of actual ranks, also prevents search pattern leakage by adding random keywords in trapdoors. The effect of added random keywords on precision would be nullified with the help of an intermediate server adopted in the proposed approach. The proposed approach is focused on returning top-

k relevant documents to the users in a single round of communication without disclosing any secret information to the cloud server and the intermediate server.

3. *Development of an efficient index structure to support dynamic updates:* The aim of this work is to develop an efficient tree-based index structure to support dynamic updates such as Insert, Delete and Modify operations and to retrieve the top-k relevant documents efficiently. In the existing tree-based indexing techniques (Dai et al. 2020; Kamara and Papamanthou 2013; Wu and Li 2019; Xia et al. 2016), only leaf nodes contain information corresponding to the documents and internal nodes contain information required to guide the search. Due to this, the existing tree-based indexes require higher height and breadth to create an index for all the documents of the dataset. Therefore, these index structures involve visiting a lot of unnecessary nodes while searching as well as while performing updates on any nodes of the tree that corresponds to the intended documents. This leads to inefficiency in performing both dynamic updates and the retrieval of top-k relevant documents. This research work is focused on developing an efficient tree-based index structure, where the content of each node, including leaf and internal nodes, represent the documents and thus reduces the height and breadth of the tree. This reduction leads to the improvement in the efficiency of performing dynamic updates as well as retrieval of the top-k relevant documents.

CHAPTER 4

MITIGATION OF FREQUENCY LEAKAGE

4.1 INTRODUCTION

In Searchable Encryption (SE), the information in searchable indexes includes unique keywords and their corresponding relevance scores. There are many keyword relevance score measures like TF (Term Frequency), TF-IDF (Term Frequency-Inverse Document Frequency), and Okapi relevance score. These measures help ranked search approaches to return relevant top-k documents for the given trapdoors (encrypted queries). These relevance scores are significant, and they convey keywords' distribution information in the dataset. This information should not be disclosed to the cloud servers. Hence, these relevance scores need to be encrypted along with the index keywords to prevent the direct leakage of keyword distribution information to the cloud server. The relevance scores should be encrypted in such a way that the cloud server could still perform rank-ordering of the documents directly from the encrypted relevance scores of keywords. To meet this requirement, the encryption scheme is required to preserve the plaintext order of the relevance scores after encryption. Cryptographic algorithms such as RSA, Elgamal, and AES can not be used to encrypt the relevant scores because they do not preserve plaintext order. Therefore, we need encryption schemes that preserve the plaintext order of keywords' relevance scores after encryption. Various encryption schemes exist, i.e., Order Preservation Encryption (OPE) (Boldyreva et al. 2009), One-to-Many OPE (Wang et al. 2012) maintain the order of plaintext values after encryption. However, in OPE, the same plaintext score will always be mapped to the same ciphertext

value due to the usage of plaintext score as the seed in its mapping process. Thus, it leaks the frequency information. In One-to-Many OPE, due to the improvisation of randomness in the seed value compared to the seed value of OPE, the same plaintext score will be mapped to a distinct ciphertext value. Hence, the frequency leakage of ciphertext values of One-to-Many OPE scheme is less than the frequency leakage of OPE scheme. However, this scheme also leaks the frequency information especially when two or more keywords have the same plaintext score within the same document. This allows the cloud server to derive some specific keywords from the encrypted relevance scores by mounting frequency analysis attack (Naveed et al. 2015). The frequency of ciphertext values (i.e., repetition of same encrypted TF values) would enable the cloud server to infer the plaintext keyword through frequency analysis attack by using some background knowledge of the uploaded dataset. The background knowledge could be the information about the uploaded dataset, e.g., the uploaded dataset is related to "computer science," or "United States,". Hence, it is required to prevent or minimize the leakage of frequency information to the cloud server. The objective of this work in this chapter is to mitigate the frequency leakage.

Illustration of frequency leakage: Assume that the plaintext version of the inverted index shown in Figure 4.1 is generated for a set D of some documents, $D_1, D_2, .., D_n$. The corresponding encrypted index is shown in Figure 4.2. In plaintext index, the TF value is stored for each unique word for each document D_i of the document set D. For example, it can be observed in plaintext index that the TF value of W_1 in documents D_1, D_3, D_4 and D_n is same. The frequency, i.e., the repetition of same TF value is 4, which can be observed in plaintext index. It is to be noted that the corresponding TF values after encryption are same value for documents D_1, D_3, D_4 and D_n in the encrypted index. The frequency of same encrypted TF value of the same encrypted keyword $EW1$ in encrypted index is also 4.

Also, the frequency of co-occurring keywords W_1 and W_2 is same in documents D_1, D_4 and D_4 in plaintext index and also the same frequency for these co-occuring keywords in the encrypted index. In spite of only encrypted index is uploaded onto the cloud server, the inference of the corresponding plaintext keyword can be inferred from

the encrypted TF values. This is possible when the cloud server has some background knowledge of the uploaded dataset, e.g., which keywords may occur the most in the given dataset. Therefore, it is necessary to ensure that the frequency of the same ciphertext values of keywords should not be the same as the frequency of corresponding plaintext values.

	D1	D2	D3	D4	D5	.....	Dn
W1	2	1	2	2	0	.....	2
W2	0	1	3	4	5	.....	3
W3	1	0	4	0	1	.....	1
W4	2	1	3	2	1	.....	2
Wm	1	2	1	0	3	.....	2

Figure 4.1: Frequency Leakage in Plaintext Index.

	D1	D2	D3	D4	D5	.....	Dn
EW1	1200	1000	1200	1200	101	.....	1200
EW2	101	1000	1500	1900	2100	.....	1500
EW3	1000	101	1900	101	1000	.....	1000
EW4	1200	1000	1500	1200	1000	.....	1200
EWm	1000	1200	1000	101	1500	.....	1200

Figure 4.2: Frequency Leakage in Encrypted Index.

Need for mitigating the frequency leakage: The existing order preserving encryption (OPE) schemes leak both the order of plaintext information and frequency information due to their deterministic property. Various attacks such as frequency analysis attack (Cash et al. 2015) and also correlation attack (Bindschaedler et al. 2018; Durak et al. 2016; Pan et al. 2020) exploit frequency information from encrypted data to infer plaintext information. Hence, it is required to minimize or prevent the frequency information leakage in order to use it for encrypting sensitive information. Probabilistic encryption schemes like Paillier (Orencik et al. 2013) and Fully Homomorphic encryption schemes (Wu 2015) have been used to prevent frequency information leakage, but they do not preserve plaintext order due to which the relevant documents cannot be identified by the cloud server from the encrypted values. Hence, only the encryption

schemes that preserve the plaintext order should be used. OPE schemes should leak only order information, but frequency information should be avoided. This ordering information helps in performing sorting and comparison operations and thus enables the cloud server to perform rank-ordering the documents. However, the existing Order Preserving Encryption (Boldyreva et al. 2009), and One-to-Many OPE (Wang et al. 2012) schemes leak the frequency information. Therefore, an Enhanced One-to-Many OPE scheme is developed to minimize the frequency leakage information than the leakage caused by the existing OPE schemes.

This chapter contributes to the existing literature in the following ways:

- Proposing an Enhanced One-to-Many OPE scheme to map a plaintext relevance score to a ciphertext value such that it minimizes the frequency leakage of same ciphertext values in indexes and yet it preserves the plaintext order.
- A thorough analysis of the proposed approach with respect to the frequency leakage of keywords, co-occuring keywords and efficiency.

The remainder of this chapter is organized as follows: Section 4.2 presents the preliminary details. Section 4.3 deals with the problem statement and the proposed approach is presented in Section 4.4. Section 4.5 presents the experimental study and analysis of the proposed approach. The summary of this chapter is presented in Section 4.6.

4.2 PRELIMINARIES

The proposed work includes the below notations that are used throughout this chapter.

Term Frequency (TF): There are many measures for determining keywords' weight information such as TF, TF-IDF, and Okapi relevance score. But TF values of the keywords are more repetitive than the others. Hence, the TF value of keyword is used in our work to demonstrate the effectiveness of the proposed approach with respect to the leakage of frequency information. The TF value of a keyword in a document represents the relevance score of the keyword in the corresponding document. TF value

Notation	Description
D	— The input document collection, denoted as a set of n documents $D = (D_1, D_2, ..., D_n)$.
W	— The distinct keywords extracted from D, denoted as set of m keywords $W = (kw_1, kw_2, ..., kw_m)$.
TF_{kw_j,D_i}	— The Term Frequency of keyword kw_j in document D_i.
I	— The plaintext inverted index generated from documents D.
$\widetilde{I}$	— Encryption of the inverted index to be outsourced, which is constructed from I.
C	— The encrypted document collection to be outsourced, denoted as the set of ciphertext documents $C = (C_1, C_2, C_3, ..., C_n)$. Each C_i corresponds to D_i in D.
Q_{kw}	— The query keywords, a subset of W, represents the keywords of user's query.
$\widetilde{M_{Q_{kw}}}$	— The masked query corresponds to Q_{kw}.

of keyword kw can be measured by counting the number of occurrences of a keyword in the given document d_i. As some keywords in longer documents will have higher TF values, the TF value is normalized by dividing it with the length of the document. The equation used for normalizing the TF value of keyword kw in a document D_i is given below:

$$TF(kw, D_i) = \frac{1}{|d_i|}(1 + \log(tf_i)) \tag{4.1}$$

where,$|D_i|$ - Length of the document, i.e., the total number of keywords in document d_i tf_i - The number of keyword kw occurrences in a document d_i.

Coordinate Matching: It is a similarity measure used to determine the relevance score of the documents based on the presence of query keywords in the document Witten et al. (1994).

4.3 PROBLEM STATEMENT

It is defined as follows: For a given set of plaintext TF values of keywords of the documents D, the proposed Enhanced One-to-Many OPE scheme maps each plaintext TF value to a ciphertext value such that it minimizes the frequency leakage of same ciphertext values for any keyword in encrypted indexes and yet it preserves the plaintext

order. This approach enables the cloud server to determine relevant documents for a given trapdoor while minimizing the leakage of frequency information to the cloud server from the encrypted relevance scores of keywords in indexes.

System Architecture: The schematic view of the system architecture shown in Figure 4.3 is considered in this work. It consists of a Data owner, Data users, and the Cloud server (CS). The data owner creates searchable indexes, which are in encrypted form, for all his/her plaintext documents. The data owner also encrypts the documents and uploads both of them onto the CS. Whenever the data users need to find the documents for their queries, they generate the trapdoors, which are then submitted to the CS. The CS processes the trapdoors and then assigns the score to each document using a similarity measure. Then, the most relevant top-k documents will be sent to the users. It is assumed that the keys required for generating trapdoors and decrypting the retrieved documents are shared with the data users by the data owners through a secure channel.

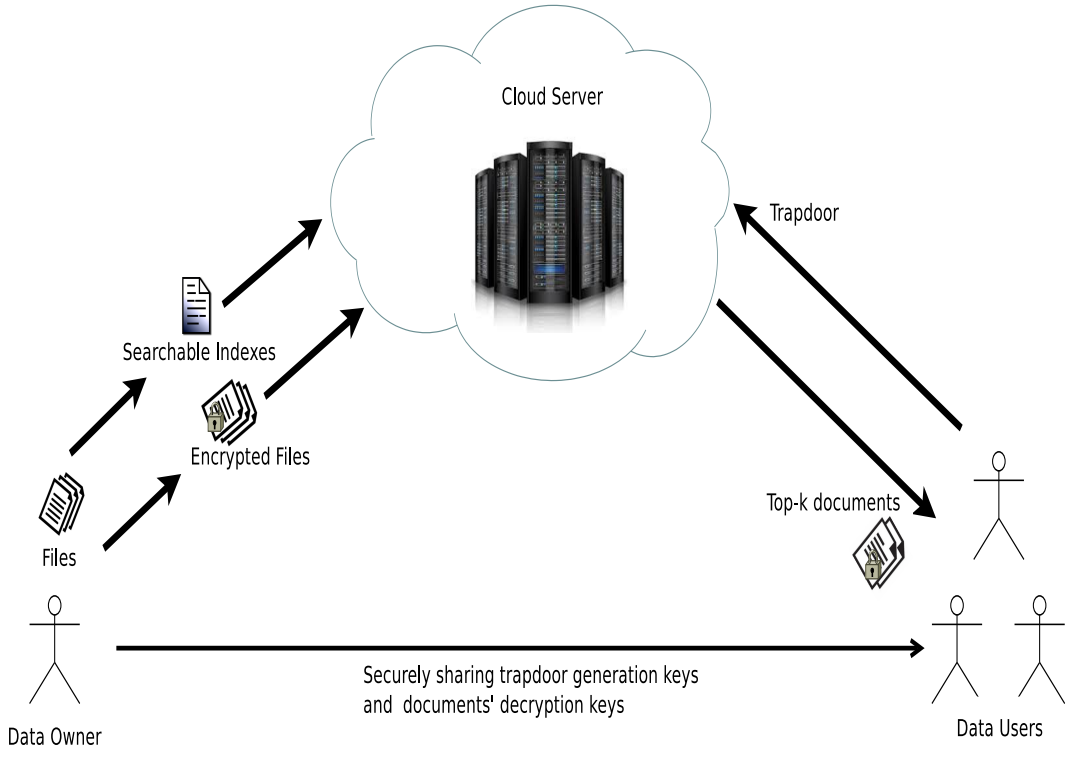


Figure 4.3: A Schematic view of the searchable encryption system.

4.4 ENHANCED ONE-TO-MANY OPE SCHEME

The aim of the proposed Enhanced One-to-Many OPE scheme is to mitigate the frequency leakage while enabling the cloud server to determine relevant documents for a given trapdoor. The proposed scheme consists of two phases,i.e.,1) Initialization phase and 2) Retrieval Phase. The initialization phase involves generating encrypted indexes and encrypted documents and then uploading both of them onto the cloud server for storage, as shown in Figure 4.1. The proposed Enhanced One-to-Many OPE scheme can be used as the encryption scheme for encrypting keywords' relevance scores in the initialization phase. The retrieval phase involves generating trapdoor for the given query and sending it to the cloud server, which processes it and sends back the users the top-k relevant documents. Each of these phases is explained in detail as follows.

Initialization Phase: This phase includes the following activities to be done by the data owner:

1. *Build the Encrypted Index:* The steps required to generate the encrypted index $\widetilde{I}$ for all the documents of dataset D is explained as follows:

   (a) *Building Dictionary W:* Construct the dictionary W by extracting all the unique keywords kw from the input documents of the dataset D.

   (b) *Building Plaintext Index I:* For each keyword kw in W, determine its TF value as per the equation (4.1) for each document D_i if it is present in document D_i and store it in index as $I[kw]= [D_i][TF]$.
   Otherwise, set its TF value to 0 if it is not present, i.e., $I[kw]= [D_i][0]$.

   (c) *Generate Encrypted Index $\widetilde{I}$:* The content of the above generated index I is encrypted as follows:

   - Each keyword in I is hashed by using a one-way secure hash algorithm SHA-2 with a 256-bit key.
   - The TF value of each keyword kw of every document D_i is encrypted using the proposed Enhanced One-to-Many OPE scheme, which is explained in next page.

- The document identities are not necessary to encrypt as they do not convey any information about the content of the documents. The encrypted index generated would then be: $\widetilde{I} = [\widetilde{TF}][D_i]$.

2. *Encrypting Documents:* After generating the encrypted index, the data owner also encrypts his/her entire document collection D using AES algorithm with 128 bit key size.

Retrieval Phase: This phase includes the following activities to be done by the data users:

1. *Query_Masking:* To retrieve the documents of users' interest, a masked query must be generated by the data user. The masked query $\widetilde{M_{Q_{kw}}}$ is generated by hashing each keyword of his/her query Q_{kw} using a secure SHA-256 hash function. Then, the data user sends the masked query and also the parameter k to the cloud server to send only the relevant top-k documents.

2. *Searching:*

   (a) The cloud server upon receiving $\widetilde{M_{Q_{kw}}}$, it utilizes the encrypted index $\widetilde{I}$ and adopt the coordinate matching similarity measure (Witten et al. 1994) to assign the scores to each document D_{id}. Then it sorts the scores of the documents in descending order and sends the top-k of them to the user.

   (b) The data user then decrypts the received documents with the corresponding secret key shared by the data owner through a secure channel.

Enhanced One-to-Many OPE Scheme

The proposed Enhanced One-to-Many OPE scheme returns a possible unique ciphertext value c for each keyword kw's TF value, i.e., a plaintext relevance score $pscore$ by mapping it to one of the output range of values. The procedure for mapping a $pscore$ to a ciphertext c is explained in Algorithm 1. It takes Key (K), input domain (D) , output range (R), plaintext relevance score ($pscore$), document identity $id(D)$, and keyword (kw) and returns a possible unique ciphertext value c. During mapping, the range R

is divided into some non-overlapping interval buckets each with different size. Each bucket contains some range of values. For the given $pscore$, the random-sized bucket is determined using a Binary_Search(.) procedure, which is explained in Algorithm 2. Binary_Search(.) is a recursive procedure, which returns a new domain and a new range of values for a given $pscore$ based on an HYGEINV(.) function. HYGEINV(.) is a hypergeometric sampling process that returns an integer value based on the initial domain, range, and middle value. This integer helps Binary_Search(.) in choosing a new domain, and a new range of values for the given $pscore$. In each iteration of binary search, the size of domain D and range R will be reduced to half. Binary_Search(.) stops when the size of the domain becomes 1 during when the domain contains only the given $pscore$. The $pscore$ then will be mapped to one of the values in a new range R using a TapeGen(.) function. TapeGen(.) is a random coin generator, which can be used for generating the seed value. This seed value helps in choosing one of the values in the new range as the ciphertext value c for the given $pscore$.

Algorithm 1: Enhanced One-to-Many OPE

Input: K (Key), D (domain), R (range), $pscore$, $id(D)$, kw
Output: Cipher text c

while $|D|! = 1$ **do**
  $\{D, R\} = Binary\_Search(K, D, R, pscore)$
coin $\leftarrow$ TapeGen(K,(D,R,1$||pscore$, $id(D)$, kw)
$c \xleftarrow{coin} R$
return c

4.5 THEORETICAL AND SECURITY ANALYSIS

In this section, the reversibility, time complexity, and security analysis of the proposed scheme are presented.

a) Reversibility

The proposed Enhanced One-to-Many OPE scheme is also reversible, i.e., returns a plaintext score $pscore$ for the given ciphertext value c. Reversibility would be helpful in many applications where the underlying plaintext values are frequently modified or

Algorithm 2: Binary_Search

Input: K, D, R, $pscore$
Output: D,R

$M \leftarrow length(D)$; $N \leftarrow length(R)$
$d \leftarrow min(D) - 1$; $r \leftarrow min(R) - 1$
$y \leftarrow r$+ceil($\frac{N}{2}$)
$coin \xleftarrow{R}$ TapeGen(K,(D,R,0$||y$))
$x \xleftarrow{R} d$ + HYGEINV($coin$,M,N,$y - r$)
if $pscore \leq x$ **then**
    $D \leftarrow \{d + 1, ..., x\}$
    $R \leftarrow \{r + 1, ..., y\}$
else
    $D \leftarrow \{x + 1, ..., d + M\}$
    $R \leftarrow \{y + 1, ..., r + N\}$
return D,R

when they are further used for some other computations. The reversibility procedure for obtaining a plaintext value $pscore$ for a given c is explained in Algorithm 3. It takes key k, ciphertext value c, the document identity $id(D)$ and keyword kw as arguments and returns a plaintext value $pscore$ that corresponds to c. In this process, the given c value would be mapped to the same bucket that the plaintext value was mapped to using the Binary_Search(.) procedure explained in Algorithm 4. The Binary_Search(.) returns a new domain and new range recursively for the given c value, and it stops when the size of the domain becomes 1. The value left in this domain is a minimum value of D, and it would be returned as the plaintext value $pscore$. To confirm the $pscore$ if it really corresponds to c, the $pscore$ is passed as an argument to the TapeGen(.) procedure along with the document identity $id(D)$ and keyword kw. TapeGen() generates a seed value using which one of the values in a new range R would be chosen as the new ciphertext value c_{new}. If the generated c_{new} and the initial c values are same, then the $pscore$ would be returned as a plaintext score for the given ciphertext value c. Thus, the proposed scheme returns a plaintext score $pscore$.

Algorithm 3: Reversibility of Enhanced One-to-Many OPE

Input: K (Key), D (domain), R (range), c (ciphertext value), $id(D)$, kw
Output: $pscore$

```
while |D|! = 1 do
    {D, R} = Binary_Search(K, D, R, c)
pscore ← min(D)
coin ← TapeGen(K,(D,R,1||pscore, id(D), kw)
c_new ←coin— R
if c_new == c then
    return pscore
else
    return ⊥
```

b) Time Complexity

The time complexity of mapping a plaintext score $pscore$ to a ciphertext c is *O(log n)*, i.e., *log n* times the Binary_Search(.) process will be called for mapping a single plaintext score $pscore$ to a value c. The time complexity of mapping n elements from a domain D to n elements in range R using the proposed approach is *O(nlogn)*.

c) Security Analysis

The aim of the proposed scheme is to prevent the cloud server from exploiting the frequency information from the encrypted keywords' relevance scores (TF values) in indexes. As the index keywords are hashed using a secure hash function SHA-256, it is difficult for anyone to break the SHA because it is a one-way hash function. For a given any hash value, it is impossible to reverse engineer the hash value to find its corresponding keyword. Similarly, the privacy of query keywords is also maintained, since query keywords are also hashed using the SHA-256 algorithm. The privacy of index keywords from the encrypted TF values is explained in the below subsection.

Privacy of index keywords' from the encrypted TF values: The TF values of keywords in indexes are encrypted using the proposed Enhanced One-to-Many OPE. As per the experimental results observed in Section 4.4, the proposed scheme minimizes the frequency leakage of not only individual keywords' relevance scores but also the co-occurring keywords. Thus, it mitigates the possibility of frequency analysis attack

Algorithm 4: Binary_Search

Input: K, D, R, c
Output: D,R

$M \leftarrow length(D)$; $N \leftarrow length(R)$
$d \leftarrow min(D) - 1$; $r \leftarrow min(R) - 1$
$y \leftarrow r$+ceil($\frac{N}{2}$)
$coin \xleftarrow{R}$ TapeGen(K,(D,R,0$||y$))
$x \xleftarrow{R} d$ + HYGEINV($coin$,M,N,$y - r$)
if $c \leq y$ **then**
    $D \leftarrow \{d+1, ..., x\}$
    $R \leftarrow \{r+1, ..., y\}$
else
    $D \leftarrow \{x+1, ..., d+M\}$
    $R \leftarrow \{y+1, ..., r+N\}$
return D,R

on Enhanced One-to-Many OPE values. It can also be observed from the experimental results that the proposed approach cannot prevent the frequency leakage completely especially when plaintext TF values are more densely distributed, i..e., same TF value for numerous keywords in the same document. Therefore, proposed approach is recommended to use it for encrypting sensitive data that is moderately distributed. For an appropriate domain D, and range R, where $|R| \geq 2|D|$, the proposed Enhanced One-to-Many OPE scheme does not leak frequency information and is secure under the frequency of ordered chosen plaintext attack (FOCPA) as per the definition of IND-FOCPA. The corresponding definition and the proof of the proposed scheme are explained below.

Definition (IND-FOCPA): For the given domain D and range R, the proposed Enhanced One-to-Many OPE scheme (K,E,D) guarantees indistinguishable ciphertexts under the frequency of ordered chosen plaintext attack (FOCPA). K refers to secret key, E refers to the encryption procedure, and D refers to the decryption procedure explained in Algorithm 3. The proposed scheme is secure under FOCPA only if the adversary has negligible advantage in the security parameter *'k'*:

$$\left| P_r[Exp^{A}_{FOCPA}(k,1) = 1] - P_r[Exp^{A}_{FOCPA}(k,0) = 1] \right| \tag{4.2}$$

where $Exp^{A}_{FOCPA}(k,b)$ is the experiment, which is explained here.

Experiment $Exp^{A}_{FOCPA}(k, b)$

$(X_0, X_1) \leftarrow A$, where $|X_0| = |X_1| = s$ and let X_i be the sequence of the form $kw_1 : v_1(d_1)$, $kw_2 : v_2(d_2)$, ..., $kw_j : v_k(d_l)$, where kw_j refers to j^{th} keyword, v_k refers to keywords' relevance score, i.e., (TF) value, and d_l refers to the identity of the document and also let X_0 and X_1 have common frequency for at least one keyword in X_i, i.e., frequency of at least one keywords' relevance score in both the sets is same.
Select a bit uniformly $b \in \{0, 1\}$ and choose X_b from X_0 and X_1.
For all $1 \leq i \leq s$, $y_{b,i} \leftarrow E(K, x_{b,i})$
$b' \leftarrow A(y_{b,1}, y_{b,2}, ..., y_{b,s})$
Output 1 if and only if $b = b'$

Theorem: The proposed Enhanced One-to-Many OPE scheme is IND-FOCPA.

Proof: Let A is assumed to be an arbitrary adversary for the game in IND-FOCPA. Also, assume that X_0 and X_1 are two plaintext sequences that are chosen by the A. As per the definition of IND-FOCPA, these two sequences have at least common frequency for any element of the sequences.

When encrypting either X_0 or X_1, the proposed Enhanced One-to-Many OPE scheme uses TF value, Keyword, and document identity $id(D)$ for generating a seed value that decides where the TF value will be mapped to one of the values within the range R. Hence, the proposed approach is independent of the frequency of the input plaintext sequence and therefore independent of the chosen bit b. The information that A receives from the challenger (E) is therefore independent of the selected bit 'b' and thus A can only guess what 'b' is. This concludes the proof.

4.6 EXPERIMENTAL STUDY AND ANALYSIS

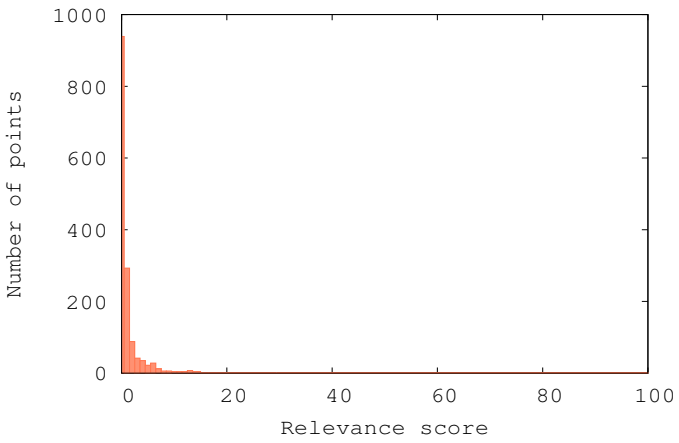
In this section, implementation methodology, experimental results of frequency leakage of the proposed Enhanced One-to-Many OPE scheme, efficiency, and its usage are presented.

4.6.1 Implementation Methodology

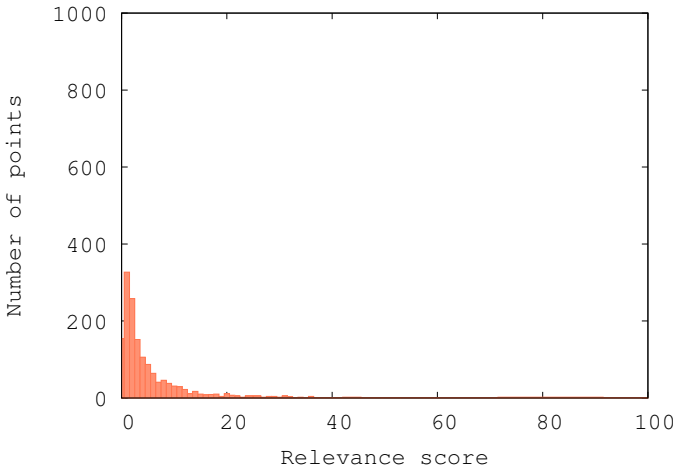
The proposed approach has been implemented using Python 3.6 version and tested on Intel i7-4770 CPU system. We have conducted experiments on Requests for Comments (RFC) (RFC 2016) dataset. For evaluating the proposed approach, frequency leakage, i.e., the repetion of same ciphertext values in index and the time efficiency of performing encrypted operations are considered. The proposed approach is assessed by practically inferring plaintext information of frequently occurring keywords by observing frequency leakages from the encrypted TF values of indexes through frequency analysis attack. The frequency leakage of the proposed approach is compared with the OPE and One-to-Many OPE schemes for different keywords as well as some phrases (co-occuring keywords) of RFC dataset.

4.6.2 Experimental Results

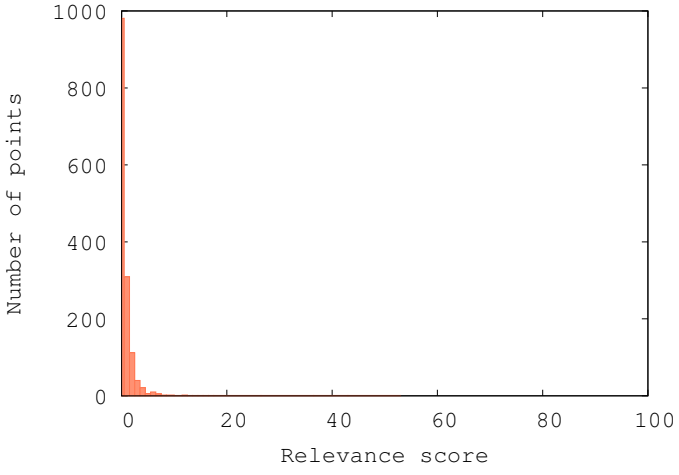
Frequency Information: The proposed Enhanced One-to-Many OPE scheme is an extension of One-to-Many OPE (Wang et al. 2012), which in turn is an extension of OPE (Boldyreva et al. 2009). In OPE, the plaintext score $pscore$, .i.e., the TF value of the keyword is mapped to a ciphertext, which is a value within the specified output range. In our experiments, the input domain is the actual plaintext relevance scores, and the output range is set between 0 and $2^{45}-1$. While mapping, initially the range R is divided into some non-overlapping interval buckets each with different size. For the given plaintext TF value $pscore$, the random-sized bucket is determined by binary search (Algorithm 2) with the help of HYGEINV(.) function. One of the values within the bucket is chosen as a ciphertext value for the given $pscore$ based on the seed value that is generated by TapeGen(.). In OPE, this seed value is completely dependent on the plaintext score $pscore$ due to which the same $pscore$ is mapped to the same ciphertext value within the values of the bucket. Therefore, the OPE is a deterministic encryption scheme; wherein, the same plaintext value would always be mapped to the same ciphertext value. Because of this deterministic property, the frequency of ciphertext c is leaked. This frequency leakage is due to the usage of plaintext score $pscore$ as the seed in mapping process of OPE. Hence, the distribution of ciphertext values in OPE



(a)



(b)



(c)

Figure 4.4: Plaintext keyword relevance score distribution for keywords: (a) computer (b) network (c) communication

follows the same distribution of plaintext scores. Therefore, the frequency leakage of ciphertext values cannot be prevented with OPE. For the given dataset, if plaintext score is repeated n number of times, then there will be a ciphertext c that will also be repeated n number of times. Frequency leakage, i.e., distribution information can be exploited by frequency analysis attack to infer the specific plaintext keywords of the index.

In RFC dataset, the keywords "computer", "network", and "communication" are the frequently occurring keywords, and they appear in most of the documents. Hence, the relevance scores of these keywords in the index of this dataset will be higher than the relevance scores of other keywords. The plaintext distribution information of these three keywords is respectively shown in figures Figure 4.4.a, 4.4.b and 4.4.c respectively. The values on the x-axis represent the actual plaintext TF values and values on y-axis represent the frequency, i.e., the number of times a particular TF value is repeated. As the OPE maps the same plaintext score to the same ciphertext, the distribution of ciphertext values of these keywords would be the same as the plaintext distribution. Hence, if the cloud server has some idea (background knowledge) that the keyword "network" is more likely to appear in the uploaded dataset than other keywords, then it could easily guess that plaintext keyword from the encrypted relevance scores. To infer the plaintext keyword, the cloud server plots the graphs for each encrypted keyword 's relevance scores in index. Then, it will note down the encrypted or masked keyword of the index for which the frequency (the repetition of the same encrypted score) is higher than the frequency of scores of other encrypted keywords. The keyword "network" is more likely to be this encrypted keyword. Thus, the cloud server infers the keyword.

Comparison with the existing approach: The frequency leakage of the existing One-to-Many OPE scheme is compared with the proposed Enhanced One-to-Many OPE scheme. In One-to-Many OPE scheme (Wang et al. 2012), for the given plaintext score $pscore$, it uses the same algorithm 2 (Binary search) to select the bucket, but TapeGen(.) uses both the $pscore$, and document identity $id(D)$ for generating a different seed value for the same $pscore$. Due to this seed, it maps the same plaintext to different ciphertext value within the values of the bucket. With One-to-Many OPE, frequency information leakage can be prevented only when there exist the same plaintext

relevance scores in different documents. However, there is a scope for the frequency leakage when there are same plaintext scores for two or more keywords within the same document. For example, assume that the plaintext scores of keywords $kw1$ and $kw2$ are identical in the document d_i. As this approach uses both plaintext score $pscore$ and document identity id(D) to generate the seed, the same plaintext score of these keywords in the same document is likely to be mapped to the same value within the values of the bucket. This is possible especially when some keywords have the equal occurrences in the same document, e.g., the keywords of the phrases like { "computer", "network"} or { "communication", "network"}, have the same number of occurrences in some of the documents of RFC dataset. Therefore, the plaintext scores of the keywords in these phrases are likely to be mapped to the same ciphertext. If One-to-Many OPE scheme is used for encrypting the scores, then the cloud server can infer the keywords of these phrases from the encrypted scores. The cloud server first plots the histogram for the relevance scores of each encrypted index keyword. Then it identifies the histogram in which the frequency of the same relevance score is higher than the other keywords in the index. Then it looks for the histogram of another keyword whose frequency of relevance scores is closer to the frequency of the chosen histogram. Then it combines the relevance scores of these two encrypted index keywords and plots the graph. Similarly, it does the same for other histograms, in which the frequency of relevance scores is closer to the already chosen histogram. Then the cloud server identifies the histogram of the phrase in which the frequency of relevance score is higher than the frequency of other possible phrases. The cloud server notes the histogram of encrypted keywords of this phrase. These encrypted keywords are more likely to be the most occurring keywords of the phrases. Thus, the cloud server could infer the plaintext keywords of the phrases from the One-to-Many OPE encrypted scores.

The proposed Enhanced One-to-Many OPE scheme, which is explained in Section 4.4, minimizes the frequency leakage of the phrases caused by the One-to-Many-OPE scheme. This approach also uses the same algorithm 2 (Binary search) to select the bucket, but the TapeGen(.) here uses $pscore$, document identity $id(D)$ and keyword kw for generating a different seed value for the same plaintext score $pscore$. Due to

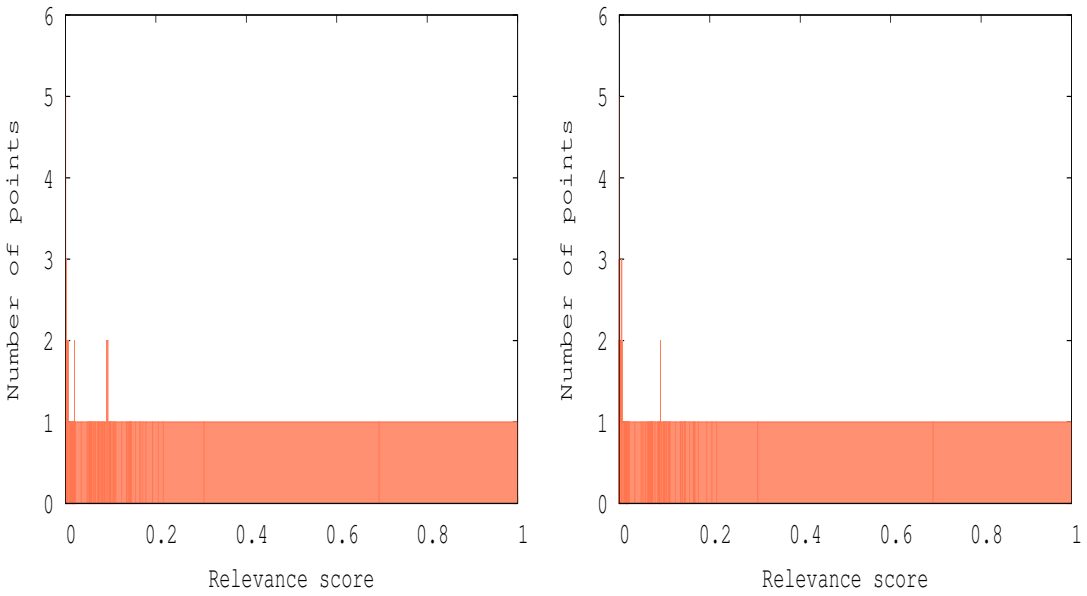
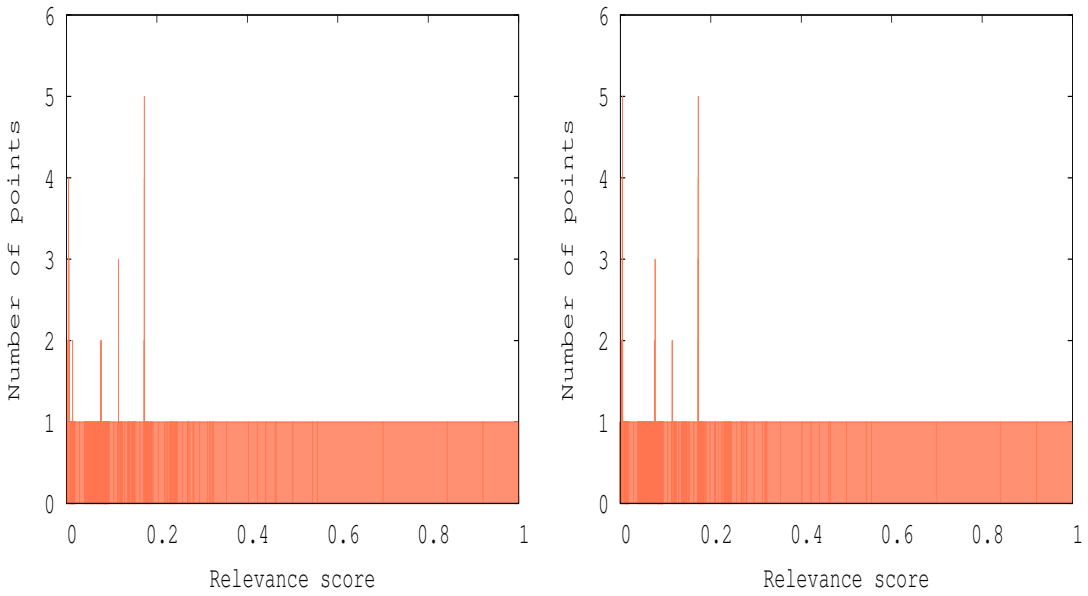


Figure 4.5: Encrypted score distribution for ”computer”: (a) One-to-Many OPE (b) Enhanced One-to-Many OPE

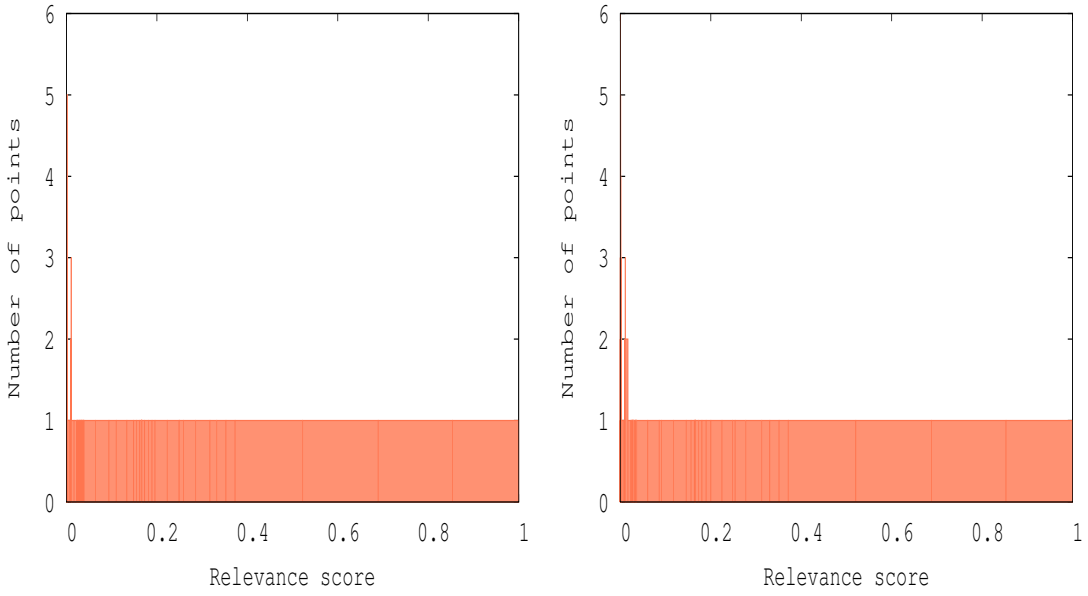
this seed, the same $pscore$ will be mapped to a different value within the values of the bucket even if there exists the same plaintext score for multiple keywords in the same document. Thus, it minimizes the frequency leakage of keywords of the phrases. This minimization is due to the improvement of the randomness in generating the seed value. Due to this seed, this approach reduces not only the frequency leakage of phrases but also the individual keyword's frequency information.

We have compared the frequency leakage of One-to-Many OPE encrypted scores and the proposed Enhanced One-to-Many OPE encrypted scores for the keywords ”computer”, ”network”, ”communication”. The distribution information (frequency) of One-to-Many encrypted scores and the proposed Enhanced One-to-many encrypted scores for the keyword ”computer” is shown in figures Figure 4.5.a and 4.5.b respectively. The values on x-axis represent the normalized One-to-Many OPE encrypted scores using the min-max normalization approach (Margae et al. 2014). Normalization is necessary because the encrypted values are very large numbers that can not be shown clearly in graphs. The values of the y-axis represent information about the frequency, i.e., the number of points with the same encrypted TF value. Similarly, the comparison of rele-



(a) (b)

Figure 4.6: Encrypted score distribution for ”network”: (a)One-to-Many-OPE (b) Enhanced-One-to-Many-OPE



(a) (b)

Figure 4.7: Encrypted score distribution for ”communication”: (a)One-to-Many-OPE (b) Enhanced-One-to-Many-OPE

vance scores for the keywords ”network” and ”communication” is also shown in figures Figure 4.6.a, 4.6.b, 4.7.a, and 4.7.b respectively. It can be observed in these figures that the frequency leakage of certain encrypted scores using the proposed Enhanced One-

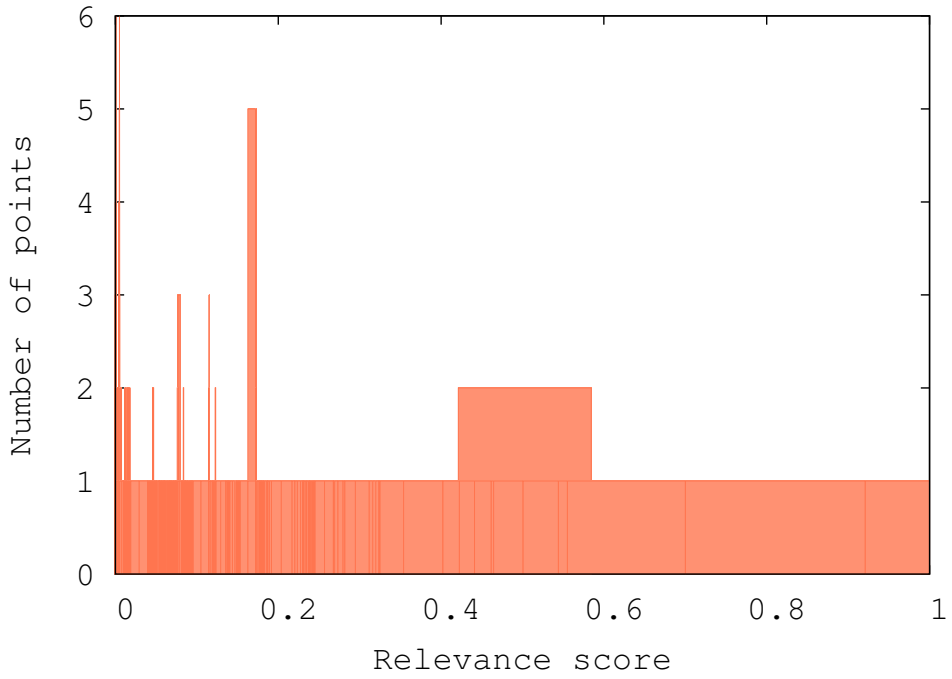


Figure 4.8: Encrypted score distribution of One-to-Many OPE scheme for ”Computer Network”

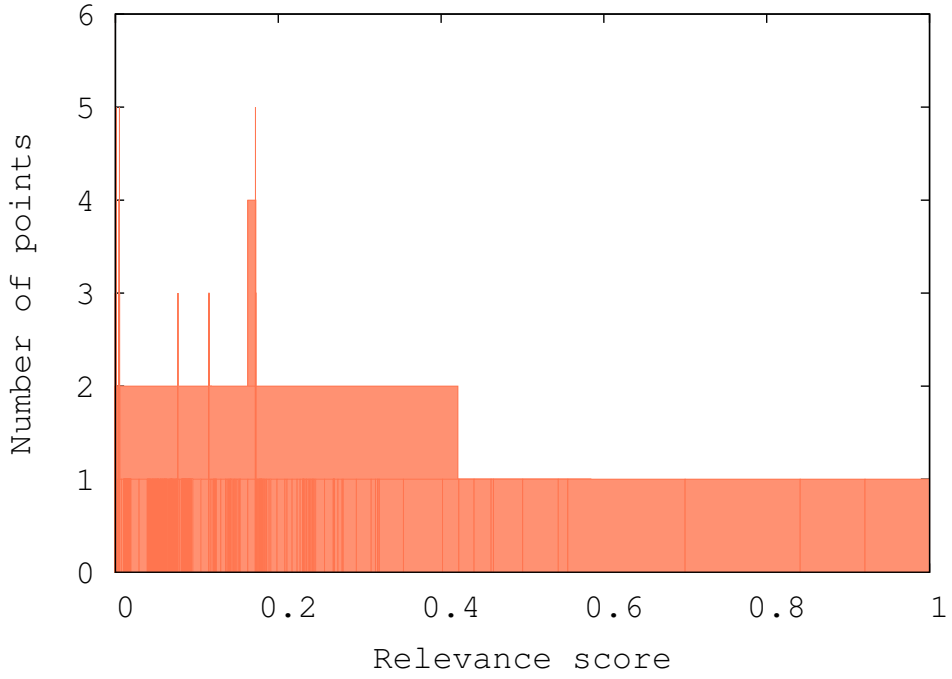


Figure 4.9: Encrypted score distribution of Enhanced-One-to-Many OPE scheme for ”Computer Network”

to-Many OPE scheme is less than the frequency leakage of One-to-Many OPE for all the keywords (”computer,” ”network,” ”communication”) respectively.

As it is already mentioned, One-to-Many-OPE scheme cannot prevent the frequency leakage of phrases whose keywords are likely to co-occur equally in the same document. The proposed scheme minimizes the frequency leakage of phrases due to the usage of plaintext score $pscore$, document identity id(D) and keyword kw for generating a different seed value. Due to this seed, the different value within the values of the bucket will be chosen as the ciphertext value for the same plaintext relevance score $pscore$. In RFC dataset, ”computer network” and ”communication network” are the most occurring phrases. The distribution information of One-to-Many-OPE encrypted scores and the proposed Enhanced One-to-Many-OPE encrypted scores for the phrase ”computer network” is shown in figures Figure 4.8 and 4.9, respectively. Similarly, figures Figure 4.10 and 4.11 respectively show the distribution information for the phrase ”communication n1etwork”. It can be observed that the frequency leakage of the proposed approach shown in figures Figure 4.9 and 4.11 is much lesser than the frequency leakage of the One-to-Many OPE scheme shown in figures Figure 4.8 and 4.10 respectively. Therefore, the proposed approach leaks less frequency information than the frequency leakage of One-to-Many OPE scheme. Thus, the proposed approach makes it difficult for the cloud server for inferring plaintext keywords of the phrases from the encrypted relevance scores. Thus, the privacy of index keywords from the encrypted relevance scores is improved using the proposed Enhanced One-to-Many OPE scheme.

Efficiency: We have compared the efficiency of the proposed approach experimentally with the efficiency of the One-to-Many OPE. The time cost comparison of mapping a $pscore$ to c over different domain and ranges using the proposed Enhanced One-to-Many OPE and One-to-Many OPE schemes is shown in Figure 4.12.a. The x-axis values in this figure represent domain size and the values on y-axis represent the amount of time taken to map a plaintext score $pscore$ to a ciphertext value c. Y-axis represents the average time of 100 trails for a single mapping operation over different domain sizes and a range $|R| = 2^{45}$. It is to be noted from this figure that the efficiency of the proposed Enhanced One-to-Many OPE and the existing One-to-Many

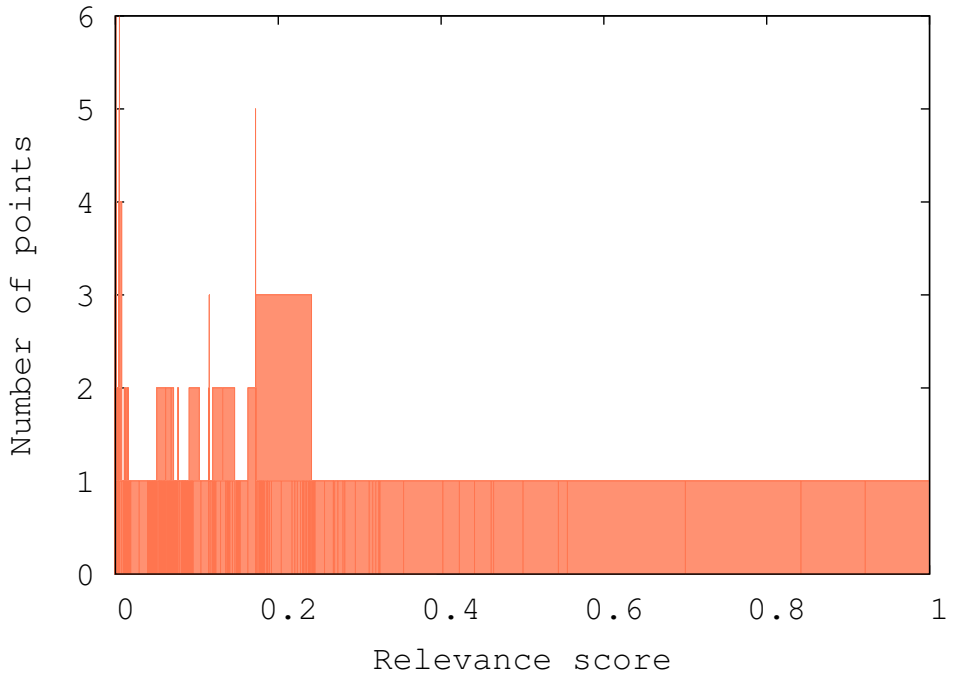


Figure 4.10: Encrypted score distribution of One-to-Many OPE scheme for ”communication network”

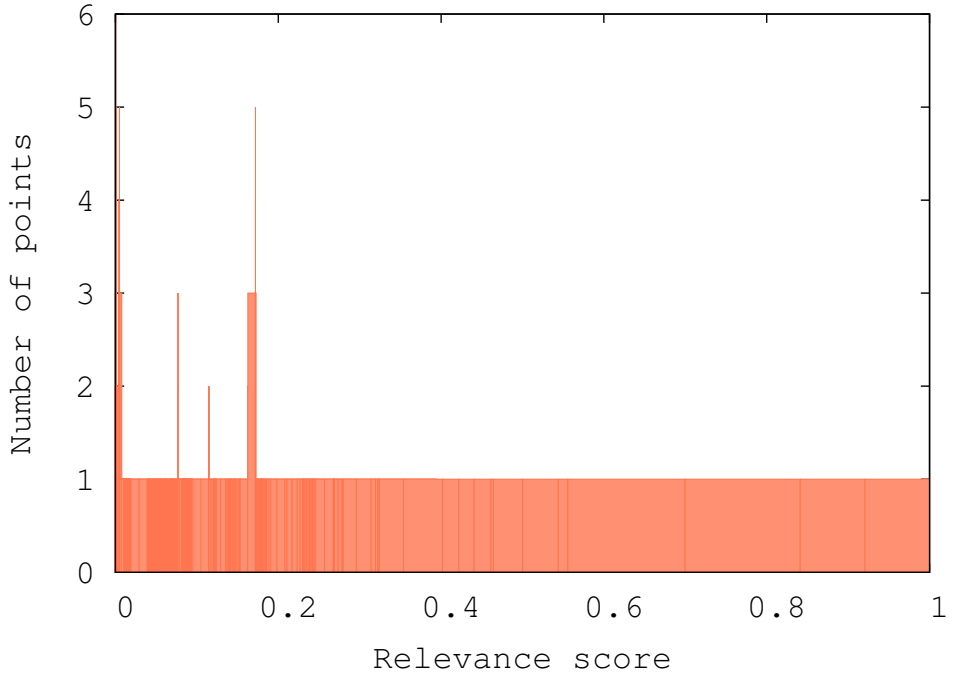


Figure 4.11: Encrypted score distribution of Enhanced One-to-Many OPE scheme for ”communication network”

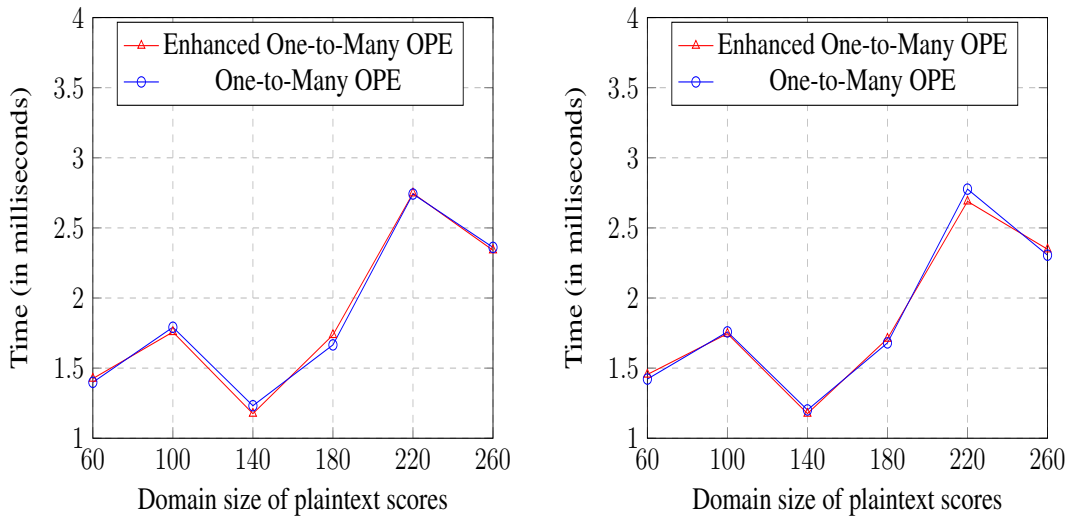


Figure 4.12: (a) The time cost comparison of single mapping operation using Enhanced One-to-Many OPE and One-to-Many OPE over a range $|R| = 2^{45}$. (b) The time cost comparison of a single reversing operation using Enhanced One-to-Many OPE and One-to-Many OPE over a range $|R| = 2^{45}$.

OPE is almost same. Similarly, the reversibility (decryption) efficiency of the proposed Enhanced One-to-Many OPE and One-to-Many OPE is shown in Figure 4.12.b. The reversibility procedure for returning a plaintext value for the given ciphertext value c is explained in algorithm 3. It is to be noted that the reversibility efficiency is also same for both the schemes. From the figures Figure 4.12.a and 4.12.b, it can be observed that efficiency of mapping and reversibility is almost same since both of them call the Binary_Search procedure recursively same number of times. Also, to be noted that the time of mapping and reversibility operations take slightly higher time for some domains that are smaller in size than the domains of larger size due to the variation in the number of times the Binear_Search(.) procedure is called for a given plaintext score $pscore$. Overall, the maximum time the proposed Enhanced One-to-Many OPE scheme takes for performing both mapping and reversibility operations is 300 milliseconds, which is reasonably efficient in today's computing environment.

4.6.3 Usage of the proposed Enhanced One-to-Many OPE scheme

The proposed scheme can be used for encrypting both unstructured data, .e.g, encrypted search engines where documents are used and also structured database, e.g., health care applications and banking applications, where patients and customers sensitive data is stored in their databases in the form of tables, e.g., balance of a customer account, salary of a doctor and age of a patient. The proposed scheme can be used for encrypting the data of all these numeric columns on top of which some arithmetic operations can be performed. However, the only problem with the approach is that the proposed approach should be used only when the distribution of plaintext values is not high otherwise, frequency information would be leaked.

4.7 SUMMARY

In this chapter, an Enhanced One-to-Many OPE scheme is proposed and developed for mapping a given plaintext score to a ciphertext value uniquely while preserving the order of plaintext scores. Our experimental study confirms that the proposed scheme reduces not only the frequency leakage of keywords but also the co-occurring keywords. Reduction of frequency leakage mitigates the possibility of frequency analysis attack. Thus, the proposed approach makes it difficult for the cloud server to infer the plaintext keywords by observing the frequency leakage of encrypted relevance scores of keywords in indexes. The proposed approach may prevent complete frequency leakage if it is used for encrypting information that is moderately distributed. Hence, the proposed approach is recommended to use it for encrypting sensitive information that is moderately distributed.

CHAPTER 5

PREVENTION OF RANK-ORDER AND SEARCH PATTERN LEAKAGES

5.1 INTRODUCTION

In Searchable Encryption (SE), there exists some other privacy concerns due to the generation of same trapdoor for the same query and also due to the retrieval of documents in descending order based on their ranks for the given trapdoors. The rank-order information and the search pattern leakage may enable the cloud server to infer plaintext keywords of trapdoors, and documents through Rank-order exploitation attack and Scale analysis attack. To prevent these leakages, the existing SE approaches like (Cao et al. 2014) add random keywords in both indexes and trapdoors and assign ranom values to these random keywords. The addition of different random keywords in a trapdoor generation approach leads to the generation of different trapdoor for the same query. Thus, search pattern leakage is prevented. However, addition of random keywords affect the search accuracy, i.e., precision. since the cloud server returns few non-relevant documents as a response to these random keywords. Also, the existing approaches cannot prevent the rank-order information leakage completely as the assignment of random values to the random keywords in their indexes follow the standard deviation (σ) of existing values of actual keywords. Due to which, the cloud server can still determine rank-order information for some of the returned documents in those approaches. Thus, they cannot prevent the rank-order exploitation attack. It has been found in the literature that there exists no approach that achieves precision while preventing the leakages of

both rank-order information and search pattern. Hence, there exists a need for a precision and privacy preserving multi-keyword search approach to address this issue. The objective of this work is to guarantee the privacy of rank-order information and search pattern without affecting precision.

Need for higher precision and privacy without affecting precision: The data users who retrieve the documents of their interests through trapdoors expect that the cloud server does not come to know about what they are searching (search pattern) and what they are retrieving (rank-order information). Since the cloud server has access to searchable indexes and also the trapdoors, which are processed by the cloud server itself, the exposure of rank-order information and search pattern to the cloud server leads to the Rank-order exploitation attack and Scale analysis attack respectively, which are explained in Section 1.3. These attacks enable the cloud server to infer plaintext information of frequently issuing query keywords or the most occurring keywords of the dataset. Hence, it is required to prevent both the leakages in order to ensure the required security. However, as security and precision contradict each other, i.e., precision might be compromised to some extent if privacy is more preferred, and privacy would be compromised if precision is more preferred. For example, to achieve higher precision, it is required to send top-k relevant documents to the users' for their trapdoors. But, the rank-order information would be leaked to the cloud server while sending the documents. Thus, the privacy of rank information would be compromised while guaranteeing higher precision. Similarly, search pattern privacy can be prevented by adding random keywords in a trapdoor generation approach, but precision would be affected due to the random keywords. It has been observed from the literature review that achieving higher precision and privacy together using a single server would be impossible. But both the data owners and users expect higher privacy as well as higher precision. In this chapter, a pseudo-ranking approach is developed to prevent both the rank-order and the search pattern leakages without affecting precision with the help of two servers, namely, i.e., cloud server and the intermediate server.

This work contributes to the existing literature in the following ways:

- Proposing a Pseudo-Ranking approach to prevent the cloud server from the exploiting rank-order information and search pattern information without affecting precision.
- A thorough analysis of the proposed approach with respect to the security and efficiency .

The remainder of the chapter explained as follows: Section 5.2 presents the information on preliminaries. Section 5.3 presents the problem statement and the proposed approach proposed approach is presented in Section 5.4 . Section 5.5 presents the experimental study and analysis of the proposed approach. The summary of this chapter is presented in Section 5.6.

5.2 PRELIMINARIES

Term Frequency-Inverse Document Frequency (TF-IDF): For returning relevant documents, it is necessary to store keywords' relevance scores in indexes. Keyword's score conveys the importance of the corresponding keywords in documents and the dataset. Two measures are used frequently in literature, e.g., Term Frequency (TF) or Term Frequency-Inverse Document Frequency (TF-IDF). Among all these measures, we have used the TF-IDF measure in our approach for storing the relevance of keywords in indexes since the TF-IDF measure is the most frequently used, and it balances the impact of frequently occurring keywords and rarely occurring keywords. The TF-IDF value of a keyword can be by obtained by multiplying TF and IDF values. TF value can be obtained by counting the number of times the keyword kw_i occurs in the document D_i and IDF value can be obtained by dividing the total number of documents (N) in the dataset with the document frequency of keyword df_kw. and The document frequency of keyword df_{kw} is the number of document that contain given keyword kw. The TF-IDF value of the keyword of a document D_i be computed using (5.1).

$$TF - IDF(kw, D_i) = \frac{1}{|D_i|}(1 + \log(TF_i)) \log(\frac{N}{df_{kw}}) \tag{5.1}$$

where $|D_i|$ is the number of keywords present in the document D_i.

Enhanced One-to-Many OPE: We have used the proposed Enhanced One-to-Many OPE scheme, which is explained in Section 4.4 for encrypting the TF-IDF values of keywords. As it preserves the order of plaintext TF-IDF values after encryption, the ranks of documents can be determined directly by a server from the encrypted TF-IDF values.

Coordinate Matching: It is a similarity measure used to assign ranks, i.e., scores to the documents based on the presence of query keywords in the documents. The rank (or relevance score) of the document D_i for the given query Q_i, which consists of *'k'* keywords, is computed using (5.2).

$$Rank(D_i, Q_i) = \sum_{1 \leq j \leq k} TF - IDF(i, j) \tag{5.2}$$

5.3 PROBLEM STATEMENT

For any given one or more trapdoors, it is required to return the top-k relevant documents to the users without leaking rank-order and search pattern information to the cloud server. The aim of work presented in this chapter is to develop a Pseudo-Ranking approach that preserves the privacy of rank-order information by assigning pseduo-ranks to documents instead of actual ranks and it also preserves search pattern privacy by adding more random keywords in a trapdoor generation approach. An intermediate server has been adopted in the proposed approach to nullify the impact of added random keywords of a trapdoor.

System Architecture: The system architecture shown in Figure 5.1 has been adopted in our proposed Pseudo-Ranking approach to ensure both higher privacy and higher precision. It consists of a Data owner, a set of Data users, the Cloud server (CS) and the Intermediate server (IS). The activities performed by each of the entities involved in this architecture is explained as follows:

- The data owner creates indexes for his/her documents based on plaintext information and encrypts the indexes and then uploads them onto the CS. The data owner also uploads the encrypted documents and the encrypted randomized indexes, which contain information about only random keywords, onto the IS.

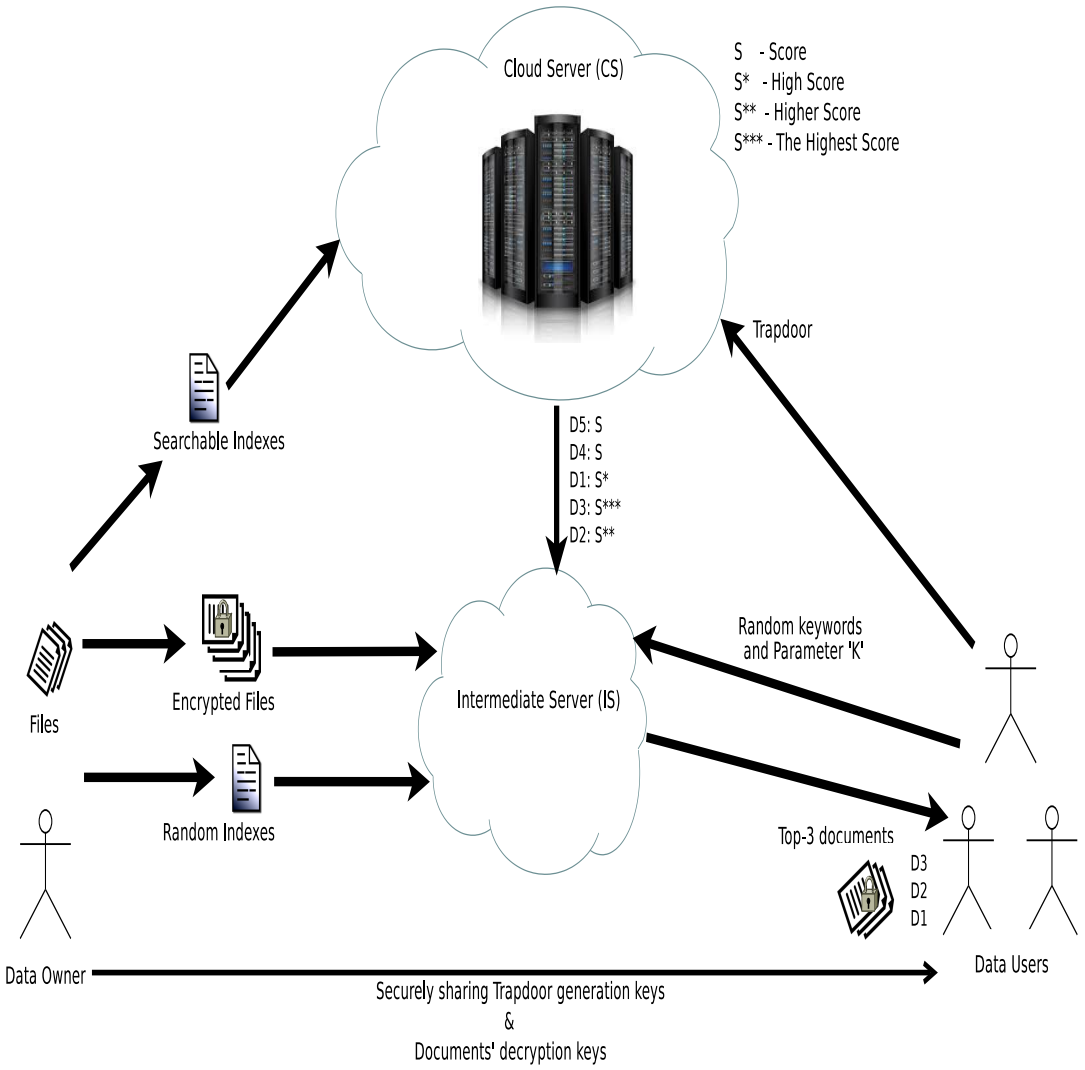


Figure 5.1: Schematic view of the proposed system architecture

- The data users retrieve the documents by generating the trapdoors, i.e., the encrypted queries, which are then submitted to the cloud server. The data users also send the encrypted random keywords of the trapdoor to the IS.

- The CS processes the trapdoor and then assigns the score to each document Id based on the given trapdoor. The document identities D_{id}'s along with their scores are sent to the IS. The relevant documents will not be in the top-k of the list of the documents sent by the CS.

- The IS based on the encrypted random keywords received from the user, determines the actual top-k relevant documents and sends them to the users.

It is assumed that the keys required by the data users for generating trapdoors and the random keywords to be added in a trapdoor are shared by the data owners through a secure channel. It is also assumed that there exists no collusion between the IS and the CS. S in the Figure 5.1 indicates scores of the returned documents and $*$ indicates the relevance of the documents for the given trapdoor. More $*$ indicates the most relevant the documents are.

5.4 PSEUDO-RANKING APPROACH

To retrieve the relevant documents for the given user's trapdoors without compromising privacy and precision, the proposed approach includes the following set of activities to be done by a Data Owner, Data User, Cloud Server, and Intermediate Server respectively.

5.4.1 Generation of Encrypted Indexes

The Data owner generates encrypted indexes and encrypted documents. The steps required to create the encrypted indexes for a given dataset are explained below.

i. *Dictionary construction:* Extract a unique set of keywords $KW = \{kw_1, kw_2, ..., kw_m\}$ from the data owner's dataset by removing punctuation marks, case folding, stopwords and applying lemmatization on each keyword of the dataset.

ii. *Plaintext indexes:* A set of indexes $I = \{I_1, I_2, ..., I_n\}$ are generated, where each index I_i is created for each document d_i in the dataset $D = \{d_1, d_2, ..., d_n\}$. The index of each document stores the following information:

   - Each keyword kw_i of the dictionary KW.
   - Store the TF-IDF value for each keyword kw_i that is determined using (5.1) if it occurs in the document. Otherwise, set *TF-IDF*= 0.

iii. *Randomized indexes:* Generate a set of randomized indexes $RI = (RI_1, RI_2, ..., RI_n)$. Each RI_i corresponds to I_i in I. Each randomized index is generated by performing the following activities:

- Determine the minimum min, maximum max *TF-IDF* values of index I and determine their mid value as $\frac{(min+max)}{2}$. Also, determine the average score of the document $Avg\_score(D)$ using TF-IDF values of actual keywords of each document D_i from index I.
- Generate a set of random keywords $RKW = \{rkw_1, rkw_2, ..., rkw_m\}$ that is same in size as the size of dictionary D.
- Add these random keywords to each document in its index I_i and assign the values to these random keywords based on document's score, which is explained as follows:
  - for each document D_i in dataset D, determine its score $dscore(D_i)$.
  - For each random keyword rkw in random keyword set RKW, assign random values to them uniquely as follows:
    - If $dscore(D_i) >= Avg\_score(D)$, then assign values to these random keywords uniformly in the range from min to $(\frac{min+mid}{3})$.
    - Else, assign uniformly in the range from $(\frac{mid+max}{1.8})$ to max.

iv. *Encrypted randomized indexes:* Generate the encrypted randomized index, $\widetilde{ERI} = (\widetilde{ERI_1}, \widetilde{ERI_2}, \widetilde{ERI_3}, ..., \widetilde{ERI_n})$, where each $\widetilde{ERI_i}$ corresponds to RI_i in RI. The generation of each encrypted index $\widetilde{RI_i}$ is explained as follows:

- Each keyword in RI_i is first hashed with the SHA-256 secure cryptographic hash function and then the resultant hash value is then encrypted using the below function f.

$$f : \{0,1\}^k \times \{0,1\}^l \rightarrow \{0,1\}^m \tag{5.3}$$

  where k, l, and m are equal in length. The first part in this equation represents a secret key parameter, the second part represents the hash value and the output of f is a ciphertext value.
- Keywords' relevance score, i.e., *TF-IDF* value is encrypted by using the proposed Enhanced One-to-Many OPE scheme, which is explained in Section 4.4.

Once the encrypted indexes are generated, the data owner separates the encrypted random keywords and the corresponding encrypted values of each index from $\widetilde{ERI}$. The data owner then also encrypts the documents using AES algorithm. The data owner then uploads both the encrypted documents and the separated randomized indexes, which contain only the encrypted random keywords with their values on to the intermediate server and also uploads the main encrypted indexes $\widetilde{ERI}$ on to the cloud server. The data owner also shares the entire random keyword set and the secret keys with the data users via a secure channel to generate trapdoors and decrypt the retrieved documents.

5.4.2 Generation of Trapdoors

The data user is required to generate the trapdoor, i.e., an encrypted query to retrieve documents of his/her interest. The generation of the trapdoor for user query is explained as follows:

- The data user first randomizes his/her plaintext query by adding more number of random keywords than the actual query keywords.
- Generates a trapdoor $\widetilde{T_{Q_{kw}}}$ by encrypting each keyword of the randomized query using both the SHA-256 hash function and the pseudo-random function (5.3). The trapdoor $\widetilde{T_{Q_{kw}}}$ is then sent to the cloud server. The data user also sends the encrypted random keywords and the parameter 'k' to the IS as shown in Figure 5.1. The parameter 'k' is to request the IS to send only the top-k relevant documents.

5.4.3 Retrieval of Top-k Relevant Documents

- *Score assignment to the documents:* Upon receiving a trapdoor $\widetilde{T_{Q_{kw}}}$, the cloud server uses the encrypted index $\widetilde{ERI}$ and uses the coordinate matching similarity measure (5.2) to assign scores to each document identity D_{id} for the given trapdoor. The document identities D_{id}'s with their scores are sent to the IS in descending order of scores.
- *Determining top-k documents:* The IS after receiving the D_{id}'s and their corresponding scores, uses the encrypted random keywords of the trapdoor (sent by

the data user) and the encrypted indexes of random keywords to find out the top-k relevant documents. This is explained as follows:

- The IS first determines the sum of encrypted *TF-IDF* values of the random keywords of the trapdoor for each document D_i using the encrypted indexes, which contains only the encrypted random keywords and their encrypted random values.
- Subtracts the value of sum from each document's score received from the cloud server and updates the corresponding document's score.

After the assignment of updated scores to each document, the IS then sorts the documents based on newly assigned scores in decreasing order and sends the top-k documents among them to the user. The data user finally decrypts the received documents with the corresponding secret key shared by the data owner.

5.5 THEORETICAL AND SECURITY ANALYSIS

In this section, the time complexity, and security analysis of the proposed Pseudo-Ranking approach are presented.

a) Time Complexity

The time complexity of the proposed Pseudo-Ranking approach is analyzed with respect to the index construction, trapdoor generation and search time.

- *Index construction:* It is required to create an index for each document in the dataset as forward indexing technique (per document indexing technique) is adopted in this approach to enable updates of the indexes efficiently, i.e., for adding random keywords in each document index. Time complexity of index construction is $(O(n(m+m^{'})))$, where n is number of documents in the dataset, m is the number of actual keywords in the dictionary and $m^{'}$ is the number of random keywords added in the dictionary for each document index.
- *Trapdoor generation:* To retrieve the top-k relevant documents securely, the user generates the trapdoor for his/her keywords of the query. Before generates trap-

doors, he/she adds some random keywords to the list of his/her actual query keywords. These keywords are then encrypted using SHA-256. and a pseudo-random function. As the data user can issue any keywords of his/her interest and can select any number of random keywords, the time complexity for generating trapdoor is $(O(m + m^{'}))$.

- *Search time:* Upon receiving a trapdoor, the cloud server processes the trapdoors and assigns scores to the documents and sends them to the IS in decreasing relevance order of the trapdoors. The IS then determines actual scores of the documents by subtracting the sum of encrypted values of added random keywords of trapdoor from the cloud server assigned scores. The IS then sort the documents in descending order using updated scores and sends the top-k documents to the user. The average search time complexity of generic trapdoor based approaches is O(nm+nlogn), and randomized trapdoor based approaches is $O(n(m + m^{'}) + nlogn)$. The search time complexity of our approach is $O(nm^{'} + nlogn + n(m + m^{'}) + nlogn)$, which is little higher than randomized trapdoor based approaches due to the re-ordering operation at the IS.

b) Security

As two servers are involved in processing each user's trapdoors for returning top-k relevant documents, the proposed approach is assessed against the threats of the cloud server and the intermediate server.

i) Privacy guarantee at the Cloud Server : The proposed approach prevents the cloud server from the exploitation of available information, i.e., encrypted indexes, which contains the encrypted information of actual and random keywords of the documents, and user's trapdoors for inferring plaintext information by guaranteeing the privacy of the following:

- *Privacy of indexes:* Each document's index size is kept same in our proposed approach so that the cloud server cannot distinguish the content of one document index from another document index. The index of each document contains keywords and their corresponding *TF-IDF* values. The keywords of indexes are en-

crypted using both a secure SHA-256 and a pseudo-random function (5.3). Due to the irreversible nature of one-way hash functions, it is computationally hard for the cloud server to infer plaintext information from the hash values of query keywords. However, as the hash functions are deterministic, keyword guessing attacks are prone to occur on hash values. Hence, the hash values are further encrypted in our approach using a pseudo-random function (5.3) to prevent the keyword guessing attacks. The *TF-IDF* values are encrypted using the proposed Enhanced One-to-Many OPE scheme Wang et al. (2012), which returns a possible unique ciphertext value for a given plaintext *TF-IDF* value. Due to this scheme, the same ciphertext values would not be there in the uploaded indexes irrespective of same plaintext *TF-IDF* values. Hence, the cloud server cannot exploit frequency information Naveed et al. (2015) to infer the plaintext index keywords from the encrypted TF-IDF values. Thus, the privacy of indexes is preserved guaranteed.

- *Privacy of trapdoors*: The user's trapdoor consists of actual query keywords and random keywords. Like index keywords, the trapdoor is generated by encrypting each keyword using both an SHA-256 secure one-way hash function and a pseudo-random function. Thus, the plaintext keywords of the trapdoors cannot be inferred. The proposed Pseudo-Ranking approach guarantees search pattern privacy since it generates different trapdoor for the same query due to the inclusion of more random keywords while generating trapdoors.
- *Rank privacy:* In general, the cloud server gets to know which documents are more important and which ones are not with the provision of ranking functionality. This information allows the cloud server to infer possible plaintext information from the rank-ordered documents through rank-order exploitation attack. In the proposed approach, the cloud server does not come to know the actual ranks of the documents due to the perturbed assignment of random values to the random keywords in indexes.

ii) Privacy guarantee at the Intermediate Server (IS): This server holds random indexes containing encrypted random keywords with the corresponding encrypted ran-

dom values, encrypted documents uploaded by the data owner and the encrypted random keywords of trapdoors sent by the data user and also the document Ids and their scores sent by the cloud server. As the IS has access only to the encrypted information related to random keywords, it cannot infer any other possible information about the actual keywords of the trapdoors and indexes. It is explained in the following cases about how the privacy of actual keywords of indexes and trapdoors can be guaranteed.

1. *Top-k documents:* The IS comes to know the rank-order information as it returns the top-k documents to the users. However, as it has no access to the actual keywords of the trapdoors and indexes, no other information can be inferred from the rank order information alone. Without relating the rank-order information to either trapdoors or indexes, it cannot infer any plaintext information.

2. *Exploitation of access pattern:* The IS returns top-k relevant documents to a user in response to his/her trapdoors sent to the cloud server. The access pattern, i.e., sets of documents returned to the user's trapdoors, is leaked to the IS. Various attacks such as access pattern exploitation attack (Cash et al. 2015), volume attacks (Grubbs et al. 2018) and database reconstruction attacks (Lacharité et al. 2018) could be used to exploit the access pattern leakage to infer plaintext information. However, the access pattern alone is not sufficient for these attacks to infer the underlying plaintext information. The success of these attacks depends on both the access pattern information and the knowledge of keywords' distribution information. In our proposed approach, assume that the IS knows the keywords' distribution knowledge of the databaset by guessing, but it cannot be exploited as the IS cannot perform any analysis using this information because the encrypted index of the actual dataset is stored at the cloud server and the IS has access to the encrypted indexes only that contain encrypted random keywords and their encrypted random values. Similarly, the actual query keywords of randomized trapdoors are sent to the cloud server, and the IS has access to the encrypted random keywords of trapdoors. Therefore, the access pattern is leaked in our proposed pseudo-ranking approach using two servers, but plaintext information cannot be inferred from it.

3. *Possibility of collusion:* It has already been mentioned in Section 5.1, that the intermediate server and the cloud server in the the proposed System architecture do not collude, but even if the intermediate server colludes with the cloud server, still no information is revealed to them directly as they both have no access to actual plaintext keywords information and the keys used for encrypting them.

5.6 EXPERIMENTAL STUDY AND ANALYSIS

In this section, implementation methodology, experimental results of the proposed approach, its efficiency, and usage are presented.

5.6.1 Implementaion Methodology

The proposed approach has been implemented using Python 3.6 version, and the experiments are conducted on an instance of the Google Cloud Platform that has two virtual CPUs with 12 GB RAM and 40GB Solid State Drive. We have used 1000 documents of RFC (RFC 2016) dataset for testing the proposed approach. After preprocessing these documents, i.e., removal of punctuation, and stopwords, a total of 20000 unique keywords are included (10000 actual keywords of documents + 10000 random keywords) in each document index. For evaluating the proposed approach, the evaluation parameters such as rank privacy, i.e., the privacy of each returned document in the top-k retrieved documents, precision, i.e., the fraction of the relevant results within the top-k documents, search pattern, and the time of index construction and processing time, i.e., search operation of generic trapdoors and randomized trapdoors are considered.

5.6.2 Experimental Results

The performance details of the proposed pseudo-ranking approach with respect to precision and privacy are provided as follows:

Precision and Rank Privacy: The search results of a given trapdoor can be evaluated using Precision and Rank privacy. The cloud server sends all the documents based on their ranks (scores) in decreasing relevance order for the given trapdoor. Precision and Rank privacy are evaluated for the top-k documents among the returned documents; as these top-k documents only the users show interest. Precision at point k can be

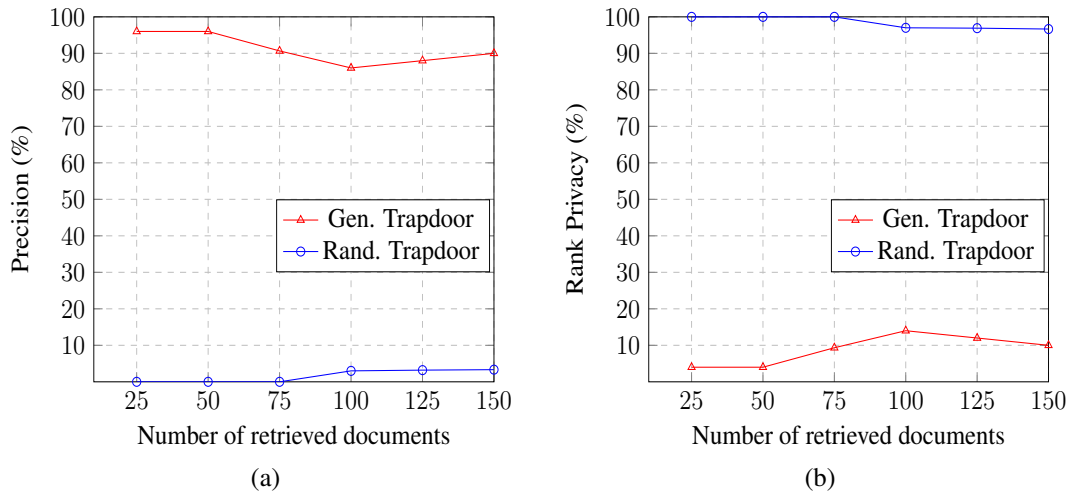


Figure 5.2: (a) Precision of top-k retrieved documents when 1000 documents are indexed at the cloud server. (b) Ranking Privacy of top-k retrieved documents when 1000 documents are indexed at the cloud server.

evaluated as $P_k = \frac{k^{'}}{k}$, where $k^{'}$ indicates the number of real relevant documents among the retrieved top k documents. The precision of the proposed approach at the cloud server is shown in Figure 5.2.a. The *Gen. Trapdoor* in this figure indicates a generic trapdoor that contains only the data user's query keywords while the *Rand. Trapdoor* indicates a randomized trapdoor that includes both random keywords and actual query keywords. Rand. Trapdoor contains a total of 100 keywords that includes 5 actual query keywords and 95 random keywords. It can be observed in Figure 5.2.a that the precision for randomized trapdoor is very less. The less precision at the cloud server indicates less scope for information exploitation and thus, guarantees a higher scope for achieving rank privacy. The rank privacy can be defined as the fraction of the number of non-relevant documents within the top-k documents. It can be determined using $\widetilde{RP_k} = \frac{k^{''}}{k}$, where $k^{''}$ indicates the number of non-relevant documents within the k documents. The more non-relevant documents within the top-k, the more rank privacy it guarantees. Figure 5.2.b shows the rank privacy of the proposed approach at the cloud server. As precision and rank privacy contradict with each other, privacy gets affected if precision is high and vice versa. It can be observed in Figure 5.2.a that precision at the cloud server is very low for the randomized trapdoor because of more random keywords in trapdoor than the actual keywords. The more we add random keywords, the more the

precision it gets affected. It can be observed in Figure 5.2.b that rank privacy at the cloud server is very high (average of 96%) for the randomized trapdoor. Therefore, to achieve higher rank privacy at the cloud server, it is required to include more number of random keywords in a trapdoor than the actual number of query keywords and thereby rank-order exploitation attack can be prevented entirely.

Among all the documents returned by the cloud server, the top-k of them cannot satisfy the user requirements because they are not the actual relevant documents. This is because of the assignment of higher random values (from min to $\frac{min+mid}{3}$) or lower values ($\frac{mid+max}{1.8}$ to max) to the random keywords in each document index based on a document's score. An intermediate server is employed to minimize the impact of these random keywords' values. This server has the encrypted indexes containing the encrypted random keywords and their corresponding encrypted values sent by the data owner and also the encrypted random keywords of the trapdoor sent by the data user. This server then reassigns the documents' scores, i.e., subtracts the sum of random keywords' weights from the scores of the documents sent by the cloud server. The intermediate server then sorts the documents in decreasing order based on updated scores and sends the top-k to the user. Thus, the user information requirements can be fulfilled without compromising precision using our proposed approach. The precision of the top-k documents that the user receives for his/her trapdoor is shown in Figure 5.3.a. It can be observed that precision at the user side is same (average of 91%) for both generic trapdoor and randomized trapdoor. Therefore, the proposed approach achieves not only higher rank privacy but also precision.

In general, the users go through only the top 10 documents instead of going through all 'k' returned documents like people generally do in Google search results. The precision and rank privacy of top-10 documents at the cloud server is shown in Figure 5.4.a and Figure 5.4.b respectively. It can be observed in Figure 5.4.b that rank privacy is 100 % while the precision is zero for the randomized trapdoor at the cloud server. Thus, for the randomized trapdoor, the cloud server cannot exploit the rank order information from the returned documents. This is due to the presence of random keywords in trapdoors and the random values assigned to these random keywords using the pro-

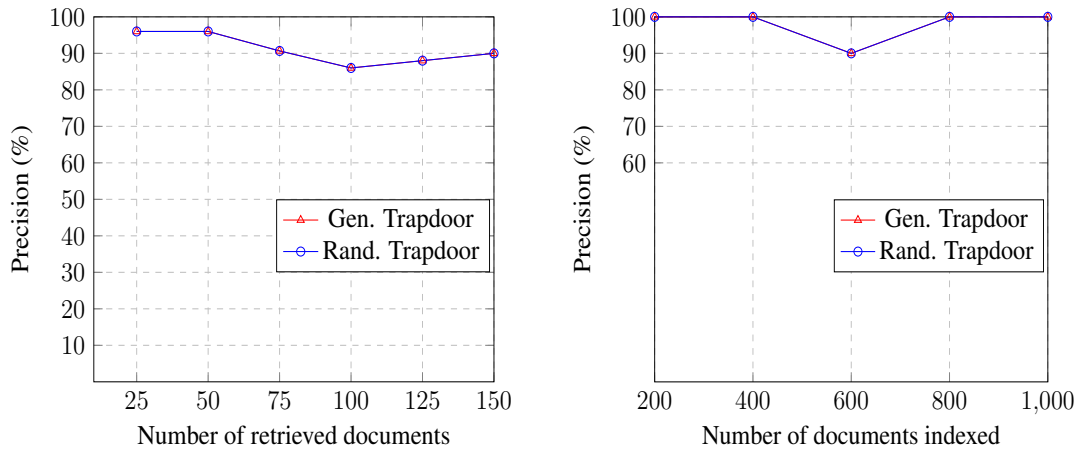


Figure 5.3: (a) Precision of top-k retrieved documents at the user side when 1000 documents are indexed. (b) Precision of top-10 retrieved documents at the user side when different number of documents are indexed.

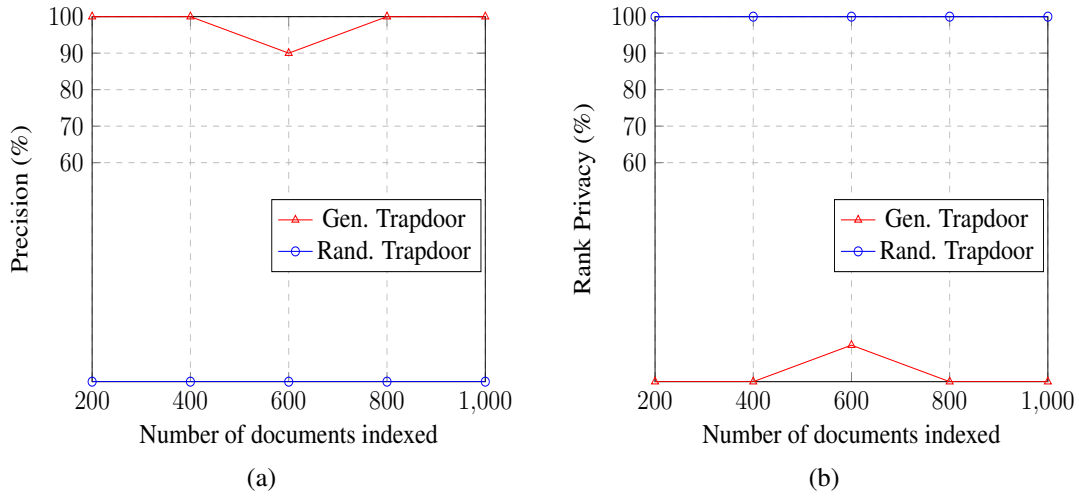


Figure 5.4: (a) Precision of top-10 documents when different number of documents are indexed at the cloud server. (b) Ranking Privacy of top-10 documents when different number of documents are indexed at the cloud server.

posed pseudo-ranking approach. The users expect higher precision beside privacy, the precision of the proposed approach at the user's side for top-10 documents is shown in Figure 5.3.b. It is to be noted in this figure that the precision of top-10 documents decreases from 100% to 90% over the index of 600 documents. This is due to the usage of Enhanced One-to-Many OPE scheme, which maps the same plaintext TF-IDF value to different ciphertext values. For example, the same plaintext values 2,2 of documents

$d1$, $d2$ would be mapped to 1000,1400 respectively, due to which, the rank of $d2$ would be higher than the rank of document $d1$. Therefore, the document $d1$ would not be in top-10 documents. Thus, precision is slightly affected. Overall, the average precision of our approach for a randomized trapdoor is 98%, which is same as the precision for a generic trapdoor. The proposed approach achieves higher ranking privacy because of the assignment of constrained random values to the random keywords in indexes and allows the users to include more number of random keywords in trapdoors. Similarly, higher precision is also achieved by the proposed approach due to the re-order operation performed by the intermediate server.

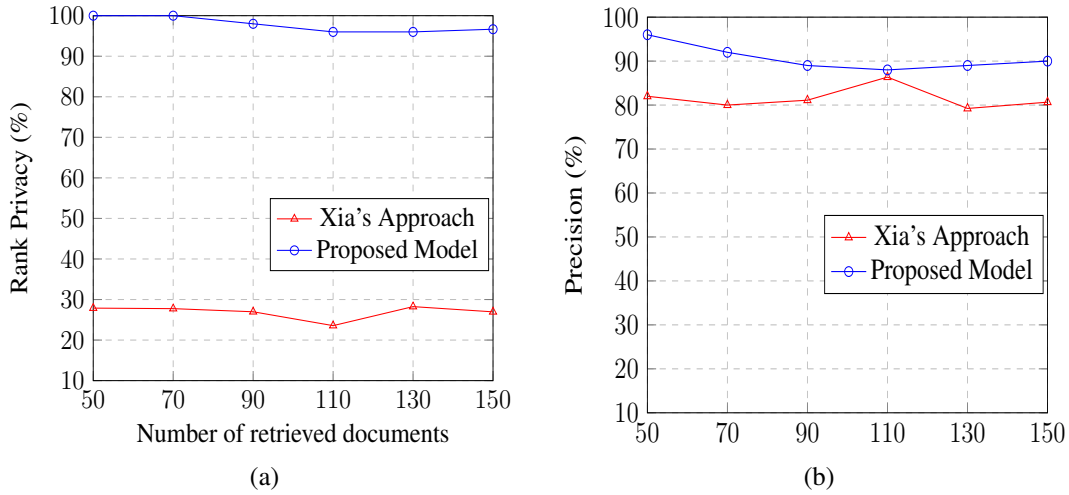


Figure 5.5: (a) Ranking Privacy of top-k retrieved documents. (b) Precision of top-k retrieved documents.

Comparison with the existing approach: In general, the data users randomize the trapdoors by adding random keywords to a list of actual keywords to prevent the leakage of the search pattern. The added random keywords affect the precision but alter rank-order information of documents. Hence, the ranking privacy and precision of the proposed Pseudo-Ranking approach are compared with the Xia et al.'s (Xia et al. 2016) multi-keyword ranked search approach for the randomized trapdoors. In Xia's approach, random values are assigned to random keywords in indexes using a uniform distribution based on the σ value, i.e., the standard deviation of *TF-IDF* values of actual keywords in the index of documents' collection. This σ value is completely dependent on the dataset. For the higher value of σ, random keywords would be assigned far apart

values from the mean of actual values to assure higher rank privacy. For the lower σ, random keywords would be assigned random values that are nearer to the mean to assure higher precision. The rank privacy comparison of randomized trapdoors using the proposed Pseudo-Ranking approach and the Xia's approach with the σ value 0.05 is shown in Figure 5.5.a. It can be observed that rank privacy of the proposed approach is very higher than the Xia's approach because of assigning random values to the random keywords based on the individual document's score. If the document's score is higher than the average score of the documents, then the random keywords would be assigned values that are higher than the TF values of actual keywords of the document. Similarly, if the document's score is lesser than the average score of the documents, then the random keywords would be assigned values that are lower than the TF values of actual keywords of the document. This sort of assigning random values to random keywords based on documents' scores changes the actual ranks of the documents because of which relevant documents become non-relevant ones and non-relevant documents become relevant ones. Thus, the proposed pseudo-ranking approach achieves higher rank-privacy (average of 96%).

The precision comparison of randomized trapdoors using the proposed Pseudo-Ranking approach and the Xia's approach is shown in Figure 5.5.b. The precision of Xia's approach is affected compared to the proposed approach as random values of random keywords involve in determining the actual ranks of the documents for the given trapdoors, which cause false positives. The proposed approach also achieves higher precision (average of 91%) as the impact of random keywords' values is nullified by the intermediate server. Thus, the proposed Pseudo-Ranking approach assures both higher rank privacy and higher precision for any trapdoor without sharing any secret information with the intermediate server.

Search pattern privacy: It can be observed from the results in Figures 5.2.b, and 5.4.b that the randomized trapdoors achieve higher rank privacy than the generic trapdoors. These trapdoors contain random keywords along with actual query keywords due to which different trapdoor would be generated for the same query. In our approach, the search pattern leakage is prevented since the randomized trapdoor includes

more number of random keywords than the number of actual query keywords. The randomized trapdoor in the proposed approach consists of 5 actual query keywords and 95 random keywords. As these random keywords change each time, distinct trapdoors are generated for the same query. Thus, leakage of search pattern is prevented, and resists scale analysis attack successfully. The probability of the cloud server knowing whether earlier trapdoors are related to the same query or not is $5/100$, i.e., 0.05, which is very low. This can be minimized further by adding more random keywords in trapdoors. In our approach, the users are provisioned with more number of random keywords, i.e., (10000 keywords) for including them uniformly while generating trapdoors. Therefore, the probability of repetition of the same random keywords for the same query is low. Search pattern leakage is thus prevented and resists scale analysis attack.

Efficiency

The efficiency of the proposed Pseudo-Ranking approach is analyzed with respect to the index construction and search time of the trapdoor.

- *Index construction:* The required time for constructing encrypted indexes is shown in Figure 5.6. The index construction time includes the time for preprocessing documents, i.e., removal of stopwords, constructing a dictionary, determining TF values and their encryptions. It can be observed that index construction time is linear to the number of documents in the dataset.

- *Search time:* Total searching time of the proposed approach for trapdoors with and without random keywords in trapdoors is shown in Figure 5.7, which includes the processing time of trapdoors at the cloud server and the intermediate server. Processing time of trapdoors at the cloud server for initial scoring of documents and at the intermediate server for re-assignment scores is shown in figures Figure 5.8 and Figure 5.9 respectively.

5.6.3 Usage of the proposed Pseudo-Ranking approach

The proposed Pseudo-Ranking approach is useful in all client-server applications, where processing data is in encrypted form, and the server is not trustworthy, but complete pro-

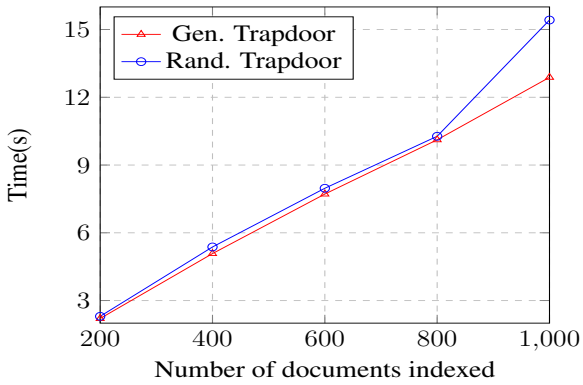


Figure 5.6: Total Searching time of trapdoors for retrieval of Top-10 documents over different sizes of indexes.

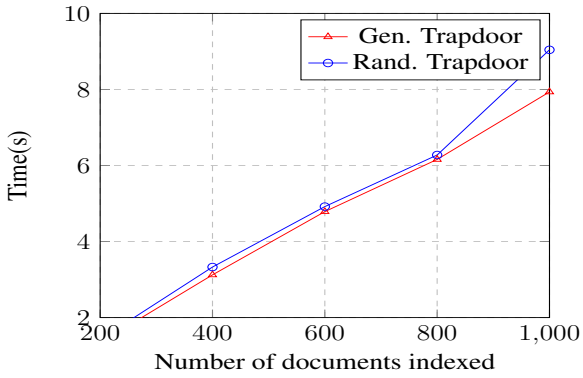


Figure 5.7: Processing time of trapdoors at cloud server for assigning initial scores to documents.

tection of data is expected from the servers. Examples of applications include encrypted search engines and banking applications, where customers' bank account details need to be protected from the administrators of the bank organizations.

The only problem with the proposed approach is that it requires two servers for which the data owner has to pay the price for storing and managing his/her data. This cost is minimal in the present day world due to a highly competitive price model where services are offered at the lower price from various cloud service vendors.

5.7 SUMMARY

The proposed Pseudo-Ranking approach is aimed to prevent the cloud server from the exploitation of rank-ordered information and the search pattern leakage information without affecting the precision. The cloud server does not come to know the actual

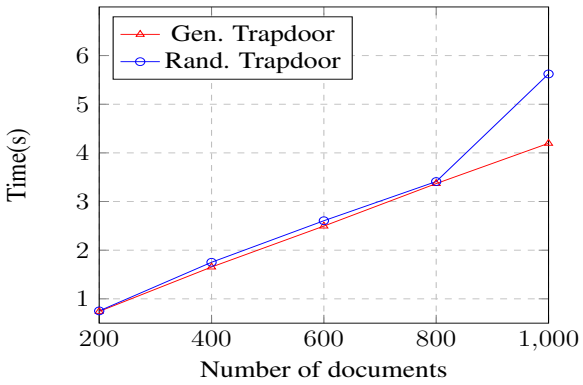


Figure 5.8: Processing time of re-assignment of scores to documents at intermediate server.

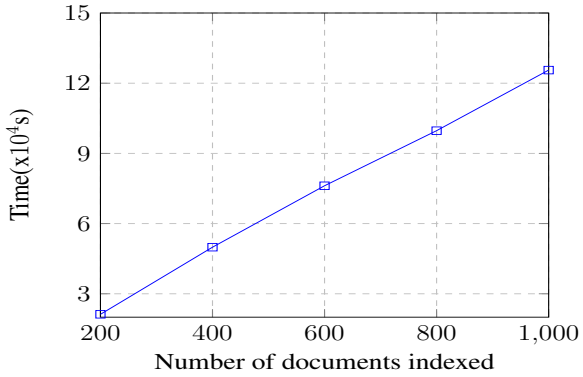


Figure 5.9: Index construction time for a different number of documents with a dictionary, $|W| = 20000$.

ranks of the documents due to the presence of random keywords in trapdoors and the perturbed assignment of random values to the random keywords in indexes. Hence, the proposed approach preserves the privacy of both rank-order information as well as a search pattern. The experimental results confirm that the proposed approach guarantees high precision and higher privacy without affecting precision with the help of the intermediate server (IS). The role of IS in this approach is to nullify the impact of added random keywords in a trapdoor and thereby determines the top-k relevant documents without knowing any secret information and actual query. To guarantee both higher precision and privacy, it is highly recommended to include more number of random keywords than the number of query keywords while generating trapdoors.

CHAPTER 6

A DYNAMIC INDEX STRUCTURE

6.1 INTRODUCTION

Cloud server uses the index structure to retrieve the relevant documents for a given user trapdoor. But in reality, the documents that are stored at the cloud server may get updated frequently by the data owner. In this case, the index structure must also be updated to reflect these changes. In order to retrieve the latest relevant documents for a given user trapdoor, an efficient index structure is required to accommodate the updates. Hence, there is a requirement of an efficient index structure that supports dynamic updates efficiently and also enables the retrieval of top-k relevant documents efficiently for a given trapdoor. Tree-based indexing schemes are helpful in improving the time complexity of performing dynamic updates and retrieval operation. Searchable Encryption (SE) schemes use various tree-based index structures such as keyword red-black (KRB) tree (Kamara and Papamanthou 2013), Privacy Bloom Filter (PB) based tree (Li et al. 2014a), Indistinguishable Binary (IB) tree (Li and Liu 2017), Keyword Balanced Binary (KBB) tree (Xia et al. 2016), and Virtual Binary (VB) tree (Wu and Li 2019) techniques for supporting dynamic updates.

However, the major problem is that the time required for top-k documents retrieval and for performing dynamic updates using these index structures are not optimal because of the large size of the tree structure. Therefore, there is a need for an efficient index structure to address this issue. The objective of this work is to develop an index structure that supports dynamic updates including Insert, Delete and Modify operations

and also helps in retrieving top-k relevant documents efficiently.

There exists some privacy issues while incorporating the updates in index structure. For example, performing a update operation (e.g., an addition of a new keyword) on encrypted documents may leak the position of the added keyword in a dictionary. The existing SE approaches (Fu et al. 2017) leak the potential positions of newly added keywords. Using this information, the cloud server can infer plaintext information of added keywords if index keywords of the documents are stored in lexicographical order. Hence, there exists a need for a privacy preserving searchable index structure that leaks no information regarding the positions of newly added keywords.

Need for an efficient index structure: The existing tree-based index structures such as KRB tree, PB tree, IBF tree and VB tree-based approaches support only boolean search but not ranked search. The KBB tree-based index supports ranked search, but it visits many internal nodes while performing dynamic updates and processing trapdoors. All these index structures are constructed either in a top-down or bottom-up fashion and many internal nodes store information required for guiding the search not the actual documents. Because of these, the size of the tree is higher in terms of height and breadth and also involves visiting more number of nodes for retrieving top-k documents for a given trapdoor. This greatly impacts the efficiency of search and update time. Therefore, it is required to minimize the size of the tree as well traversing time involved. Hence, a max-heap based binary tree index structure is developed to address this issue.

Need for a secure keyword dictionary expansion approach: Addition of new keywords to the already existing dictionary requires the re-creation of unencrypted index followed by re-encryption of the entire index. This incurs a huge cost because of re-encrypting the whole index. In order to reduce the re-encryption time, we use partitioned matrices concept (Li et al. 2014b). When new keywords are added to the existing index, the existing SE schemes (Fu et al. 2017; Li et al. 2014b) leak the newly added keywords' positions to the cloud server. This privacy breach leaks the information regarding the newly added keywords to the cloud server. To minimize this leakage, a secure keyword dictionary expansion approach is required such that the cloud

server should not come to know the locations of the newly added keywords when new keywords are added.

This work contributes to the existing literature in the following ways:

- Proposing an efficient Max-heap based Binary Tree index structure for supporting dynamic updates.
- Proposing a Privacy Preserving Keyword Dictionary Expansion approach for adding new keywords securely to the existing dictionary.
- Performing security analysis of the proposed approaches.

The rest of the chapter is explained as follows: Section 6.2 presents the information on preliminaries. Section 6.3 presents the problem statement and the proposed approach is presented in Section 6.4. Section 6.5 presents the experimental study and analysis of the proposed approach. The summary of this chapter is presented in Section 6.6.

6.2 PRELIMINARIES

The proposed work includes the below notations that are used throughout this chapter.

Vector Space Model: The proposed index structure uses vector space model in which both documents and queries are represented as vectors. Since the proposed index structure supports dynamic updates, any update on a single document may change the IDF values of many documents if TF-IDF values are stored as the keywords' relevance scores in existing indexes. To avoid this, both TF and IDF values of keywords are separated and stored in the document and query vectors respectively as shown below:

Document vector: Each document is represented as a sequence of TF values of each keyword of the dictionary present in the document. The vector information would be used as an index information of the document.

Query vector: Each query is represented as a sequence of IDF values of dictionary's keywords present in the query. The size of both the query and document vectors should be the same.

Notation	Description
D	— The input document collection, denoted as a set of n documents $D = (D_1, D_2, ..., D_n)$.
W	— The distinct keywords extracted from D, denoted as set of m keywords $W = (kw_1, kw_2, ..., kw_m)$.
n	— The total number of documents of the input dataset D.
m	— The total number of keywords of the dictionary W.
TF_{kw_j,D_i}	— The Term Frequency of keyword kw_j in document D_i.
$IDF_{kw_j,W}$	— The Inverse Document Frequency of keyword kw_j of dictionary W_i.
I	— The plaintext index generated from documents D.
I'	— Encrypted index to be outsourced, which is constructed from I.
C	— The encrypted document collection to be outsourced, denoted as the set of ciphertext documents $C = (C_1, C_2, C_3, ..., C_n)$. Each C_i corresponds to D_i in D.
$TFVector$	— The vector of TF values of each keyword of the dictionary present in D_i.
$MaxVector$	— A $MaxVector$ of parent node consists of maximum TF values from the left and right child nodes' TF values.
TD	— The trapdoor, which is encrypted Query vector consists of encrypted IDF values of query keywords
$QueueNodesList$	— It contains a list of nodes added while constructing max-heap tree in a level order form.
$MaxVectorList$	— It contains of list of identities of the nodes and the corresponding encrypted $MaxVector$s.

Secure inner product similarity operation: A secure k-nearest neighbor algorithm is used to compute the relevance score of the document without conveying plaintext information of document and query vectors to the cloud server. It involves the following processes to determine the relevance scores of the document securely for a given trapdoor.

- *Key generation:* A secret key SK is generated that consists of a random bit vector and two symmetric invertible matrices, i.e., SK = {S, M_1, M_2}, where the size of S should be same as the size of dictionary (e.g., m keywords) and the size of both matrices M_1 and M_2 should also be (m x m).

- *Encrypting document vector:* It involves splitting the document vector into two parts based on a random vector S and then encrypt each part by multiplying with the invertible matrices M_1 and M_2, respectively. The index of encrypted document vector would then be generated as $\mathtt{I'}_{\mathtt{D}_i}$=$\{\mathtt{I'}_{\mathtt{D}_1}, \mathtt{I''}_{\mathtt{D}_2}\}$ = $\{\mathtt{D'}_1\mathtt{M_1}^{\mathtt{T}}, \mathtt{D''}_2\mathtt{M_2}^{\mathtt{T}}\}$.

- *Encrypting Query Vector:* It also involves splitting the query vector into two parts based on the random bit vector S and then encrypt each part by multiplying with the inverses of the invertible matrices M_1^{-1} and M_2^{-1}, respectively. For a given query $\mathtt{Q}$, the trapdoor TD, i.e., the encrypted query vector would then be generated as $\{\mathtt{TD'}, \mathtt{TD''}\}$ = $\{\mathtt{Q'M_1}^{-1}, \mathtt{Q''M_2}^{-1}\}$

- *Determining relevance score:* The description of determining the relevance score of a document D using the encrypted document vector $\mathtt{I'}_{\mathtt{D}_i}$ and the trapdoor TD is provided below.

$$
\begin{aligned}
RScore(\mathtt{I}_{\mathtt{D}_i}\mathtt{.TD}) &= \mathtt{I'}_{\mathtt{D}_1}\mathtt{.TD'} + \mathtt{I''}_{\mathtt{D}_2}\mathtt{.TD''} \\
&= (\mathtt{M_1}^{\mathtt{T}}\mathtt{.D'}_1).(\mathtt{M_1}^{-1}\mathtt{.Q'}) + (\mathtt{M_2}^{\mathtt{T}}\mathtt{.D''}_2).(\mathtt{M_2}^{-1}\mathtt{.Q''}) \\
&= \mathtt{D'}_1\mathtt{.Q'} + \mathtt{D''}_2\mathtt{.Q''} \\
&= \mathtt{D.Q} \\
&= \mathtt{RScore(D,Q)}
\end{aligned}
$$

  For any given document vector D_i of TF values and a query vector Q of IDF values, the relevance score (rank) of a document can be determined as

$$RScore(D_i, Q) = \sum_{1 \le i \le m} TF_{D_i} * IDF_{Q_i} \tag{6.1}$$

  where, m indicates the length of the dictionary.

6.3 PROBLEM STATEMENT

An efficient index structure is required to return the latest top-k relevant documents efficiently for a given trapdoor. To return the latest relevant documents, the index structure is required to accommodate the dynamic updates, i.e., Insert, Delete and Update operations efficiently and securely over encrypted documents.

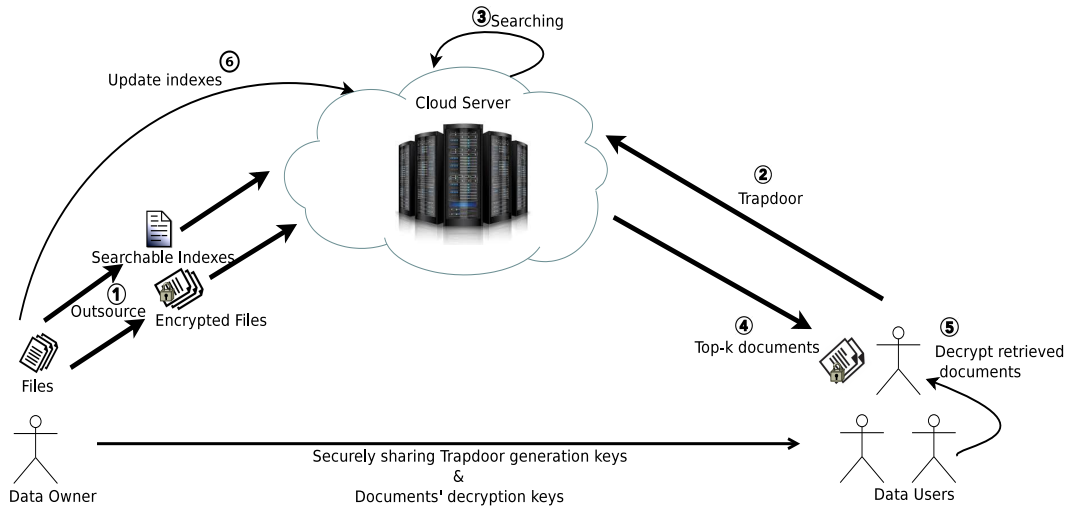


Figure 6.1: System architecture for performing dynamic updates.

System Architecture: This work adopts the system architecture shown in Figure 6.1 to test the efficiency of the proposed index structure for retrieving top-k relevant documents. In this approach, the data owner uploads the encrypted index onto the cloud server and keeps the plaintext version of the index with him for performing dynamic updates. The data owner can later perform the dynamic updates efficiently in an already existing index. The data owner then shares the secret keys and also the IDF values of the keywords of the dictionary with the data users. The data users then sends the trapdoor, which consists of encrypted IDF values of query keywords to the cloud server. The cloud server then retrieves the latest top-k relevant documents efficiently using the above index and sends them to the users, who decrypt them with the corresponding secret keys.

6.4 MAX-HEAP BASED BINARY TREE INDEX

This section explains the proposed Max-heap based Binary Tree index structure. This structure uses the properties of both Max-heap based binary tree and Keyword Balanced Binary (KBB) tree. The KBB tree helps in searching for relevant documents for a given trapdoor using greedy depth first search approach, while max heap based binary tree helps in constructing the tree index with minimum height and breadth. Three major steps such as index construction, retrieving top-k relevant documents and performing dynamic updates, are involved in retrieving top-k relevant documents efficiently while

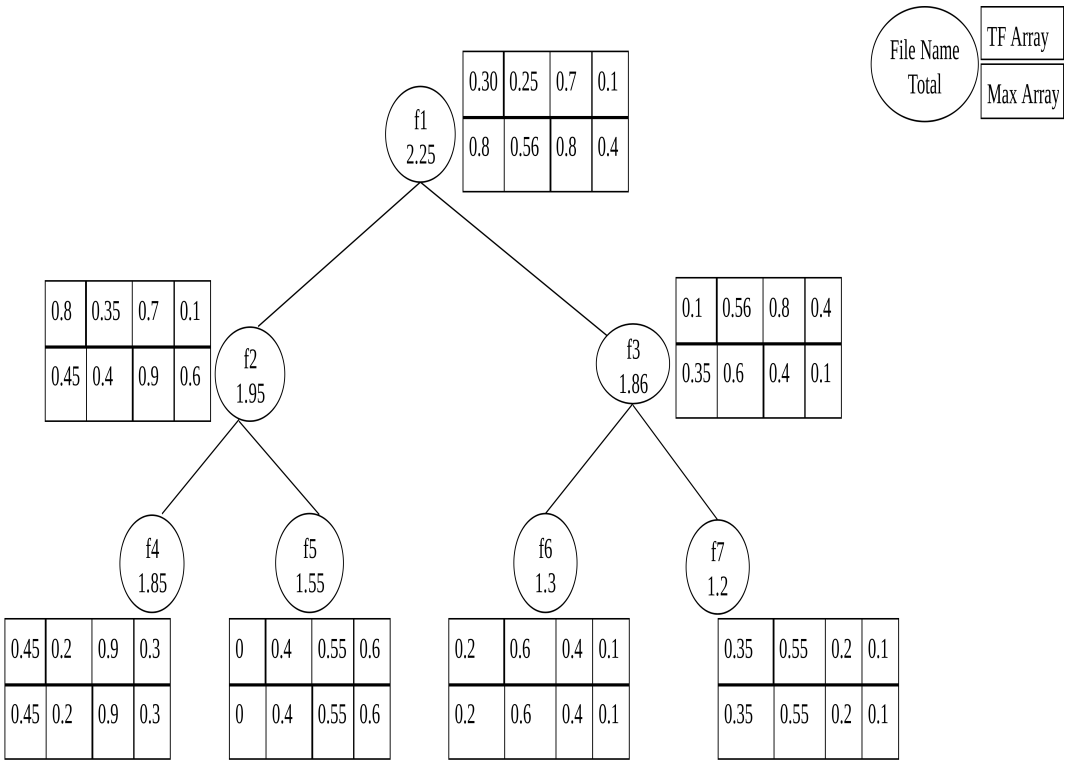


Figure 6.2: Max-heap based Searchable Tree Index.

supporting dynamic updates using the max-heap based binary tree index structure. Each of the steps are explained in the subsequent subsections.

6.4.1 Index construction

The data owner is required to generate an index for all the documents of the dataset D. First, plaintext tree index is created and then the content present in each node of the tree is encrypted. The procedure for constructing the plaintext tree index is explained as follows:

i) Generating plaintext tree index: A separate node u is created for each document D_i of the dataset D, which is represented in the following form.

$$u = <id(u), id(D_i), TFVector, MaxVector, D_iScore, P_l, P_r, P_p>$$

where, $id(u)$ is the identity of the node that is useful while performing dynamic updates, $id(D_i)$, the identity of the document D_i, $TFVector$ the vector of TF values of each keyword of the dictionary present in D_i, $MaxVector$ the vector of the maximum TF values of left and right child nodes' TF values, D_iScore the score of the document D_i

that is determined by summing all the values of $TFVector$, the pointers P_l points to left child node, P_r points to right child, P_P points to a parent node.

Initially the $MaxVector$ is same as $TFVector$ but gets updated as the number of nodes in tree index get added. After the index construction, $MaxVector$ of leaf node is same as $TFVector$, the P_P of root node points to $NULL$ value and the pointers P_l and P_l are set to $NULL$ for all the leaf nodes. Once the nodes for all the documents are created, the construction of the max-heap based binary tree index is initiated based on the total scores of the documents. The first node is considered as a root node of the tree initially, then it picks the next node and is inserted at a next available location in a level order fashion starting from left to right. After the insertion of a new node, heapify property (total document score of a parent node must be higher or equal to their left and right child nodes' scores.) is maintained. As new nodes are incorporated into an existing tree, the heapify property may be affected, i.e., the total score of the parent nodes would not be greater than or equal to the total scores of the child nodes. Therefore, it is required to call the heapify procedure at the newly inserted node to maintain the heapify property. Thus, the plaintext version of the proposed Max-heap based binary tree index is created. The plaintext version of the proposed max-heap based KBB tree indexing for seven documents is shown in Figure 6.2. The complete procedure for constructing the proposed Max-heap based KBB tree index is given in Algorithm 5. This algorithm takes the input the documents of the given dataset D and returns the max-heap index tree T. The procedure for heapifying tree starting from the newly inserted node till the root node in an upward direction is provided in Algorithm 6.

ii) Encrypting the content of the plaintext tree index: After the generation of the plaintext tree index, the data owner then encrypts the content in each node of the tree index, i.e., $TFVector$, $MaxVector$, D_iScore. This information in each node of the proposed tree index is encrypted with the 3-step procedure given below.

- *Setup:* It involves generating secret key that is required for encrypting the content of the nodes. An m-bit secret random binary vector S and two symmetric random

Algorithm 5: Max-heap based Binary Tree Index

Input: D, the input document collection, denoted as a set of n documents $D = (D_1, D_2, ..., D_n)$.

Output: The max-heap based binary tree index T.

for *each document* D_i *in* D **do**
  create a node u with the pointers set to $u.C_{left}$=$u.C_{right}$=$u.P$=$Null$, $u.Id$=$Generate\_Id(D_i)$, $TFVector(D_i)$=TF_{D_i,w_j}, and $MaxVector(D_i)$=TF_{D_i,w_j}, $Dscore$=$sum(TF_{D_i,w_j})$ for j=1,2,3,...,m of dictionary W.
  Insert node u to $TreeNodesList$
Delete the first node from $TreeNodesList$, then make it as a $root$ and append it to the $QueueNodesList$.
while *number of nodes in* $TreeNodesList$ *is not* $NULL$ **do**
  Delete a node from $QueueNodesList$, which will act as a current $parent$ node.
  if *Number of nodes in* $TreeNodesList \geq 2$ **then**
    Extract the first two nodes of $TreeNodesList$ and add them as $left$ and $right$ child nodes to the current $parent$ node, i.e., $parent.left$=$parent.right$=$parent$ and also assign $parent$ to the $left$ and $right$ child nodes, i.e., $left.parent = right.parent = P$.
    Call $Heapify\_Upward()$ procedure on $parent$ node based on D_iscore.
  else
    Extract a node from $TreeNodesList$ and add it as either $left$ or $right$ child to the $parent$.
    Append it to the $QueueNodesList$
    Call $Heapify\_Upward()$ on $parent$ node based on D_iscore.
return $root$

invertible matrices M_1 and M_2 of size m $\times$ m are created, where m is the cardinality of the dictionary. Secret key SK is set as SK = {S, M_1^T and M_2^T}.

- *Encrypting the vectors at node u:* The node u of the tree contains two vectors $TFVector$ and $MaxVector$ that are having the same size m as the size of the dictionary. Each of the vectors are encrypted as follows:

  The $TFVector$ at node u in the tree index is split into two random vectors D_{uTF}', D_{uTF}'', which are generated based on the secret vector S, i.e., if S[i] = 0 then D'_{uTF}[i] = D''_{uTF}[i] = D_{uTF}[i]. Otherwise, if S[i] = 1 then D'_{uTF}[i] = b, where b is any random number such that b $\in$ (0, D_{uTF}[i]), D''_{uTF}[i] = (D_{uTF}[i] - D'_{uTF}[i]). The split vectors D'_{uTF} and D''_{uTF} are then encrypted by

Algorithm 6: Heapifying_Upward(parent)

Input: parent, Tree T
Output: Heapfied Tree T

if $parent$==*Null* **then**
  return
else
  if *parent has both the childs* **then**
    if *leftchild.*$D_i score >$ *rightchild.*$D_i score$
    and leftchild.$D_i score >$ *parent.*$D_i score$ **then**
      Swap the content, i.e., ($TFVector(D_i)$,$D_i score$, and $id(D_i)$) of left child with $parent$ node's content and update the parent's $MaxVector[j]$= Max(leftchild.$TFVector[j]$,rightchild.$TFVector[j]$), for all j=1,2,3,...,m of dictionary W.
      $Heapifying\_Upward$(parent of parent)
    else
      if *rightchild.*$D_i score >$*leftchild.*$D_i score$ *and*
      rightchild.$D_i score >$ *parent.*$D_i score$ **then**
        Swap the content, i.e., ($TFVector(D_i)$,$D_i score$, and $id(D_i)$) of right child with $parent$ node's content and update the parent's $MaxVector[j]$= Max(leftchild.$TFVector[j]$,rightchild.$TFVector[j]$), for all j=1,2,3,...,m of dictionary W.
        $Heapifying\_Upward$(parent of parent)
  else
    Only one child exists, i.e., parent has either left or right child.
    if *child's* $D_i score >$ *parent.*$D_i score$ **then**
      Swap the content, i.e., ($TFVector(D_i)$,$D_i score$, and $id(D_i)$) of child node with the $parent$ node's content and update the parent's $MaxVector[j]$= child.$TFVector$
      $Heapifying\_Upward$(parent of parent)
return *Heapfied Tree* T

multiplying $\mathtt{D'}_{\mathtt{uTF}}$ and $\mathtt{D''}_{\mathtt{uTF}}$ with the transposes of random invertible matrices $\mathtt{M_1}^{\mathtt{T}}$ and $\mathtt{M_2}^{\mathtt{T}}$. The encrypted TF vectors after splitting the TFVector into two parts at index node u would then be $\mathtt{I}_{\mathtt{uTF}}$=$\{\mathtt{I'}_{\mathtt{uTF}}, \mathtt{I''}_{\mathtt{uTF}}\}$ = $\{\mathtt{D'}_{\mathtt{uTF}}\mathtt{M_1}^{\mathtt{T}}, \mathtt{D''}_{\mathtt{uTF}}\mathtt{M_2}^{\mathtt{T}}\}$.

Similarly, the $MaxVector$ at node u is encrypted in the same way as $TFVector$ is encrypted. The $MaxVector$ at node u in the tree index is split into two random

vectors $\mathtt{D}_{\mathtt{uMax}}'$, $\mathtt{D}_{\mathtt{uMax}}''$, which are generated based on the secret vector S, i.e., if S[i] = 0 then $\mathtt{D}'_{\mathtt{uMax}}\mathtt{[i]} = \mathtt{D}''_{\mathtt{uMax}}\mathtt{[i]} = \mathtt{D}_{\mathtt{uMax}}\mathtt{[i]}$. Otherwise, if S[i] = 1 then $\mathtt{D}'_{\mathtt{uMax}}\mathtt{[i]} = \mathtt{b}$, where b is any random number such that $\mathtt{b} \in (0, \mathtt{D}_{\mathtt{uMax}}\mathtt{[i]})$, $\mathtt{D}''_{\mathtt{uMax}}\mathtt{[i]} = (\mathtt{D}_{\mathtt{uMax}}\mathtt{[i]} - \mathtt{D}'_{\mathtt{uMax}}\mathtt{[i]})$. The split vectors $\mathtt{D}'_{\mathtt{uMax}}$ and $\mathtt{D}''_{\mathtt{uMax}}$ are encrypted by multiplying $\mathtt{D}'_{\mathtt{uMax}}$ and $\mathtt{D}''_{\mathtt{uMax}}$ with the transposes of the random invertible matrices $\mathtt{M}_1{}^{\mathtt{T}}$ and $\mathtt{M}_2{}^{\mathtt{T}}$. The encrypted Max vectors after splitting the $MaxVector$ into two parts at index node u would then be $\{\mathtt{I}'_{\mathtt{uMax}}, \mathtt{I}''_{\mathtt{uMax}}\} = \{\mathtt{D}'_{\mathtt{uMax}}\mathtt{M}_1{}^{\mathtt{T}}, \mathtt{D}''_{\mathtt{uMax}}\mathtt{M}_2{}^{\mathtt{T}}\}$.

- *Encrypting the score of the document at the node u:* The score of the document D_i is encrypted using the proposed Enhanced One-to-Many OPE scheme. This scheme takes the input id(D_i), $score$ and NULL value as the input and returns ciphertext score. As this scheme preserves the order of the plaintext scores, the cloud server can heapify the tree index based on the encrypted scores.

After encrypting the content in each node of the tree index, the data owner then encrypts all the documents of the dataset using the AES algorithm. The data owner then uploads both the tree index in encrypted form and the encrypted documents C onto the cloud server.

6.4.2 Retrieval of Top-k relevant documents

Retrieving top-k relevant documents for a given trapdoor includes two processes:
i) trapdoor generation and ii) searching. Each of them are explained below:

i) Trapdoor Generation: The data user first represents his query in a query vector Q, which consists of a sequence of IDF values of query keywords. The size of the query vector should be same as the size of the dictionary in order to determine the score of the documents correctly using the secure inner product operation. Therefore, the query vector may include some random values along with the IDF values of actual query keywords. The query vector is then encrypted in the same way as the TF and MAX vectors are encrypted. The query vector, i.e., the IDF vector, is split into two random vectors $\mathtt{Q}'$ and $\mathtt{Q}''$, which are generated based on the same secret vector S, i.e., if S[i]

= 1 then $\mathtt{Q'[i]} = \mathtt{Q''[i]} = \mathtt{Q[i]}$. Otherwise, if $\mathtt{S[i]} = 0$ then $\mathtt{Q'[i]} = \mathtt{b}$, where $\mathtt{b}$ is any random number such that $\mathtt{b} \in (0, \mathtt{Q[i]})$, $\mathtt{Q''[i]} = (\mathtt{Q[i]} - \mathtt{Q'[i]})$. The split query vectors $\mathtt{Q'}$ and $\mathtt{Q''}$ are encrypted by multiplying them with the inverses of random invertible matrices $\mathtt{M_1}^{-1}$ and $\mathtt{M_2}^{-1}$. The trapdoor, i.e., encrypted query vectors after splitting the query vector into two parts would then be, $\mathtt{TD}=\{\mathtt{TD'}, \mathtt{TD''}\} = \{\mathtt{Q'M_1}^{-1}, \mathtt{Q''M_2}^{-1}\}$. The data user then sends the trapdoor to the cloud server.

ii) Searching: Upon receiving the given trapdoor, the cloud server processes it to determine relevant documents efficiently using the proposed max-heap based index structure. The search procedure for determining top-k relevant documents for a given trapdoor, is given in Algorithm 7. The search procedure starts from the root node to leaf nodes in downward direction until the top-k relevant documents are determined. It uses the greedy depth first search approach to determine the top-k relevant documents efficiently without visiting all the nodes of the tree index. While searching at each node u of the tree index structure, two scores are determined, i.e., relevance score of the document D and the $maxscore$ of a node u. The determination of the two scores at node u based on its vectors, i.e., encrypted $TFVector$ and $MaxVector$ is given below.

The relevance score of a document at index node I_u is obtained by applying secure inner product similarity measure between the node's encrypted $TFvector$ and trapdoor vector TD. This relevance score is equal to the plaintext score that is obtained by applying inner product operation on the plaintext TFvector and the query vector $\mathtt{Q}$, which is shown below:

$$\begin{aligned}
\mathtt{I_{uTF}.TD} &= \mathtt{I'_{uTF}.TD'} + \mathtt{I''_{uTF}.TD''} \\
&= (\mathtt{M_1}^{\mathtt{T}}.\mathtt{D'_{uTF}}).(\mathtt{M_1}^{-1}.\mathtt{Q'}) + (\mathtt{M_2}^{\mathtt{T}}.\mathtt{D''_{uTF}}).(\mathtt{M_2}^{-1}.\mathtt{Q''}) \\
&= \mathtt{D'_{uTF}}.\mathtt{Q'} + \mathtt{D''_{uTF}}.\mathtt{Q''} \\
&= \mathtt{D.Q} \\
&= \mathtt{RScore(D,Q)}.
\end{aligned}$$

Similarly, the $maxscore$ at node u can be obtained by applying secure inner product

similarity measure between trapdoor vector TD and encrypted $Maxvector$.

$$\begin{aligned}
\mathtt{I}_{\mathtt{uMax}}.\mathtt{TD} &= \mathtt{I'}_{\mathtt{uMax}}.\mathtt{TD'} + \mathtt{I''}_{\mathtt{uMax}}.\mathtt{TD''} \\
&= (\mathtt{M_1}^{\mathtt{T}}.\mathtt{D'}_{\mathtt{uMax}}).(\mathtt{M_1}^{-1}.\mathtt{Q'}) + (\mathtt{M_2}^{\mathtt{T}}.\mathtt{D''}_{\mathtt{uMax}}).(\mathtt{M_2}^{-1}.\mathtt{Q''}) \\
&= \mathtt{D'}_{\mathtt{uMax}}.\mathtt{Q'} + \mathtt{D''}_{\mathtt{uMax}}.\mathtt{Q''} \\
&= \mathtt{D}_{uMax}.\mathtt{Q} \\
&= \mathtt{RScore}(\mathtt{D}_{uMax},\mathtt{Q}).
\end{aligned}$$

After both the scores are determined at node u, the score of the document along with its identity $id(D_i)$ is added to the top-k list only if it is greater than the k^{th} score of the document. The k^{th} score in the top-k list is initially set to be 0. The searching continues either in the left subtree or in the right subtree based on the maxscore of the child nodes. At any node u, if its document's relevance score is greater than the k^{th} score in the top-k list, then it replaces the k^{th} score element in the top-k list with a determined relevance score. It continues searching recursively till the top-k relevant documents are determined. It stops searching when the $maxscore$ determined node u is not greater than the k^{th} score of the document. At this node, it checks whether the document's score of this node is greater than the k^{th} score; if so, replace it with this document; otherwise, it stops searching from this node. The cloud server then finally returns the top-k relevant documents to the users.

6.4.3 Dynamic Updates

Besides retrieving the top-k relevant documents efficiently, the proposed index structure also supports Insert, Delete and Modify operations efficiently over the encrypted documents. The description of incorporating the updated information in an already existing max-heap based KBB tree index is explained below.

i) Insertion of new documents: When a set of new documents are to be inserted, it is required to update the existing tree index by accommodating the updated index information of newly added documents. The data owner first creates a separate node containing plaintext information for each of the new documents as explained in Section 6.4.1. Each node contains $id(D_i), TFVector, MaxVector, D_iScore, P_l, P_r, P_p$, where $id(D_i)$ is the identity of document (document name), $TFVector$ is a vector which

Algorithm 7: $GDFS(MaxTreeNode\ node)$

Input: root
Output: Top_k_List

```
1  if root==Null then
2      return
3  else
4      Determine the relevance score rscore at root node using TFVector(D_i)
         and then add both the Dscore and its document identity id(D_i) to the
         Top_k_List.
5      Also, determine maxscore using MaxVector of root node.
6      if maxscore > kthscore then
7          Determine left child's maxscore using MaxVector of left child and
             also determine right child's maxscore using MaxVector of right
             child
8          Consider the Maximum score of child, then determine D_iscore using
             its TFVector
9          if number of elements in Top_k_List ≤ K then
10             Add its identity id(D_i) and D_iscore to the Top_k_List.
11         else
12             Replace the element at k^th position in the Top_k_List with the
                 child's D_iscore and its identity id(D_i).
13             Sort the elements of Top_k_List in descending order using D_iscore.
14         GDFS(maxScore of child)
15         GDFS(Other child node)
16     else
17         return
18 return Top_k_List
```

consists of TF values of each keyword of the document, $MaxVector$ is determined with the help of plaintext version of max-heap based binary tree and it consists of the maximum TF values that are determined from the maximum of left and right child nodes' TF values, D_iScore is a score of the document that is determined by adding all the values of $TFVector$, the pointers P_l, P_r, and P_P are set to $Null$. Once the plaintext information of $TFVector$, $MaxVector$ and D_iScore are ready for each node of the new documents, the data owner then encrypts the content present in each node, i.e., $TFVector$, $MaxVector$ by using two randomized invertible matrices, and the D_iScore by using Enhanced One-to-Many OPE scheme. Then, data owner sends the nodes containing the encrypted information related to newly inserted documents to the

cloud server. The data owner also encrypts the newly added documents using the AES algorithm and sends them onto the cloud server.

Algorithm 8: Heapifying Downward

Input: Replaced Node RN, Tree T
Output: Heapified tree T

```
1  if RN==Null then
2      return
3  else
4      if RN has both childes then
5          if leftchild.D_iscore > rightchild.D_iscore and
6          leftchild.D_iscore > RN.D_iscore then
7              Swap the content of RN node, i.e., id(D_i),TFArray,D_iscore with
                the content of leftchildnode.
8              Then update the MaxVector of replaced node, i.e.,
                RN.MaxVector=
                Max(leftchildnode.TFVector[j],rightchildnode.TFVector[j]),
                for all j=1,2,3,...,m of dictionary W.
9              Heapify_Downward(RN.left)
10         else
11             if rightchild.D_iscore > leftchild.D_iscore and
12             rightchildnode.D_iscore > RN.D_iscore then
13                 Swap the content of RN node, i.e., id(D_i),TFArray,D_iscore
                    with the content of rightchildnode.
14                 Then update the MaxVector of RN, i.e., RN.MaxVector=
                    Max(leftchildnode.TFVector[j],rightchildnode.TFVector[j])
                    for all j=1,2,3,...,m of dictionary W.
15                 Heapify_Downward(RN.right)
16     else
17         RN has only one child, i.e., either or left or right child.
18         if childnode.D_iscore > RN.D_iscore then
19             Swap the content of RN node, i.e., id(D_i),TFArray,D_iscore with
                the content of childnode.
20             Then update the MaxVector of RN, i.e.,
                RN.MaxVector=childnode.TFVector
21             Heapify_Downward(RN.childnode)
22 return
```

The cloud server then stores each of the received nodes at an appropriate positions with the help of QueueNodeslist given in an Algorithm 5, which always point to loca-

tion where the new nodes are to be incorporated. After the accommodation of each new node, the heapify property may get affected and therefore, it is required maintain the heapify property based on $D_i score$ of the new inserted node. The procedure for heapifying the plaintext max-heap based tree index starting from the parent of the newly inserted node till the root node of the tree is provided in Algorithm 6.

ii) Deletion of existing documents: To delete a set of documents, the data owner is required to send the list of document identities to the cloud server that are to be deleted from an existing tree index. The cloud server then deletes the nodes that contain these document identities from an existing tree index and then deletes corresponding encrypted documents. In max-heap based tree index, deletion of intended node is to be replaced by the last node of the existing tree index. Therefore, deletion of each node in a max-heap tree index affects the max heap property after the replacement at the deleted node. Hence, it is required to heapify the updated tree index at the deleted node till the leaf node in a downward direction. The procedure for heapifying the plaintext max-heap based tree index after deleting and replacing it with a last node of the tree is provided in Algorithm 8.

iii) Modification of existing documents: While performing modify operation on any existing document, it is highly desirable to incorporate only the updated part of the document in an already existing indexes instead of re-generating the entire index of this document. For example, when a document is modified, i.e., a set of new keywords are added to a document, then it is required to update only this part in an already existing index without regenerating the entire index. The modification of a document required to include a set of newly added keywords into an already existing dictionary securely and call the heapify proedure. But, there exists a privacy problem when any new keywords are added to the existing dictionary. As the keywords in dictionary are stored in lexicographical order, any leakage of the locations of newly added keywords allows the cloud server to infer plaintext information. Hence, it is required to prevent the cloud server from knowing the actual locations in order to prevent it from inferring added keywords. To address this issue, a privacy preserving keyword dictionary expansion approach is proposed, which is explained in the below subsection.

6.4.3.1 Secure Keyword Dictionary Expansion Approach

It is aimed to prevent the leakage of the actual location of the newly added keyword to the cloud server. The proposed privacy preserving keyword dictionary expansion approach is shown in Figure 6.3 for preserving the privacy of the location of newly added keyword. It shows that there exists already some original keywords (m+m'), where m is length of the dictionary and m' is number of dummy keywords added to ensure that the size of each document's index is same. The picture depicts the dictionary expansion when a new genuine keyword is added along with $y-1$ dummy keywords to the list of already existing keywords (m+m'). In this approach, a set of $(y-1)$ dummy keywords are added along with the genuine keyword, where y is a positive integer which is varied randomly every time we add a new keyword. Even the position of the genuine keyword is varied during every addition. Let $x < (y-1)$ be the number of dummy keywords ahead of the genuine keyword. This adds randomness in the newly added keywords to the dictionary that greatly reduces the probability of finding the position of the newly added genuine keyword.

For the newly added keyword, the actual TF value would be assigned in the corresponding document vector, but it is required to assign random values to the added random keywords in that vector. To store the TF value of actual keyword and the random values of random keywords in an already existing document vector, it is required to extend the document vector D_u corresponding to node u in the proposed tree index from $m+m'$ to $(m + y)$ as shown in the Figure 6.3. After extending the document vector, it is required to encrypt only the index information of the newly added keywords instead of re-encrypting the whole document's vector.

Encrypting the extended document vector of newly added keywords without re-encryption: In general, addition of new keywords to the already existing dictionary requires the re-creation of unencrypted index tree followed by the re-encryption of the entire content in each node of tree. This incurs a huge cost because of the generation of new invertible random matrices of new size. In order to reduce the re-encryption time, we use partitioned matrices concept proposed by (Li et al. 2014b). Partitioned

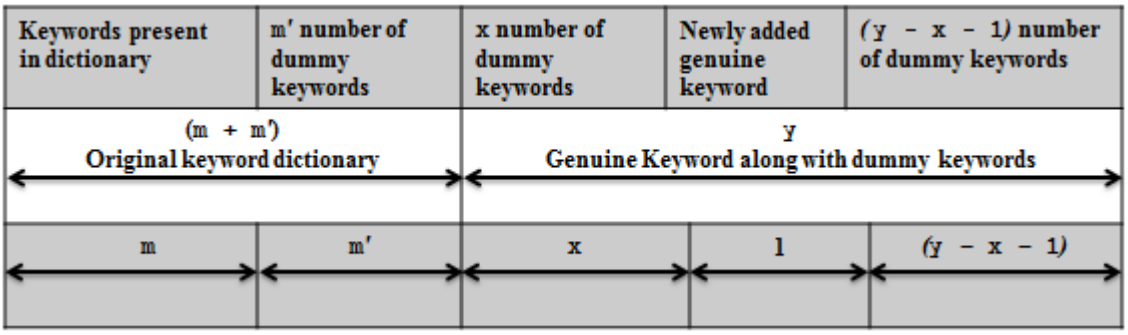


Figure 6.3: Keyword Dictionary Expansion

matrices help in incorporating the new values of added keywords in an already existing document's vector without re-encrypting the whole document vector.

To encrypt only the extended vector, the secret key $Sk = \{S, M_1, M_2\}$ is required to be extended, where the S is a inital random bit vector, matrices M_1 and M_2 correspond to the initial invertible matrices that are suitable for encrypting the documents' vectors before modification, but not after the addition of new keywords. Hence, the random bit vector and the matrices are extended as shown below so that the resulting matrices can be used to encrypt only the extended vector information of newly added keywords. S=SS_{new}, where S_{new} is the extended random bit vector, whose length is equal to the new of newly added keywords (y), and the extended matrices are given below.

$$\mathtt{M'_1}^{\mathtt{T}} = \begin{pmatrix} \mathtt{M_1}^{\mathtt{T}} & 0 \\ 0 & \mathtt{M_y}^{\mathtt{T}} \end{pmatrix} \quad \mathtt{M'_2}^{\mathtt{T}} = \begin{pmatrix} \mathtt{M_2}^{\mathtt{T}} & 0 \\ 0 & \mathtt{M'_y}^{\mathtt{T}} \end{pmatrix}$$

The transpose of newly created matrices $\mathtt{M'_1}$ and $\mathtt{M'_2}$ can be used for encrypting the new document's vectors and the matrices $\mathtt{M_y}^{\mathtt{T}}$, $\mathtt{M'_y}^{\mathtt{T}}$ can be used for encrypting the extended document vector. After encrypting only the extended document vector, incorporate the same in the corresponding node u of the proposed indexing tree.

Heapifying after the modification of documents: The modify operation on any of the existing documents may change their scores after incorporating the extended document vector at a node u of the max-heap tree. Hence, it is required to keep maintain the heapify property of the proposed max-heap based KBB tree based on the updated document score$D_i score$. The procedure for heapifying the tree starting at a node u after

incorporating the index information of modified document is provided in Algorithm 6. This heapify procedure is called recursively in upward direction till the root of the node to maintain the heapify property.

Retrieving relevant documents after modifying the documents: The length of the document's vector and query vector should be same for determining the relevance score of the document correctly. With the incorporation of vector information of newly added keywords, the length of document vector is extended. Hence, the query vector also should be extended in order to perform the vector product operation correctly. The extended query vector can contain any IDF value of other query keywords or random values of random keywords. Then, the inverses of the invertible matrices can be used to encrypt the new query vectors, which are given below.

$$\mathrm{M'_1}^{-1} = \begin{pmatrix} \mathrm{M_1}^{-1} & 0 \\ 0 & \mathrm{M_y}^{-1} \end{pmatrix} \quad \mathrm{M'_2}^{-1} = \begin{pmatrix} \mathrm{M_2}^{-1} & 0 \\ 0 & \mathrm{M'_y}^{-1} \end{pmatrix}$$

The extended inverses of the invertible matrices can be used to encrypt only the extended query vector without re-encrypting the whole query vector. This encrypted query vector after extension, i.e., the trapdoor can be sent to the cloud server, which determines the scores of the documents using the secure inner product similarity measure and sends the top-k relevant documents among them to the users.

6.5 THEORETICAL AND SECURITY ANALYSIS

In this section, time complexity, and security analysis of the proposed max-heap based tree index are presented.

a) Time Complexity

The time complexity of the proposed Max-heap based tree index is analyzed with respect to the index construction, search and dynamic updates.

Index construction: Each node in the proposed tree consists of a TF vector, which is encrypted by using two invertible matrices of size $m \times m$ and a Max vector, which is also encrypted by using the same two invertible and D_{score}, which is a document score

encrypted by using the proposed Enhanced one-to-Many OPE scheme. Therefore, the total time complexity of index construction using the proposed Max-heap based tree index structure is $O(nm^2)$, where n is the number of documents in the dataset and m is the number of unique keywords in the dictionary of the dataset.

Search: The time complexity of searching for top-k relevant documents using the proposed max-heap based tree index is $O(\mathrm{m}log\mathrm{n})$, which is lesser than the time complexity of searching using the KBB tree index is $O(mlogn + m)$.

Dynamic updates: The time complexity of performing any dynamic update operation (Insert, Delete and Modify) over existing encrypted index using the proposed tree index structure is $O(m^2logn)$. As each node in the proposed tree index represents a document, the time complexity for any update is $O(m^2logn)$. Where as in KBB tree, only the leaf nodes represent the documents because of which all nodes along the path from the root node to leaf node must be visited for performing any dynamic update operation. The time complexity of performing each dynamic update using KBB tree index structure is $O(m^2logn + m)$. Thus, proposed index structure is efficient.

b) Security

The objective of the proposed max-heap based tree index structure is to retrieve the top-k relevant documents efficiently for a given trapdoor while supporting dynamic updates. In this approach, both the content of indexes and queries are represented in vectors and then encrypted using two randomized symmetric invertible matrices. As the matrices are randomly generated and each element of the matrices are random, the privacy of indexes and queries is guaranteed since they are not accessible to the cloud server. The privacy of dynamic updates with respect to Insert, Delete and Modify operations over encrypted documents is explained below.

Privacy of incorporating newly inserted documents: To incorporate the index information of new documents in an already existing tree index, the data owner needs to communicate nodes corresponding to these new documents and each node contains encrypted information, i.e., encrypted $TFVector$, $MaxVector$,D_score and the point-

ers P_l, P_r, P_p set to addresses of corresponding parent and child nodes based on the plaintext tree index. The data owner also communicates $MaxVectorList$ along with the nodes. The $MaxVectorList$ contains of list of identities of the nodes and the corresponding encrypted $MaxVector$s. The $MaxVectorList$ is useful for updating the $MaxVector$ at the corresponding nodes after inserting the new nodes. The cloud server then stores these nodes at appropriate positions in an already existing tree index and calls the heapify procedure at each new node based on the encrypted D_scores. Since the cloud server does not have any access to plaintext information while incorporating indexes of newly inserted documents, the cloud server cannot infer any information while incorporating index information of newly inserted documents.

Privacy of deleting the existing documents: To delete the intended documents, the data owner communicates only the list of document identities and $MaxVectorList$ to the cloud server. The cloud server then deletes the nodes that contain these identities and calls the heapify_downward(.) procedure at the respective deleted nodes for heapifying the tree with the help of $MaxVectorList$. The cloud server cannot infer any information with the deletion of documents since the cloud server knows only the list of document identities as they cannot convey any information about the content of the deleted documents.

Privacy of newly added keywords: Addition of new genuine keywords to the end of existing keyword dictionary reduces the privacy as the positions of newly inserted keywords are revealed to the cloud server. Though the vector information of newly added keywords is in encrypted form, the positions of incorporating this information cannot be prevented from the leakage. In our proposed dictionary expansion approach, the position of newly added genuine keyword is secured as the number of dummy keywords inserted are always random. Along with this, the position of the genuine keyword is randomly chosen on every subsequent addition. The probability of the cloud server coming to know the position of newly added genuine keyword is $1/2^{\mathtt{y}}$, where $\mathtt{y-1}$ is the number of dummy keywords added along with the genuine keyword. This deviates the cloud server from knowing the positions of newly included keywords into the already vector. The privacy of newly added keywords is preserved since the cloud server

cannot infer any pattern regarding the positions of the newly added keywords using the proposed dictionary expansion approach.

6.6 EXPERIMENTAL STUDY AND ANALYSIS

In this section, implementation methodology, experimental results of the proposed tree index, its efficiency, and usage are presented.

6.6.1 Implementation Methodology

The proposed approach has been implemented using Python 3.6 version, and the experiments of the proposed approach are conducted on Intel Xeon 2.6 GHz processor. Request for Comments (RFC) (RFC 2016) dataset has been used to test the efficiency of the proposed index structure. For evaluating the proposed approach, the time efficiency of index construction, search operation, and dynamic updates have been considered.

6.6.2 Experimental Results

The performance details of the proposed index structure with respect to index construction, searching, and dynamic updates (Insert, Delete and Modify) are provided below.

Index Construction: As there exists only one Keyword Balanced Binary tree (KBB) tree index structure that supports ranked search, we have compared the index construction time of the proposed tree with the KBB tree. The information in each node of the proposed tree index consists of two vectors $TFVector$, $MaxVector$, and a D_iScore, while the information in each leaf node of the KBB tree index consists of TFvector of the document and the each internal node consists of $MaxVector$. The $TFVector$ and D_iScore are constant in each node of the tree while the $MaxVector$ content changes as the new nodes are incorporated into the tree. Similarly, the $TFVector$ in all leaf nodes constant while the $MaxVector$ changes, which takes maximum TF values of both left and right child nodes. The comparison of index construction time is also performed using a different number of indexed documents and the results are shown in Figure 6.4a. Figure 6.4b depicts the construction time of these two structures for a fixed number of documents with a varied number of dictionary keywords (different number of keywords

in each document vector). It can be observed from these results that the index construction time of the proposed index structure is slightly efficient than the KBB based index construction time due to the reduction in size of the tree index, i.e., the height and breadth of the tree index. The number of nodes in the last level of the KBB tree index is equal to the number of nodes in all levels of the proposed tree index. This reduces the height and breadth of the tree and thus, improves the index construction time.

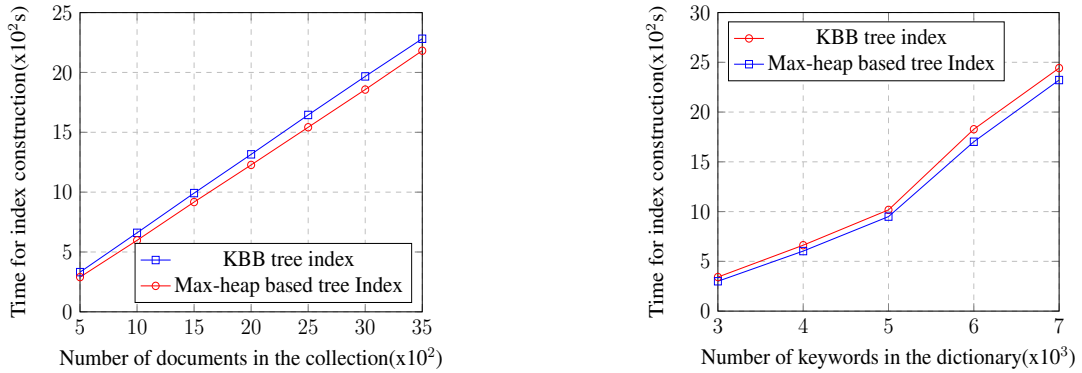


Figure 6.4: Index construction time for (a) constant size of dictionary, m = 4000 and for various cardinalities of document collections, and (b) for fixed document collection, n = 1000 and with varied number of keywords in dictionary.

Searching: Search is performed on both KBB tree and proposed tree for a given trapdoor using a greedy depth first search approach. At each node u in the proposed tree index, two scores are determined, i.e., $Rscore$ and $Maxscore$. The $Rscore$ of a node and its document identity $id(D_i)$ is added to the top-k list only if the $Rscore$ is greater than the k^{th} score in the top-k list. It stops searching the left and right subtrees when the $maxScore$ at a parent node u is not greater than the k^{th} score in the top-k list. While in KBB tree, only one score is determined, i.e., $maxscore$ at the internal nodes and $Rscore$ of a document only at the leaf nodes of the KBB tree index. Therefore, to determine top-k relevant documents for a given trapdoor, it must visit leaf nodes of the tree in the KBB tree index but not necessary in the proposed max-heap based tree index. The comparison of proposed tree index and the KBB tree index in terms of search time is given in Figure 6.5. It can be observed that searching time of the proposed max-heap based tree index is better than the KBB tree index. This is due to the less number of nodes in the proposed tree. Though the two scores are determined at each node in the

proposed tree index compared to only one score either $maxscore$ or $Rscore$ in KBB tree, the searching time of the proposed tree index is still efficient. The less number of nodes in tree reduces the height and breadth of the proposed tree index and thereby avoids computing $Rscore$ and $maxscore$ on many nodes. The searching process using the proposed tree index also avoids visiting left and right child nodes if the $maxscore$ at node u is not greater than the k^{th} score of the element in the top-k list. Thus, the search time of the proposed tree index is efficient than the search time of the KBB tree index. This is achieved by reducing the size of the tree and by avoiding many internal nodes that are not relevant while processing the given trapdoor.

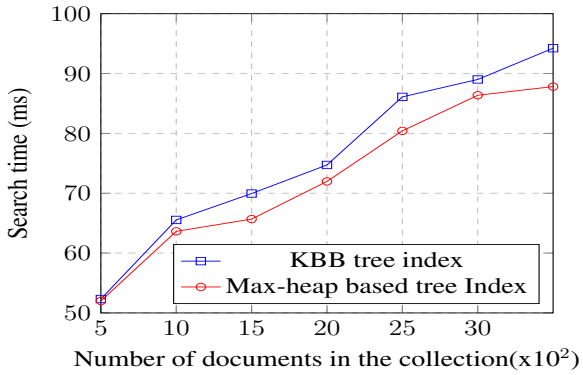


Figure 6.5: Search time for various document collection sizes with constant dictionary size, m = 4000

Dynamic Updates: The time of performing dynamic updates includes the time for performing update operations and also the time required for accommodating the updated content in an already existing tree index. We have compared the time of performing dynamic updates on the proposed tree index and the KBB tree index.

Inserting new documents: When a set of new documents are to be inserted, it is required to accommodate the index content of the newly inserted documents in an existing tree index. The time of accommodating the index content of new documents includes the time of generating new of nodes containing index information ($TFVector$, $MaxVector$ and D_iscore) in encrypted for for each of the new documents, followed by the time of incorporating each new node as the last node from the left to right in the last level of the existing tree and finally the time required for heapifying the tree based on the encrypted D_iscore of newly inserted node.

While in KBB tree, the time required for accommodating the index content of new documents includes the time of generating leaf nodes that contain index information ($TFVector$) in encrypted form for each of the new documents, followed by the time of generating internal nodes containing encrypted $MaxVector$s , which are generated based on the leaf nodes' plaintext $TFVector$ and also includes the time of adjusting the $MaxVector$s of some other internal nodes that were already existing. The time required for inserting a new document using both the tree index structures over a different number of already indexed documents is shown in Figure 6.6. It can be noted from the results that the proposed tree index structure is comparatively better than the existing KBB tree index for incorporating the index content of newly inserted documents. This efficiency is due to the reduction in height and breadth of the proposed tree index and thereby reduction in the number of times the $MaxVector$s of nodes get affected in the proposed tree index. Thus, the proposed max-heap based tree index is efficient in inserting new documents.

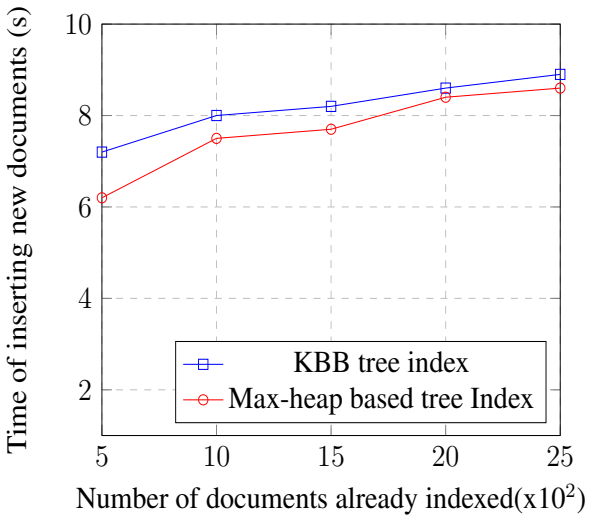


Figure 6.6: Time for inserting a new document over the different number of indexed documents.

Deleting existing documents: In the proposed max-heap based tree index, deleting a set of documents involves sending a list of document identities to the cloud server. The cloud server then first deletes the nodes that contain the given document identities from an existing index tree and then deletes the corresponding encrypted documents. The

time required for deleting a document includes the time required for identifying a node within the tree, replacing it with the last node of the tree index and then heapifying the tree index starting from the replaced node towards the leaf nodes of the tree for updating the $MaxVector$ information in the corresponding nodes. While in the KBB tree index, the time of deleting a node involves the time for identifying a leaf node for deletion and then setting the $TFVector$ of that node to null value, followed by updating the $MaxVector$s of parent nodes starting from the deleted node to root node. The comparison of deleting an existing document over different number of already indexed documents is shown in Figure 6.7. It is to be noted that deleting a document using the proposed max-heap based tree index is efficient than deleting a document using the KBB tree index due to less number of nodes and the information in nodes. This helps in avoid visiting many nodes. It is also not required to update the content of $MaxVector$ in each parent node along the path from the deleted node towards the root node. Thus, the proposed max-heap based tree index is efficient than the KBB tree index in deleting the existing documents.

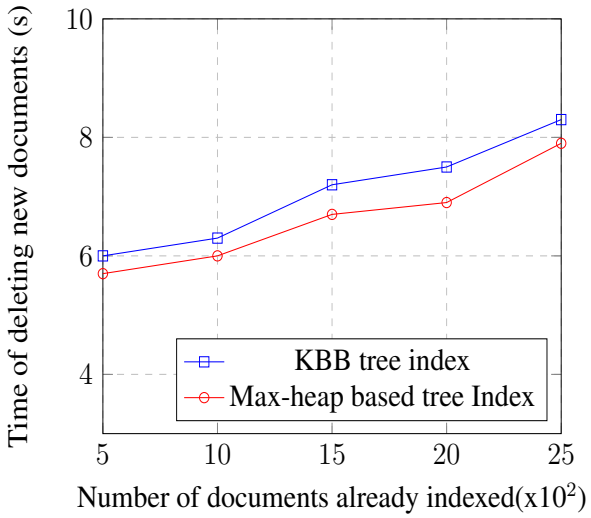


Figure 6.7: Time of deleting an existing document over the different number of indexed documents.

Modifying existing documents Modification of a document implies either adding new keywords or deleting existing keywords. In this research work, addition of new keywords to the existing documents is done securely using the proposed privacy pre-

serving keyword dictionary expansion approach, which is explained in Section 6.4.3.1. Addition of new keywords means adding the new keywords to the dictionary first and then incorporating the vector information of added keywords in each document's index of node u in an already existing tree index. Therefore, the time needed for incorporating the index content of the addition of new keywords includes the time taken to reflect the vector information of newly added keywords in the already existing dictionary. In the proposed approach, (`y - 1`) number of dummy keywords are added for every genuine keyword that also needs to be inserted.

In general, addition of new keywords to the already existing dictionary requires the re-creation of unencrypted tree idnex followed by the re-encrypting the content of the entire tree index. This incurs a huge cost because of having to re-encrypt the content of updated vectors in each node by using the new invertible random matrices. To avoid this, the proposed approach uses the partitioned based matrices approach by which only the updated vector information of added keywords can be incorporated in the already existing vectors at each node of the tree index. Since, the addition of keywords to existing dictionary securely is not done using the KBB tree index approach, the proposed privacy preserving dictionary expansion approach is compared with the re-encryption method for incorporating the index information of newly added keywords. Figure 6.8 demonstrates the time taken to add 100 genuine keywords over existing dictionary of different sizes using the proposed approach and re-encryption method. It can be noted that the proposed keyword dictionary expansion approach based on the partitioned matrices is more efficient than the re-encryption approach because the re-encryption involves re-computing the whole vectors again due to the addition of new keywords.

Efficiency The efficiency of the proposed approaches are analyzed with respect to the index construction and search time of the trapdoor and dynamic updates.

Index construction: It can be observed from the Figure 6.4 that the index construction time of our proposed max-heap based tree index is slightly efficient since the size of the tree is less in terms of number of nodes, height, and breadth. It is to be noted that the index construction is to be done only once by the data owner.

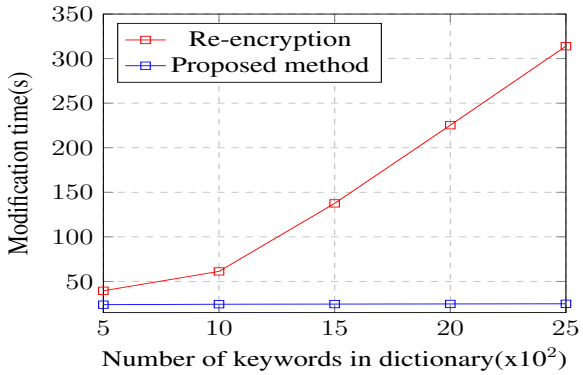


Figure 6.8: Modification time for newly added keywords to various dictionary sizes with fixed document collection, n = 1000 using proposed method and re-encryption method.

Search: From the Figure 6.5, it can be observed that time of searching using the proposed index structure is much efficient than searching using the KBB tree. This efficiency is due to the reduction of the proposed tree index size in terms of height and breadth compared to the size of KBB tree-based index structure. This reduction is achieved by merging the $MaxVector$ information along with $TFVector$ in each node of the tree instead of storing $MaxVector$ exclusively in internal nodes for guiding the search process to determine top-k relevant documents. Thus, searching can be efficiently performed using the proposed max-heap based tree index than the KBB tree index.

Dynamic updates: It can be observed from the Figures 6.6, 6.7 and 6.8 that the efficiency of dynamic updates, i.e., Insert, Delete and Modify using the proposed max-heap based tree index is efficient than performing dynamic updates using KBB tree index. This efficiency is due to reduction in height and breadth of the tree thereby helps in performing dynamic updates efficiently. Thus, proposed index structure is efficient.

6.6.3 Usage of the proposed tree indexing structure

As the proposed Max-heap based tree index is efficient in retrieving and performing update operations over the existing index, it could be used in the following use cases of retrieving encrypted data.

- All encrypted query and answering applications, where queries would be in the

form of like, finding out the best student of the year, extracting top-10 bidders information from a list of all the people who filed their bids for allocation of a government project and identifying the best suitable place for living in a country.

- Health care applications, where the proposed tree index would help in identifying patients who have been in a hospital for long, finding out which doctor has served more patients in a day, identifying underperforming doctors and seriously ill patients.

In spite of various advantages using the proposed tree index in different applications, the only concern with this approach is that the data owner has to store and maintain the plaintext version of uploaded encrypted index in order to perform the update operations correctly in an existing encrypted index. Therefore, the usage of the proposed approach is subjected to the storage capacity of the data owner.

6.7 SUMMARY

In this work, a max-heap based binary tree index structure is proposed to retrieves top-k relevant documents efficiently for a given trapdoor. It also supports dynamic update operations over encrypted documents efficiently. With the support of dynamic updates in the proposed tree index structure, it helps in returning the latest top-k relevant documents for a given trapdoor. The proposed tree index structure reduces both the height and breadth of the tree by storing $MaxVector$ along with the $TFVector$ in each node instead of storing $MaxVector$ exclusively in internal nodes for guiding the search operation to leaf nodes that contain $TFVector$ of actual documents. Thus, the proposed index structure performs search and dynamic updates efficiently. A secure keyword dictionary expansion approach is also proposed to allow the data owner to add a set of new keywords to the existing dictionary securely without re-encrypting the index. It preserves the privacy of newly added keywords by preventing the cloud server from knowing the locations of newly added encrypted keywords. The experimental results confirm that the proposed index structure is efficient than the existing KBB tree index with respect to the retrieval of top-k relevant documents and performing dynamic updates, and it also incorporates the updates securely in an already existing index.

CHAPTER 7

CONCLUSIONS AND FUTURE SCOPE

This dissertation has focused on addressing the issues of privacy concerns and supporting dynamic updates efficiently in order to enable the users to get the latest top-k relevant documents for their trapdoors. An Enhanced One-to-Many order preserving encryption scheme is developed to address the problem of mitigating the frequency leakage in searchable indexes. It mitigates the frequency leakage due to the improved randomness of a seed value because of which the repetition of same plaintext TF values in index would be mapped to different ciphertext values. Thus, it mitigates the frequency leakage of same ciphertext values in searchable index. The experimental results confirm that the proposed scheme reduces not only the frequency leakage of individual keywords but also the co-occurring keywords. The proposed scheme can be used for encrypting sensitive information that is moderately distributed.

A pseudo-ranking approach is developed to address the problem of preventing the leakages of both rank information and search pattern to the cloud server while retrieving top-k relevant documents for the given trapdoors. The search pattern leakage is prevented by including more random keywords in a trapdoor. The rank-order information leakage is also prevented by assigning perturbed random values to the random keywords in searchable indexes. The experimental results confirm that the proposed approach guarantees higher precision and higher privacy without affecting precision with the help of the intermediate server.

A max-heap based binary tree index structure has been developed to address the issue of supporting dynamic updates efficiently. The accommodation of updates in an already existing index enables the users to get the latest top-k relevant documents for their trapdoors. The experimental results demonstrate that the proposed indexing structure is efficient in performing dynamic updates and retrieving top-k relevant documents by reducing the size of the tree. A secure keyword dictionary expansion approach has also been developed to securely add a set of new keywords to the existing dictionary. The experimental results demonstrate that this approach supports modify operation efficiently without leaking the locations of the newly added keywords to the cloud server.

FUTURE SCOPE

This research work can be extended in the following directions to improve their applicability in diverse applications and further to improve precision, privacy, and efficiency.

- The Enhanced One-to-Many OPE scheme mitigates the frequency leakage of phrases and keywords than the existing order preserving encryption schemes. However, this scheme returns a large ciphertext values. These values impact efficiency while applying similarity measures on such large ciphertext values. Hence, it can be focused on improving the size of ciphertext values while preventing the frequency leakage.
- The Pseudo-Ranking approach can be extended to support a multi-owner and multi-user architecture model, where any user can search over encrypted data of multiple owners instead of a single owner's document. The main challenge involved in this model is to generate a common searchable index that represents the documents of all the data owners and allowing any user to search over this index securely and efficiently.
- The proposed max-heap based binary tree index can be extended to meet forward privacy and backward privacy.
- The proposed secure keyword dictionary expansion approach can be extended to handle deletion of existing keywords without affecting the existing indexes.

BIBLIOGRAPHY

Abdalla, M., Bellare, M., Catalano, D., Kiltz, E., Kohno, T., Lange, T., Malone-Lee, J., Neven, G., Paillier, P. and Shi, H. (2005). "Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions." In *Proceedings of the 25th Annual International Conference on Advances in Cryptology*, CRYPTO'05, Springer-Verlag, Berlin, Heidelberg, 205–222.

Agrawal, R., Kiernan, J., Srikant, R. and Xu, Y. (2004). "Order preserving encryption for numeric data." In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, ACM, New York, NY, USA, 563–574.

Armknecht, F. and Dewald, A. (2015). "Privacy-preserving email forensics." *Digit. Investig.*, 14(S1), S127–S136.

Arriaga, A., Tang, Q. and Ryan, P. (2014). *Trapdoor Privacy in Asymmetric Searchable Encryption Schemes*, 31–50. Springer International Publishing, Cham.

Bellare, M., Boldyreva, A. and O'Neill, A. (2007). "Deterministic and efficiently searchable encryption." In *Proceedings of the 27th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO'07, Springer-Verlag, Berlin, Heidelberg, 535–552.

Bellare, M., Fischlin, M., O'Neill, A. and Ristenpart, T. (2008). "Deterministic encryption: Definitional equivalences and constructions without random oracles." In *Proceedings of the 28th Annual Conference on Cryptology: Advances in Cryptology*, CRYPTO 2008, Springer-Verlag, Berlin, Heidelberg, 360–378.

Bellare, M. and Goldwasser, S. (2008). "Lecture notes on cryptography.").

Bentley, J. L. (1975). "Multidimensional binary search trees used for associative searching." *Commun. ACM*, 18(9), 509–517.

Bindschaedler, V., Grubbs, P., Cash, D., Ristenpart, T. and Shmatikov, V. (2018). "The tao of inference in privacy-protected databases." *Proc. VLDB Endow.*, 11(11), 1715–1728.

Bloom, B. H. (1970). "Space/time trade-offs in hash coding with allowable errors." *Commun. ACM*, 13(7), 422–426.

Boldyreva, A., Chenette, N., Lee, Y. and O'Neill, A. (2009). "Order-preserving symmetric encryption." In *Proceedings of the 28th Annual International Conference on Advances in Cryptology: The Theory and Applications of Cryptographic Techniques*, EUROCRYPT '09, Springer-Verlag, Berlin, Heidelberg, 224–241.

Boneh, D. (1998). *The Decision Diffie-Hellman problem*, 48–63. Springer Berlin Heidelberg, Berlin, Heidelberg.

Boneh, D., Di Crescenzo, G., Ostrovsky, R. and Persiano, G. (2004a). "Public key encryption with keyword search." In Cachin, C. and Camenisch, J. L., editors, *Advances in Cryptology - EUROCRYPT 2004*, Springer Berlin Heidelberg, Berlin, Heidelberg, 506–522.

Boneh, D., Di Crescenzo, G., Ostrovsky, R. and Persiano, G. (2004b). *Public Key Encryption with Keyword Search*, 506–522. Springer Berlin Heidelberg, Berlin, Heidelberg.

Boneh, D., Kushilevitz, E., Ostrovsky, R. and Skeith, III., W. E. (2007). "Public key encryption that allows pir queries." In *Proceedings of the 27th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO'07, Springer-Verlag, Berlin, Heidelberg, 50–67.

Boneh, D., Sahai, A. and Waters, B. (2011). "Functional encryption: Definitions and challenges." In *Proceedings of the 8th Conference on Theory of Cryptography*, TCC'11, Springer-Verlag, Berlin, Heidelberg, 253–273.

Boneh, D. and Waters, B. (2007). "Conjunctive, subset, and range queries on encrypted data." In *Proceedings of the 4th Conference on Theory of Cryptography*, TCC'07, Springer-Verlag, Berlin, Heidelberg, 535–554.

Bösch, C., Hartel, P., Jonker, W. and Peter, A. (2014). "A survey of provably secure searchable encryption." *ACM Comput. Surv.*, 47(2).

Boyen, X. and Waters, B. (2006). *Anonymous Hierarchical Identity-Based Encryption (Without Random Oracles)*, 290–307. Springer Berlin Heidelberg, Berlin, Heidelberg.

Büttcher, S. and Clarke, C. L. A. (2005). "A security model for full-text file system search in multi-user environments." In *Proceedings of the 4th Conference on USENIX Conference on File and Storage Technologies - Volume 4*, FAST'05, USENIX Association, Berkeley, CA, USA, 13–13.

Cao, N., Wang, C., Li, M., Ren, K. and Lou, W. (2014). "Privacy-preserving multi-keyword ranked search over encrypted cloud data." *IEEE Transactions on Parallel and Distributed Systems*, 25(1), 222–233.

Cash, D., Grubbs, P., Perry, J. and Ristenpart, T. (2015). "Leakage-abuse attacks against searchable encryption." In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, ACM, New York, NY, USA, 668–679.

Cash, D., Jarecki, S., Jutla, C., Krawczyk, H., Roşu, M.-C. and Steiner, M. (2013). *Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries*, 353–373. Springer Berlin Heidelberg, Berlin, Heidelberg.

Chang, Y.-C. and Mitzenmacher, M. (2005). "Privacy preserving keyword searches on remote encrypted data." In *Proceedings of the Third International Conference on Applied Cryptography and Network Security*, ACNS'05, Springer-Verlag, Berlin, Heidelberg, 442–455.

Chen, C., Zhu, X., Shen, P., Hu, J., Guo, S., Tari, Z. and Zomaya, A. Y. (2016). "An efficient privacy-preserving ranked keyword search method." *IEEE Transactions on Parallel and Distributed Systems*, 27(4), 951–963.

Chor, B., Kushilevitz, E., Goldreich, O. and Sudan, M. (1998). "Private information retrieval." *J. ACM*, 45(6), 965–981.

Cui, S., Belguith, S., Zhang, M., Asghar, M. R. and Russello, G. (2018). "Preserving access pattern privacy in sgx-assisted encrypted search." In *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, 1–9.

Curtmola, R., Garay, J., Kamara, S. and Ostrovsky, R. (2006). "Searchable symmetric encryption: Improved definitions and efficient constructions." In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, ACM, New York, NY, USA, 79–88.

Dai, H., Ji, Y., Yang, G., Huang, H. and Yi, X. (2020). "A privacy-preserving multi-keyword ranked search over encrypted data in hybrid clouds." *IEEE Access*, 8, 4895–4907.

Dawoud, M. and Altilar, D. T. (2017). "Cloud-based e-health systems: Security and privacy challenges and solutions." In *2017 International Conference on Computer Science and Engineering (UBMK)*, 861–865.

Diffie, W. and Hellman, M. (1976). "New directions in cryptography." *IEEE Transactions on Information Theory*, 22(6), 644–654.

Drazen, W. A., Ekwedike, E. and Gennaro, R. (2015). "Highly scalable verifiable encrypted search." In *Communications and Network Security (CNS), 2015 IEEE Conference on*, 497–505.

Durak, F. B., DuBuisson, T. M. and Cash, D. (2016). "What else is revealed by order-revealing encryption?." In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, ACM, New York, NY, USA, 1155–1166.

El Makkaoui, K., Ezzati, A., Beni-Hssane, A. and Motamed, C. (2016). "Cloud security and privacy model for providing secure cloud services." In *2016 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech)*, 81–86.

Fabian, B., Ermakova, T. and Junghanns, P. (2015). "Collaborative and secure sharing of healthcare data in multi-clouds." *Inf. Syst.*, 48(C), 132–150.

Feng, T. and He, W. (2018). "Research on privacy preserving of searchable encryption." In *Proceedings of the 2018 2nd High Performance Computing and Cluster Technologies Conference*, HPCCT 2018, Association for Computing Machinery, New York, NY, USA, 58–68.

Freedman, M. J., Nissim, K. and Pinkas, B. (2004). *Efficient Private Matching and Set Intersection*, 1–19. Springer Berlin Heidelberg, Berlin, Heidelberg.

Fu, Z., Liu, Y., Sun, X., Liu, Y. and Tian, Z. (2020). "Confusing-keyword based secure search over encrypted cloud data." *Mobile Networks and Applications*, 1–8.

Fu, Z., Sun, X., Linge, N. and Zhou, L. (2014). "Achieving effective cloud search services: multi-keyword ranked search over encrypted cloud data supporting synonym query." *IEEE Transactions on Consumer Electronics*, 60(1), 164–172.

Fu, Z., Wu, X., Wang, Q. and Ren, K. (2017). "Enabling central keyword-based semantic extension search over encrypted outsourced data." *IEEE Transactions on Information Forensics and Security*, 12(12), 2986–2997.

Gardiyawasam Pussewalage, H. S. and Oleshchuk, V. A. (2016). "Privacy preserving mechanisms for enforcing security and privacy requirements in e-health solutions." *Int. J. Inf. Manag.*, 36(6), 1161–1173.

Ghosh Ray, I., Rahulamathava, Y. and Rajarajan, M. (2018). "A new lightweight symmetric searchable encryption scheme for string identification." *IEEE Transactions on Cloud Computing*, 1–1.

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

Goldreich, O. and Ostrovsky, R. (1996a). "Software protection and simulation on oblivious rams." *J. ACM*, 43(3), 431–473.

Goldreich, O. and Ostrovsky, R. (1996b). "Software protection and simulation on oblivious rams." *Journal of the ACM (JACM)*, 43(3), 431–473.

Goldwasser, S. and Micali, S. (1984). "Probabilistic encryption." *Journal of Computer and System Sciences*, 28(2), 270 – 299.

Golle, P., Staddon, J. and Waters, B. (2004). *Secure Conjunctive Keyword Search over Encrypted Data*, 31–45. Springer Berlin Heidelberg, Berlin, Heidelberg.

Grubbs, P., Lacharite, M.-S., Minaud, B. and Paterson, K. G. (2018). "Pump up the volume: Practical database reconstruction from volume leakage on range queriezzs." In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, 315–331.

Grubbs, P., Sekniqi, K., Bindschaedler, V., Naveed, M. and Ristenpart, T. (2017). "Leakage-abuse attacks against order-revealing encryption." In *2017 IEEE Symposium on Security and Privacy (SP)*, 655–672.

Guo, C., Zhuang, R., Chang, C. and Yuan, Q. (2019). "Dynamic multi-keyword ranked search based on bloom filter over encrypted cloud data." *IEEE Access*, 7, 35826–35837.

Hwang, Y. H. and Lee, P. J. (2007). "Public key encryption with conjunctive keyword search and its extension to a multi-user system." In *Proceedings of the First International Conference on Pairing-Based Cryptography*, Pairing'07, Springer-Verlag, Berlin, Heidelberg, 2–22.

Ibrahim, A., Jin, H., Yassin, A. A. and Zou, D. (2012). "Secure rank-ordered search of multi-keyword trapdoor over encrypted cloud data." In *Services Computing Conference (APSCC), 2012 IEEE Asia-Pacific*, 263–270.

Jin, X., Agun, D., Yang, T., Wu, Q., Shen, Y. and Zhao, S. (2016). "Hybrid indexing for versioned document search with cluster-based retrieval." In *Proceedings of the*

25th ACM International on Conference on Information and Knowledge Management, CIKM '16, ACM, New York, NY, USA, 377–386.

John, T. M., Haider, S. K., Omar, H. and van Dijk, M. (2020). "Connecting the dots: Privacy leakage via write-access patterns to the main memory." *IEEE Transactions on Dependable and Secure Computing*, 17(2), 436–442.

Joux, A. (2002). "The weil and tate pairings as building blocks for public key cryptosystems." In *Proceedings of the 5th International Symposium on Algorithmic Number Theory*, ANTS-V, Springer-Verlag, London, UK, UK, 20–32.

Kamara, S. and Lauter, K. (2010). "Cryptographic cloud storage." In *International Conference on Financial Cryptography and Data Security*, Springer, 136–149.

Kamara, S. and Papamanthou, C. (2013). "Parallel and dynamic searchable symmetric encryption." In Sadeghi, A.-R., editor, *Financial Cryptography and Data Security*, Springer Berlin Heidelberg, Berlin, Heidelberg, 258–274.

Karame, G. O., Capkun, S. and Maurer, U. (2011). "Privacy-preserving outsourcing of brute-force key searches." In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, Association for Computing Machinery, New York, NY, USA, 101–112.

Katz, J., Sahai, A. and Waters, B. (2008). "Predicate encryption supporting disjunctions, polynomial equations, and inner products." In *Proceedings of the Theory and Applications of Cryptographic Techniques 27th Annual International Conference on Advances in Cryptology*, EUROCRYPT'08, Springer-Verlag, Berlin, Heidelberg, 146–162.

Kerschbaum, F. (2015). "Frequency-hiding order-preserving encryption." In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, ACM, New York, NY, USA, 656–667.

Kumar, D. S. and Thilagam, P. S. (2019a). "Approaches and challenges of privacy preserving search over encrypted data." *Information Systems*, 81, 63 – 81.

Kumar, D. S. and Thilagam, P. S. (2019b). "Searchable encryption approaches: attacks and challenges." *Knowledge and Information Systems*, 61(3), 1179–1207. cited By 1.

Kuzu, M., Islam, M. S. and Kantarcioglu, M. (2012). "Efficient similarity search over encrypted data." In *2012 IEEE 28th International Conference on Data Engineering*, IEEE, 1156–1167.

Lacharité, M., Minaud, B. and Paterson, K. G. (2018). "Improved reconstruction attacks on encrypted data using range query leakage." In *2018 IEEE Symposium on Security and Privacy (SP)*, 297–314.

Lee, D. L., Huei Chuang and Seamons, K. (1997). "Document ranking and the vector-space model." *IEEE Software*, 14(2), 67–75.

Li, H., Liu, D., Dai, Y., Luan, T. H. and Shen, X. S. (2015). "Enabling efficient multi-keyword ranked search over encrypted mobile cloud data through blind storage." *IEEE Transactions on Emerging Topics in Computing*, 3(1), 127–138.

Li, J., Wang, Q., Wang, C., Cao, N., Ren, K. and Lou, W. (2010). "Fuzzy keyword search over encrypted data in cloud computing." In *INFOCOM, 2010 Proceedings IEEE*, IEEE, 1–5.

Li, R. and Liu, A. X. (2017). "Adaptively secure conjunctive query processing over encrypted data for cloud computing." In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, 697–708.

Li, R., Liu, A. X., Wang, A. L. and Bruhadeshwar, B. (2014a). "Fast range query processing with strong privacy protection for cloud computing." *Proc. VLDB Endow.*, 7(14), 1953–1964.

Li, R., Xu, Z., Kang, W., Yow, K. C. and Xu, C.-Z. (2014b). "Efficient multi-keyword ranked query over encrypted data in cloud computing." *Future Generation Computer Systems*, 30, 179 – 190. Special Issue on Extreme Scale Parallel Architectures and Systems, Cryptography in Cloud Computing and Recent Advances in Parallel and Distributed Systems, ICPADS 2012 Selected Papers.

Li, Z., Fang, R., Shen, F., Katouzian, A. and Zhang, S. (2017). "Indexing and mining large-scale neuron databases using maximum inner product search." *Pattern Recognition*, 63, 680 – 688.

Liang, J., Qin, Z., Xiao, S., Zhang, J., Yin, H. and Li, K. (2020). "Privacy-preserving range query over multi-source electronic health records in public clouds." *Journal of Parallel and Distributed Computing*, 135, 127 – 139.

Liu, X., Guan, Z., Du, X., Wu, L., Abedin, Z. U. and Guizani, M. (2019). "Achieving secure and efficient cloud search services: Cross-lingual multi-keyword rank search over encrypted cloud data." In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 1–6.

Maffei, M., Reinert, M. and Schröder, D. (2017). "On the security of frequency-hiding order-preserving encryption." In *CANS*.

Manning, C. D., Raghavan, P. and Schütze, H. (2008). *Introduction to Information Retrieval*, Cambridge University Press, New York, NY, USA.

Margae, S. E., Sanae, B., Mounir, A. K. and Youssef, F. (2014). "Traffic sign recognition based on multi-block lbp features using svm with normalization." In *2014 9th International Conference on Intelligent Systems: Theories and Applications (SITA-14)*, 1–7.

Miao, Y., Ma, J., Liu, X., Li, X., Jiang, Q. and Zhang, J. (2017). "Attribute-based keyword search over hierarchical data in cloud computing." *IEEE Transactions on Services Computing*, PP(99), 1–1.

Naveed, M., Kamara, S. and Wright, C. V. (2015). "Inference attacks on property-preserving encrypted databases." In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, ACM, New York, NY, USA, 644–655.

Naveed, M., Prabhakaran, M. and Gunter, C. A. (2014). "Dynamic searchable encryption via blind storage." In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, IEEE Computer Society, Washington, DC, USA, 639–654.

Orencik, C., Kantarcioglu, M. and Savas, E. (2013). "A practical and secure multi-keyword search method over encrypted cloud data." In *2013 IEEE Sixth International Conference on Cloud Computing*, 390–397.

Örencik, C. and Savaş, E. (2012). "Efficient and secure ranked multi-keyword search on encrypted cloud data." In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, EDBT-ICDT '12, ACM, New York, NY, USA, 186–195.

Pan, Y., Efrat, A., Li, M., Wang, B., Quan, H., Mitchell, J., Gao, J. and Arkin, E. (2020). "Data inference from encrypted databases: A multi-dimensional order-preserving matching approach." *arXiv preprint arXiv:2001.08773*.

Park, J. H. (2011). "Efficient hidden vector encryption for conjunctive queries on encrypted data." *IEEE Transactions on Knowledge and Data Engineering*, 23(10), 1483–1497.

Rahman, M. F., Liu, W., Thirumuruganathan, S., Zhang, N. and Das, G. (2014). "Rank-based inference over web databases." *ArXiv*, abs/1411.1455.

Rahman, M. F., Liu, W., Thirumuruganathan, S., Zhang, N. and Das, G. (2015). "Privacy implications of database ranking." *Proc. VLDB Endow.*, 8(10), 1106–1117.

Rao, D., Kumar, D. V. N. S. and Thilagam, P. S. (2018). "An efficient multi-user searchable encryption scheme without query transformation over outsourced encrypted data." In *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 1–4.

Rashmi, R., Kumar, D. S. and Thilagam, P. S. (2018). "Privacy-preserving searchable encryption scheme over encrypted data supporting dynamic update." In *International Symposium on Security in Computing and Communication*, Springer, 24–38.

RFC (2016). "Request for comments database." `https://www.rfc-editor.org/retrieve/bulk/`).

Ryu, E. K. and Takagi, T. (2007). "Efficient conjunctive keyword-searchable encryption." In *Advanced Information Networking and Applications Workshops, 2007, AINAW '07. 21st International Conference on*, volume 1, 409–414.

Shahzad, F. (2014). "State-of-the-art survey on cloud computing security challenges, approaches and solutions." *Procedia Computer Science*, 37, 357 – 362. The 5th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2014)/ The 4th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH 2014)/ Affiliated Workshops.

Shen, E., Shi, E. and Waters, B. (2009). "Predicate privacy in encryption systems." In *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, TCC '09, Springer-Verlag, Berlin, Heidelberg, 457–473.

Shen, Z., Shu, J. and Xue, W. (2018). "Preferred search over encrypted data." *Frontiers of Computer Science*, 12(3), 593–607.

Song, D. X., Wagner, D. and Perrig, A. (2000). "Practical techniques for searches on encrypted data." In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, SP '00, IEEE Computer Society, Washington, DC, USA, 44–.

Song, W., Wang, B., Wang, Q., Peng, Z., Lou, W. and Cui, Y. (2017). "A privacy-preserved full-text retrieval algorithm over encrypted data for cloud storage applications." *Journal of Parallel and Distributed Computing*, 99(Supplement C), 14 – 27.

Subashini, S. and Kavitha, V. (2011). "A survey on security issues in service delivery models of cloud computing." *Journal of Network and Computer Applications*, (1), 1 – 11.

Sun, W., Wang, B., Cao, N., Li, M., Lou, W., Hou, Y. T. and Li, H. (2013). "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking." In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, ACM, New York, NY, USA, 71–82.

Swaminathan, A., Mao, Y., Su, G.-M., Gou, H., Varna, A. L., He, S., Wu, M. and Oard, D. W. (2007). "Confidentiality-preserving rank-ordered search." In *Proceedings of the 2007 ACM workshop on Storage security and survivability*, ACM, 7–12.

Tahir, S., Ruj, S., Rahulamathavan, Y., Rajarajan, M. and Glackin, C. (2017). "A new secure and lightweight searchable encryption scheme over encrypted cloud data." *IEEE Transactions on Emerging Topics in Computing*, 1–1.

Tang, J., Cui, Y., Li, Q., Ren, K., Liu, J. and Buyya, R. (2016). "Ensuring security and privacy preservation for cloud data services." *ACM Comput. Surv.*, 49(1).

Tang, Q. (2012). "Search in encrypted data: Theoretical models and practical applications.." *IACR Cryptology ePrint Archive*, 2012, 648.

Tari, Z. (2014). "Security and privacy in cloud computing." *IEEE Cloud Computing*, 1(1), 54–57.

Wang, B., Song, W., Lou, W. and Hou, Y. T. (2015). "Inverted index based multi-keyword public-key searchable encryption with strong privacy guarantee." In *2015 IEEE Conference on Computer Communications (INFOCOM)*, 2092–2100.

Wang, B., Yu, S., Lou, W. and Hou, Y. T. (2014a). "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud." In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, 2112–2120.

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

Wang, Q., Zhu, Y. and Luo, X. (2014b). "Multi-user searchable encryption with fine-grained access control without key sharing." In *2014 3rd International Conference on Advanced Computer Science Applications and Technologies*, 145–150.

Waters, B. R., Balfanz, D., Durfee, G. and Smetters, D. K. (2004). "Building an encrypted and searchable audit log.." In *NDSS*, volume 4, 5–6.

Witten, I. H., Bell, T. C. and Moffat, A. (1994). *Managing Gigabytes: Compressing and Indexing Documents and Images*, John Wiley & Sons, Inc., New York, NY, USA, 1st edition.

Wong, K.-S. and Kim, M. H. (2013). "Privacy-preserving similarity coefficients for binary data." *Computers & Mathematics with Applications*, 65(9), 1280 – 1290. Advanced Information Security.

Wu, D. J. (2015). "Fully homomorphic encryption: Cryptography's holy grail." *XRDS*, 21(3), 24–29.

Wu, Z. and Li, K. (2019). "Vbtree: Forward secure conjunctive queries over encrypted data for cloud computing." *The VLDB Journal*, 28(1), 25–46.

Xia, Z., Wang, X., Sun, X. and Wang, Q. (2016). "A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data." *IEEE Transactions on Parallel and Distributed Systems*, 27(2), 340–352.

Xiangyu, W., Ma, J., Yinbin, M., Liu, X. and Ruikang, Y. (2019). "Privacy-preserving diverse keyword search and online pre-diagnosis in cloud computing." *IEEE Transactions on Services Computing*, 1–1.

Xu, J., Zhang, W., Yang, C., Xu, J. and Yu, N. (2012). "Two-step-ranking secure multi-keyword search over encrypted cloud data." In *2012 International Conference on Cloud and Service Computing*, 124–130.

Yan, Z., Li, X., Wang, M. and Vasilakos, A. V. (2017). "Flexible data access control based on trust and reputation in cloud computing." *IEEE Transactions on Cloud Computing*, 5(3), 485–498.

Yang, Y., Liu, X., Deng, R. H. and Weng, J. (2017). "Flexible wildcard searchable encryption system." *IEEE Transactions on Services Computing*, PP(99), 1–1.

Yu, J., Lu, P., Zhu, Y., Xue, G. and Li, M. (2013). "Toward secure multikeyword top-k retrieval over encrypted cloud data." *IEEE Transactions on Dependable and Secure Computing*, 10(4), 239–250.

Yu, S., Wang, C., Ren, K. and Lou, W. (2010). "Achieving secure, scalable, and fine-grained data access control in cloud computing." In *2010 Proceedings IEEE INFOCOM*, 1–9.

Yunling WANG, Jianfeng WANG, X. C. (2016). "Secure searchable encryption: a survey." *Journal of Communications and Information Networks*, 1(4), 52.

Zerr, S., Demidova, E., Olmedilla, D., Nejdl, W., Winslett, M. and Mitra, S. (2008). "Zerber: r-confidential indexing for distributed documents." In *EDBT*, volume 261 of *ACM International Conference Proceeding Series*, ACM, 287–298.

Zerr, S., Olmedilla, D., Nejdl, W. and Siberski, W. (2009). "Zerber+ r: Top-k retrieval from a confidential index." In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, ACM, 439–449.

Zhang, B. and Zhang, F. (2011). "An efficient public key encryption with conjunctive-subset keywords search." *J. Netw. Comput. Appl.*, 34(1), 262–267.

Zhang, W., Ji, J., Zhu, J., Li, J., Xu, H. and Zhang, B. (2016a). "Bithash: An efficient bitwise locality sensitive hashing method with applications." *Knowledge-Based Systems*, 97, 40 – 47.

Zhang, W., Lin, Y., Xiao, S., Wu, J. and Zhou, S. (2016b). "Privacy preserving ranked multi-keyword search for multiple data owners in cloud computing." *IEEE Transactions on Computers*, 65(5), 1566–1577.

PUBLICATIONS

1. Kumar, D.V.N.S. and Thilagam, P. S. (2019). Approaches and challenges of privacy preserving search over encrypted data, *Information Systems*, Elsevier, Volume 81, Pages 63-81. (DOI: https://doi.org/10.1016/j.is.2018.11.004, (Impact Factor: 2.066)

2. Kumar, D.V.N.S. and Thilagam, P. S. (2019), Searchable encryption approaches: attacks and challenges, *Knowledge and Information Systems*, Springer, Volume 61 (3), Pages 1179-1207. (DOI: https://doi.org/10.1007/s10115-018-1309-4 (Impact Factor: 2.397).

3. Rashmi, R., Kumar, D.V.N.S. and Thilagam, P. S. (2018). Privacy-preserving searchable encryption scheme over encrypted data supporting dynamic update, *International Symposium on Security in Computing and Communication (SSCC)*, Springer CCIS, Bangalore. (DOI: https://doi.org/10.1007/978-981-13-5826-5_2.

4. Rao, D., Kumar, D.V.N.S. and Thilagam, P. S. (2018). An efficient multi-user searchable encryption scheme without query transformation over outsourced encrypted data, *9 th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, IEEE, Paris. (DOI:10.1109/NTMS.2018.8328677.

5. Kumar, D.V.N.S. and Thilagam, P. S. *High Precision Privacy Preserving Multi-Keyword Ranked Search over Encrypted Data*. *IEEE Transactions on Services and Computing* (Comments came and asked to resubmit the paper).

6. Kumar, D.V.N.S. and Thilagam, P. S. Enhanced One-to-Many OPE Scheme for

Mitigating Frequency Analysis Attack over Encrypted Data. *Journal of Information Security and Applications*. (Paper submitted on September 30, 2019).

BIO-DATA

Name	: D Venkata Naga Siva Kumar
Date of Birth	: 09-11-1989
Gender	: Male
Marital Status	: Single
Father's Name	: D Krishnamoorthy
Mother's Name	: D Krishnavenamma
Email Id	: dvnsivakumar@gmail.com
Mobile	: 9113603795, 7795545072
Present Address	: Flat No: 302, H4 Tower, BITS PILANI Hyderabad, Javahar Nagar, Medchal, Hyderabad, Telangana.
Educational Qualifications	:1) B.Tech (AMIETE) – CSE, 2007-2011 Institute of Electronics and Telecommunication Engineers, New Delhi. 2) M.Tech –CSE, 2011-2013 I.I.I.T Bhubaneswar, Odisha.
Teaching Experience	: Assistant Professor, CSE Department, July-2013 to June-2015, Vignan University, Vadlamudi, Guntur, Andhra Pradesh
Areas of Interest	: Cloud Data Security and Information Retrieval.