ANALYSING THE ENCRYPTED SEARCH ALGORITHMS ON ENCRYPTED DATA

**A Thesis Submitted to
the Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of**

MASTER OF SCIENCE

in Computer Engineering

**by
Leyla TEKİN**

**July 2018
İZMİR**

We approve the thesis of **Leyla TEKİN**

Examining Committee Members:

Assoc. Prof. Dr. Tolga AYAV
Department of Computer Engineering, İzmir Institute of Technology

Dr. Hüseyin HIŞIL
Department of Computer Engineering, Yaşar University

Dr. Serap ŞAHİN
Department of Computer Engineering, İzmir Institute of Technology

9 July 2018

Dr. Serap ŞAHİN
Supervisor, Department of Computer Engineering
İzmir Institute of Technology

Assoc. Prof. Dr. Yusuf Murat ERTEN
Head of the Department of
Computer Engineering

Prof. Dr. Aysun SOFUOĞLU
Dean of the Graduate School of
Engineering and Sciences

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my supervisor, *Dr. Serap Şahin*, for her guidance, patience, and suggestions throughout my master thesis.

I would like to thank *Dr. Hüseyin Hışıl* and *Assoc. Prof. Dr. Tolga Ayav* for their acceptance to evaluate this thesis.

I would also like to thank my colleagues and friends in Computer Engineering Department of İzmir Institute of Technology for their motivation during this thesis period. Especially, I am grateful to my room mate *Damla Yaşar* for her helpful insights and delicious chocolates.

Finally, I would like to thank my dear family, *my mother*, *my father*, and *my beloved sisters*, for their unconditional support and encouragement in all the stages of my life. I feel so honored to be a member of such a family to whom I dedicate this work.

ABSTRACT

ANALYSING THE ENCRYPTED SEARCH ALGORITHMS ON ENCRYPTED DATA

In this thesis, we study the static and dynamic Searchable Symmetric Encryption (SSE) schemes (Cash et al. (2014), Kamara and Moataz (2017)). We present different approaches for secure single- and multi-keyword ranked searches, that are: Sorted, OPE-Based, Paillier-Based, Embedded, and Matrix-Based. We extend the base schemes according to these approaches so that the matching documents of a search query are ranked by a relevance score calculation technique like term frequency (tf), term frequency-inverse document frequency (tf-idf) or keyword frequency, depending on the characteristics of the scheme. For this, the existing structures of the schemes are modified since they cannot be directly used for ranked searches. Therefore, the ranking facility is added to them. Further, Matrix-Based Approach is a new hybrid approach that is based on an updated structure of the static scheme (Cash et al. (2014)) and fills a matrix to rank the relevant documents for a search keyword, as in the work (Ibrahim et al. (2012)), however, computing the matrix is totally different from their work.

ÖZET

ŞİFRELİ VERİ ÜSTÜNDE ŞİFRELİ ARAMA ALGORİTMALARININ ANALİZİ

Bu tez çalışmasında, statik ve dinamik aranabilir simetrik şifreleme şemalarını inceliyoruz (Cash vd. (2014), Kamara ve Moataz (2017)). Tek ve çoklu kelime içeren sorgularla yapılan güvenli sıralı aramalar için farklı yaklaşımlar sunuyoruz: Sıralı, OPE-Tabanlı, Paillier-Tabanlı, Gömülü ve Matris-Tabanlı. Temel şemaları bu yaklaşımlara göre genişletiyoruz, böylece bir arama sorgusu ile eşleşen dokümanlar, kullanılan şemanın karakteristiklerine bağlı olarak terim frekansı, terim frekansı-ters doküman frekansı, ya da kelime frekansı gibi bir ilgi puanı hesaplama tekniğine göre sıralanır. Şemaların mevcut veri yapıları, sıralı aramalar için kullanılamadığından dolayı bu yapılara sıralama özelliği eklenerek değiştirilmiştir. Ayrıca, Matris-Tabanlı Yaklaşım, Cash vd.'nin statik şemasının güncellenmiş versiyonunu temel alan ve Ibrahim vd. (2012)'nin çalışmasında olduğu gibi bir aranan kelimeye ilgili olan dokümanları sıralamak için bir matristen yararlanan hibrit bir yapıya sahip yeni bir yaklaşımdır. Ancak, bu yaklaşımın matrisi hesaplama yöntemi Ibrahim vd.'nin çalışmasından tamamen farklıdır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

B	A big block size in the variants of RR2Lev scheme.
b	A small block size in the variants of RR2Lev scheme.

LIST OF ABBREVIATIONS

SSE	Searchable Symmetric Encryption.
RR2Lev	Response Revealing 2Lev.
RH2Lev	Response Hiding 2Lev.
DynRR	Dynamic Response Revealing.
DynRH	Dynamic Response Hiding.
SR-RR2Lev	Single Keyword Ranked Response Revealing 2Lev.
SR-RH2Lev	Single Keyword Ranked Response Hiding 2Lev.
SR-DynRR	Single Keyword Ranked Dynamic Response Revealing.
SR-DynRH	Single Keyword Ranked Dynamic Response Hiding.
MR-RR2Lev	Multi-Keyword Ranked Response Revealing 2Lev.
MR-RH2Lev	Multi-Keyword Ranked Response Hiding 2Lev.
MR-DynRR	Multi-Keyword Ranked Dynamic Response Revealing.
MR-DynRH	Multi-Keyword Ranked Dynamic Response Hiding.
OPE	Order Preserving Encryption.
BF	Bloom Filter.
AES	Advanced Encryption Standard.
CTR	Counter Mode.
HMAC	Hash-based Message Authentication Code.
PPM	Privacy Preserving Mapping.
ORAM	Oblivious RAM.
SE	Searchable Encryption.

CHAPTER 1

INTRODUCTION

In recent years, vast amounts of data are produced by several sources such as millions of digital sensors, social media applications, smart phones, financial transaction records, etc. Thanks to many capabilities offered by cloud computing, data owners and organizations have extensively moved their huge datasets from traditional local data centres to the cloud so that they can utilize the possibilities of greater flexibility and lower cost. However, this requires to be kept their sensitive data on remote untrusted servers and introduces new security and privacy challenges that needs to be handled. Therefore, the data is encrypted before sending to the untrusted servers in order to protect the data confidentiality. Although data encryption ensures data confidentiality, it certainly prevents the server from operating on the data like keyword-based search over it.

The search functionality enables a data user to receive the related data with a keyword from a remote data server. The proposed solutions to perform a keyword search over the encrypted data are (i) downloading all the stored data to the user side, decrypting it locally and searching the keyword over the decrypted data and (ii) allowing the server to decrypt the data, search the keyword, and return the related results to the user. The first approach downloads the entire data when a keyword search is performed, even if a very small part of the data is related to the search keyword. Hence, it leads to an increase in the communication overhead. The second approach allows the server to know the secret key and plaintext, thereby it creates a situation that is contrary to the intended privacy requirement.

On the other hand, various searchable encryption schemes have been developed to support searching over encrypted data in a secure and efficient way. In this thesis, we study Searchable Symmetric Encryption (SSE) schemes. Most of them focus on a single keyword or Boolean search and do not provide a relevance ranking in the search result. To take advantage of the result ranking, this thesis offers various single keyword and multi-keyword search schemes that enable ranked search over encrypted documents, which are analyzed theoretically in terms of efficiency, security, and functionality, implemented in Java, and their performances are compared using an up-to-date real dataset, namely the

Request For Comments (RFC), with regard to the following evaluation metrics: (i) the query efficiency, (ii) the communication overhead, and (iii) the storage overhead.

1.1. Motivation

This master thesis aims at answering the following research question?

- *Can we design and implement privacy preserving single- and multi-keyword ranked schemes based on the practically efficient data structures and approaches proposed by Cash et al. (2014) in order to rank the search results using a relevance calculation method?*

To answer this question, first we investigate the static SSE scheme (Cash et al. (2014)) and the dynamic SSE scheme in Clusion framework (Kamara and Moataz (2017)), a variant of the dynamic scheme of Cash et al., and combine other tools such as order preserving encryption (OPE) and partially homomorphic encryption to achieve more functionalities, i.e. ranked search, so that the matching documents of a search query are ranked by different relevance scores like term frequency (tf), term frequency-inverse document frequency (tf-idf) or keyword frequency, depending on the characteristics of the schemes.

1.2. Thesis Goals and Contributions

The objectives of this thesis are:

- To examine searchable encryption schemes with different properties in terms of efficiency, security and functionality aspects.
- To propose and implement a variety of ranked searchable encryption schemes that meet different requirements using approaches from information retrieval systems and cryptography, and determine the advantages and disadvantages of each scheme with detailed discussions.
- To compare our proposed schemes with the base searchable encryption schemes that we use to provide ranked search so as to understand the cost of this functionality, and previously proposed searchable encryption schemes with ranked results.

Our contributions in this thesis can be summarized as follows:

- We enhance the static SSE scheme (Cash et al. (2014)) to support single- and multi-keyword ranked searches over encrypted documents.
- We also extend the dynamic SSE scheme in Clusion framework (Kamara and Moataz (2017)) to provide it ranking facilities for single- and multi-keyword searches.
- Further, we introduce a new hybrid multi-keyword ranked searchable encryption approach that is based on an updated version of the static structures (Cash et al. (2014)) and fills a matrix to rank the relevant documents for a search keyword, as in the work (Ibrahim et al. (2012)), however, computing the matrix is totally different from their work.
- Efficiency and privacy analysis of the proposed schemes are performed.
- The technical knowledge and experience gained by the analysis and experimental work can help the researchers to choose the solution that matches their criteria.

1.3. Outline of Thesis

The thesis is organized as follows. Chapter 2 first presents system & threat model, defines security requirements and design goals, and then gives general information about the research background on information retrieval, cryptography and data structures. Chapter 3 presents the related work in the literature about Searchable Symmetric Encryption and Ranked Searchable Encryption. Chapter 4 provides the base schemes that are (i) response-revealing 2Lev (RR2Lev), (ii) response-hiding 2Lev (RH2Lev), (iii) dynamic response-revealing (DynRR), (iv) dynamic response-hiding (DynRH), and (v) secure ranked multi-keyword SE scheme (SRMES). Each of these schemes are examined in detail. Chapter 5 describes our proposed approaches for ranked searchable encryption which are (i) Sorted, (ii) OPE-Based, (iii) Paillier-Based, (iv) Embedded, and (v) Matrix-Based. Then, the schemes we introduce for these approaches are explained. Chapter 6 gives the theoretical analysis of base and proposed schemes. Chapter 7 includes the experiments and discussions on them. Finally, Chapter 8 gives the conclusion and future work of the thesis.

CHAPTER 2

BACKGROUND

2.1. System & Threat Model

In the system architecture of our proposed secure ranked schemes, there are three different types of entities: data owner, data user, and cloud server. The data owner has a document collection $D = \{D_1, ..., D_n\}$ and wishes to outsource it in an encrypted form to the cloud server. Hence, first the data owner extracts a set of distinct keywords $W = \{w_1, ..., w_m\}$ and relevance score values (to enable ranking) from the document collection and builds an encrypted searchable inverted index, I. Then, the data owner generates the encrypted document collection $C = \{C_1, ..., C_n\}$ using a symmetric encryption scheme (e.g., AES), and submits both the encrypted document collection C and the index I to the cloud server. After that, to retrieve the related documents matching a single- or multi-keyword query, an authorized user sends the search token for the query to the cloud server. Once receiving the search request, the cloud server searches over the index I with the search token and performs one of the following operations depending on the chosen scheme:

(i) The cloud server directly returns the ranked document identifiers.

(ii) The cloud server directly returns the ranked encrypted document identifiers, and then the user decrypts the received document identifiers.

(iii) The cloud server sorts the document identifiers by the encrypted scores and returns the ranked document identifiers.

(iv) The cloud server sorts the encrypted document identifiers by the encrypted scores and returns the ranked encrypted document identifiers, and then the user decrypts the received document identifiers.

(v) The cloud server returns the (document identifier-encrypted score) pairs, and then the user decrypts the received scores and sorts the document identifiers by these scores.

(vi) The cloud server returns the (encrypted document identifier-encrypted score) pairs, and then the user decrypts both the received document identifiers and scores, and sorts the document identifiers by these scores.

(vii) The cloud server computes the encrypted score of a matching document identifier by summing up the encrypted relevance score of this document to each keyword in the query homomorphically, returns the (document identifier-encrypted score) pairs, and then the user decrypts the received scores and sorts the document identifiers by these scores.

(viii) The cloud server returns a list of encrypted concatenations, each consists of a matching document identifier and its score, and then the user decrypts and parses the received concatenations, gets the document identifiers and corresponding scores, and sorts the document identifiers by these scores.

(ix) The cloud server computes the score of a matching document identifier by summing up the relevance score of this document to each keyword in the query, sorts the document identifiers by these scores, and returns them to the user.

(x) The cloud server cumulates all (encrypted document identifier, score) pairs for each keyword, and returns these pairs to user who decrypts each document identifier and adds the related score to the score of the document identifier in order to find the total score of this document to the query. After that, the user sorts the document identifiers by these scores.

In dynamic schemes, the data owner/user can update the index stored on the server after setup phase by adding new (keyword/identifier) pairs or deleting the existing pairs. For this, he/she sends an update token (add/delete) to the server. Then, the server adds the coming pairs to the index or deletes the pairs from the index.

In this thesis, we do not discriminate between the data owner and data user, and we called them as user or client. In such a single-user scenario, the user is assumed to be fully-trusted. Further, as in most of the searchable encryption schemes, we consider the cloud storage server as a single server which provides services for both storage and query, and the cloud server is *honest-but-curious* which means it follows the protocol correctly, however, it may try to learn additional private information from the stored documents, index and search request. Considering this setting, in which there are a single user and single server, our schemes can target individual outsourcing of encrypted documents to a cloud server for their practical usage.

2.2. Security Requirements

The security properties that a ranked searchable encryption scheme should achieve are listed below.

- Document privacy: The server cannot learn anything about the documents given the corresponding encrypted documents. This is satisfied with encrypting the documents using a symmetric encryption scheme.
- Index privacy: The encrypted index should leak nothing about the plaintext information. Some schemes may leak additional information, such as the number of keywords, the number of documents, the number of keyword/document pairs in the document collection (size pattern), and so on.
- Token privacy: The server cannot learn the actual query keyword from the query token since the server does not have the secret key used to generate valid tokens.
- Access pattern: It refers to the search results that are the identifiers of the documents matching the queried keyword.
- Search pattern: It refers to the information about when a search query is repeated.
- Relevance score privacy: The server cannot learn the actual value of the encrypted relevance scores.
- Relevance order privacy: The server cannot infer the relative order of the relevance scores.

Some of these properties, such as access pattern and search pattern, are not satisfied by even most known schemes (Curtmola et al. (2006), Kamara et al. (2012), Cash et al. (2014)) in the literature, and so a leakage function is defined to describe what information is allowed to leak such that the scheme is said to be secure under the controlled leakage. The forward and backward privacy properties are also defined for dynamic schemes by Stefanov et al. (2014). The forward privacy is satisfied if the server does not know whether a new added document includes a keyword that was previously queried, and the backward privacy is satisfied if queries cannot be performed over deleted documents. Our all proposed schemes leak search pattern, and dynamic ones do not achieve forward and backward privacy, both of which are inherited from the base schemes we improve. However, forward privacy can be adapted to them. Additionally, the schemes may leak access pattern, size pattern, the number of keywords in the document set, or the relevance order depending on their properties, that are explained in Chapter 6.

2.3. Design Goals

Following the explained scenario(s), assumptions on the user and server, and security requirements, the design of our system should meet the requirements listed below:

1. Secure ranked single keyword/multi-keyword search: We propose new schemes with improved query functionalities so that the matching document identifiers are ranked using the relevance scores.

2. Efficiency: The computation, communication and storage overhead of the schemes should be acceptable.

3. Security: The schemes should satisfy the security requirements that we state in Section 2.2.

4. Dynamic: The schemes can be designed to support dynamic updates on the index.

On the other hand, there are always different tradeoffs among these goals to design an encrypted search scheme, as mentioned in Kamara (2015).

2.4. Cryptographic Primitives

Basic Primitives: We use symmetric encryption schemes and pseudorandom functions (PRFs) in the construction of the schemes (see Katz and Lindell (2007)). A symmetric encryption scheme consists of three polynomial-time algorithms that are Key Generation (Gen), Encryption (Enc), and Decryption (Dec). Gen and Enc are probabilistic algorithms and Dec is a deterministic algorithm. The properties of the algorithms are:

– Gen takes as input a security parameter k and returns a secret key K.

– Enc takes as input a key K and a message m, and returns a ciphertext c.

– Dec takes as input a key K and a ciphertext c, and returns m provided that c was produced under the key K.

A PRF is a polynomial-time computable function that is computationally indistinguishable from a random function by a probabilistic polynomial-time adversary.

Order Preserving Encryption (OPE): We also make use of OPE in some of the schemes to encrypt the scores. An OPE scheme produces encrypted values that maintain the order of the plaintext values, which means: If $x < y$, then $\text{Enc}_K(x) < \text{Enc}_K(y)$, for any secret key K. Hence, it enables the server to efficiently determine the relative order of the plaintext values without needing them.

The concept of OPE was first introduced by Agrawal et al. (2004) within the database area, and it was first formally analyzed by Boldyreva et al. (2009), giving a security guarantee. After that, a number of OPE schemes were proposed while they leaked most of the plaintext bits. Further schemes were introduced with ideal-security, which do not reveal anything about the plaintexts, except for their order relations (Popa et al. (2013), Kerschbaum and Schroepfer (2014)). A scheme can still leak a significant amount of sensitive information only revealing order relations. To solve this problem, Kerschbaum (2015) presented a frequency-hiding scheme by randomizing ciphertexts, but this scheme also needs more client storage and may lead to some error for queries.

Paillier Encryption: Homomorphic encryption allows operations, such as addition and multiplication, over ciphertexts and produces an encrypted result which equals to the encryption of the result of operations on their plaintexts (Yi et al. (2014)). We use Paillier encryption scheme which consists of three algorithms: Key Generation, Encryption and Decryption. These algorithms are explained in Table 2.1. In this thesis, the following additive homomorphic property of the scheme is exploited, that is:

$$\begin{aligned} \text{Enc}(m_1) \cdot \text{Enc}(m_2) &= (g^{m_1} r_1^n)(g^{m_2} r_2^n)(\bmod n^2). \\ &= g^{m_1+m_2}(r_1 r_2)^n(\bmod n^2). \\ &= \text{Enc}(m_1 + m_2). \end{aligned} \tag{2.1}$$

Table 2.1. Paillier Encryption Scheme.

Key Generation: Select two large primes p and q randomly and independently. Set $n = pq$. Set $\lambda = \text{lcm}(p-1, q-1)$. Choose random $g \in \mathbb{Z}^*_{n^2}$, such that n divides the order of g. $\mu = (L(g^\lambda \pmod{n^2}))^{-1} \pmod{n}$. The public key: (n, g). The private key: (λ, μ).
Encryption: Message $m \in \mathbb{Z}_n$. Choose random $r \in \mathbb{Z}^*_{n^2}$. Ciphertext $c = g^m r^n \pmod{n^2}$.
Decryption: Ciphertext $c \in \mathbb{Z}^*_{n^2}$. Message $m = \dfrac{L(c^\lambda \pmod{n^2})}{(L(g^\lambda \pmod{n^2}))^{-1}} \pmod{n}$, where $L(u) = (u-1)/n$.

2.5. Information Retrieval

Ranking function: A ranking function can be used to compute relevance scores of matching documents to a given search request so that the documents are sorted according to these scores. In some of our ranked schemes, we utilize from TF $\times$ IDF weighting rule, where TF (term frequency) measures the importance of a keyword for a document, and IDF (inverse document frequency) of a keyword measures the importance of the keyword within the entire document collection. The following TF $\times$ IDF formula is applied in the schemes to find the relevance score of the document F_{id} to the keyword w:

$$\text{score}(w, F_{\text{id}}) = \frac{1}{|F_{\text{id}}|}(1 + ln(f_{\text{id},w}))(1 + \frac{N}{f_w}), \tag{2.2}$$

where $|F_{\text{id}}|$ is the length of the document F_{id}; $f_{\text{id},w}$ specifies how many times the keyword w appears in the document F_{id}; N is the total number of documents in the collection; f_w is the number of documents that contain the keyword w.

If a query consists of multiple keywords, then the overall relevance score of each matching document to the query Q is calculated by summing up the relevance scores of the related document to each keyword in Q:

$$\text{score}(Q, F_{\text{id}}) = \sum_{w \in Q} \text{score}(w, F_{\text{id}}). \tag{2.3}$$

2.6. Data Structures

Bloom Filter: A Bloom filter (BF) (Bloom (1970)) is a fast probabilistic data structure that allows to test whether an element is a member of a set in a limited memory space. It is a bit array of length m to represent a set of n elements. It uses k distinct independent hash functions $h_1, ..., h_k$, each of them maps elements to the interval $[1, m]$ with a uniform random distribution. All bits are firstly set to 0. To insert an element x in the set S, the array bits at positions $h_1(x), ..., h_k(x)$ are set to 1. Some bits can be set to 1 multiple times by coincidences for different elements. To test whether an element y belongs to the set S, the array bits corresponding to the positions $h_1(y), ..., h_k(y)$ are checked. If at least one bit is set to 0, then y is definitely not a member of S. However, if all the checked bits are set to 1, then y is a member of the set S with a high probability. It means that there is some probability of a false positive. The false positive probability (fp) can be estimated:

$$fp = (1 - e^{-kn/m})^k. \tag{2.4}$$

CHAPTER 3

RELATED WORK

In this chapter, we present existing literature related to Searchable Symmetric Encryption (SSE) and Ranked Searchable Encryption.

3.1. Searchable Symmetric Encryption

Song et al. (2000) proposed the first SSE scheme to perform keyword searches on encrypted data, in which no index is built and each word in a document is encrypted separately, therefore it requires linear search time with the total number of words in the document collection.

Goh et al. (2003) developed a secure index scheme based on PRFs and BFs in order to improve search performance. A BF represents a set using much less space than other data structures since it just stores the membership of an element rather than the element itself. In the scheme, a BF is built per document, and so each document is searched in $O(1)$ time. So, its search time is linear to the total number of documents in the collection. Tekin and Şahin (2017) improved this scheme using Counting Bloom Filters (CBFs) to update a document without rebuilding its index.

Chang and Mitzenmacher (2005) offered two schemes to do keyword searches on remote encrypted data. The authors constructed the schemes by thinking two cases in which there is or is not enough room to store a dictionary on the user's mobile device. In these schemes, a masked index string is generated per document using pseudorandom bits. Hence, they also require linear search time with the number of the documents.

Curtmola et al. (2006) proposed new security definitions and presented two schemes, namely SSE-1 and SSE-2, which are secure under these new definitions. Specifically, SSE-1 and SSE-2 are secure against *non-adaptive* and *adaptive* adversaries, respectively. According to this paper, all previous SSE schemes are within the non-adaptive setting. An adaptive adversary can choose their search queries by taking into account previous trapdoors and search outcomes. Besides stronger security, these schemes are more efficient than the previous ones since the search computation per keyword is linear to the number of documents that contain the keyword, which means that they provide *sublinear search*. In the literature of SSE, their security definitions and indexing techniques are used in subsequent schemes. (Kamara et al. (2012); Kamara and Papamanthou (2013); Cash et al. (2014); Stefanov et al. (2014); Hahn and Kerschbaum (2014); Naveed et al. (2014); Baldimtsi and Ohrimenko (2015))

The first scheme of Curtmola et al. (2006) creates a linked list for each distinct keyword, and each node in the list keeps a pointer to the next node. Then, all nodes of all lists are stored in an array permuted randomly and encrypted. Also, there is a lookup table to find and decrypt the first node of each list. To succeed their security definition, padding is necessary to the array and the table. Their second scheme also uses a lookup table. In the scheme, for each keyword, different labels are produced up to the number of documents containing the keyword. For instance, if a keyword, say *tea*, is included in three documents, then the labels *tea1*, *tea2*, *tea3* are produced, and a pseudorandom permutation (PRP) is used to generate the keys from the labels. Then, each key and corresponding document identifier is added to the table. However, this scheme also requires padding so that the number of entries in which each document identifier appears is the same. Curtmola et al. (2006) considered secure updating as if only adding new documents, where each time a new index is created for the document set that consists of the old set and a new set of documents (*re-indexing*).

Despite the approach based on the inverted index provided efficient schemes until that time, they were not explicitly dynamic and naturally required sequential searches. Kamara et al. (2012) introduced an extension of the previous scheme SSE-1 to allow dynamic updates efficiently and be secure against adaptive attacks. Nevertheless, their scheme is so hard to implement and leaks a certain amount of information, namely the tokens in an updated document *(update leakage)*.

Kamara and Papamanthou (2013) put forward a new dynamic approach based on keyword red-black (KRB) trees adapted from red-black trees, which supports parallel search and update operations, and prevents the update leakage. In each node of KRB tree, they store two vectors whose lengths are the number of documents and values depend on the output of a random oracle. But, the index size of their scheme is the product of the number of documents and the number of keywords, so it is much larger than that of the previous ones.

Hahn and Kerschbaum (2014) proposed a novel secure and efficient dynamic scheme based on the idea of constructing the index from the access pattern. The search complexity for a new keyword is linear, that amortizes to sub-linear time as repeated searches are performed, while the scheme requires the client to store a search history to be used for the add operation.

Cash et al. (2014) also proposed a dynamic scheme based on the inverted index structure that is more efficient than the previous schemes to handle very large datasets by taking into account ignored factors like locality and parallelism, and has optimal leakage and index size, while it needs either some local storage or communication linear to the size of keywords to be updated.

Stefanov et al. (2014) presented a dynamic scheme by introducing important stronger properties like forward and backward privacy, whereas this scheme has logarithmic search overhead and client storage due to the using oblivious RAM (ORAM) techniques. Subsequent dynamic schemes have been designed to efficiently achieve these properties (Yavuz and Guajardo (2015), Bost et al. (2017), Ozmen et al. (2017)).

3.2. Ranked Searchable Encryption

We present different categorizations of ranked searchable encryption schemes in Table 3.1 and Table 3.2 using the categorizations related to the ranked search in Poh et al. (2017). We try to explain some of them below in detail.

The problem of ranked search on encrypted data was first defined and solved by Wang et al. (2012). Their scheme is based on non-adaptive secure SSE-1 scheme of Curtmola et al. (2006) mentioned in the previous section. They utilize from the TF $\times$ IDF rule to calculate relevance scores, whereas the IDF part is ignored because of supporting only single-keyword searches. They use the OPE with some modifications to encrypt the relevance scores. The scheme concatenates the encrypted relevance scores to the document identifiers and adds to the index. The search operation is efficiently done on the server side. The scheme weakens the security guarantee of their base scheme because the server knows additional information, i.e. the relative order.

Sun et al. (2013) proposed a multi-keyword ranked search scheme that is based on a tree structure and vector space model (VSM) that enables both conjunctive (AND) and disjunctive (OR) search. For each document, an index vector containing normalized tf values is created, and then the vector is split into two random vectors. For a query, a query vector containing normalized idf values is generated, and then it is also split into

Table 3.1. Development of Ranked Searchable Encryption Schemes I. (Source: Poh et al. (2017)).

Scheme & Year	Setups		Query Functionalities				Structures			
	u	s	1w	rnk	cnj	fz	w	di	inv	t
Wang et al. (2012)	1	1	●	●					●	
Ibrahim et al. (2012)	1	2	○	●					●	
Sun et al. (2013)	1	1	○	●						●
Cao et al. (2014)	1	1	○	●				●		
Strizhov and Ray (2014)	1	1	○	●					●	
Xia et al. (2016)	1	1	○	●						●
Baldimtsi and Ohrimenko (2015)	1	2	○	●					●	
Orencik et al. (2016)	1	2	○	●		●			●	
Shen and Zhang (2017)	1	1	○	●	●				●	
Jiang et al. (2017)	1	2	○	●	●				●	

Notation: u = user/multiple users, s = server/multiple servers, 1w = single keyword, rnk = ranked, cnj = conjunctive, fz = fuzzy, w = word/block based, di = direct, inv = inverted, t = tree. ○ = A scheme is multi-keyword, but can be used for single-keyword search. ● = A scheme is single-keyword, but was extended to multi-keyword search.

two random vectors. The ranking of documents is achieved by calculating the cosine similarity values. To improve search efficiency, they use a tree-based search algorithm. Phantom terms can be added to the query in order to increase privacy, but it may lead to some precision loss.

Cao et al. (2014) offered a scheme with ranking functionality to perform multi-keyword searches. In this scheme, a ”secure inner product” technique is used to sort the documents. Each document score is found by counting the number of query keywords included in the document. Adding dummy keywords to data vectors can result in some precision loss by returning false results (positives or negatives), while it increases rank privacy, so there is a tradeoff between precision and rank privacy and this scheme provides a balance parameter. But, the search overhead of this scheme is very high for the client.

Strizhov and Ray (2014) proposed a scheme for multi-keyword similarity searches by using dot product technique. The scheme is based on the SSE-2 scheme of Curtmola et al. (2006). In addition to the inverted index, a TF-IDF table is built, and encrypted with a fully homomorphic encryption (FHE) scheme. The server only locates the identifiers and returns them along with encrypted scores to the user who performs decryption and

Table 3.2. Development of Ranked Searchable Encryption Schemes II. (Source: Poh et al. (2017)).

Scheme & Year	Characteristics			Prim. (PRF, PRP, Enc.)	Security Model		
	sta	dyn	pt	+others	Ind	S1	S2
Wang et al. (2012)	●			● (OPE)			
Ibrahim et al. (2012)	●			● (PPM)			
Sun et al. (2013)			●				
Cao et al. (2014)	●						
Strizhov and Ray (2014)	●			● (Hom. enc.)			●
Xia et al. (2016)		●					
Baldimtsi and Ohrimenko (2015)	●			● (Hom. enc.)			●
Orencik et al. (2016)	●		●	● (Hom. enc.)			●
Shen and Zhang (2017)	●			● (OPE)			●
Jiang et al. (2017)	●			● (Hom. enc.)			●

Notation: sta = static, dyn = dynamic, pt = probabilistic tokens, ind = indistinguishability-based, S1 = non-adaptive, S2 = adaptive.

sorting. In this scheme, the server is not concerned with sorting. For this reason, this scheme is very inefficient despite of its sublinear search.

Baldimtsi and Ohrimenko (2015) proposed a solution for private sorting. In their solution, a secure co-processor helps the server to sort the scores encrypted with Paillier encryption scheme. Both the server and the co-processor do not learn the relative order of search results. Even though this approach is very attractive because of maintaining privacy, it is not practicable to deploy.

Shen and Zhang (2017) extended the SSE protocol OXT (Cash et al. (2013)) so that results of conjunctive queries are ranked on the server side by keyword occurrences which are encrypted using OPE. Like the OXT, the search complexity of this scheme is linear in the number of documents containing the least frequent keyword in the query.

CHAPTER 4

BASE SCHEMES

In this chapter, we explain the base schemes that we improve to support efficient single- and multi-keyword ranked searches. Each scheme has a form of response-revealing or response-hiding. The first form reveals the response of the queries, while the second one does not. We follow an example inverted index illustrated in Figure 4.1 throughout the thesis to understand the structures of the schemes more clearly.

Label	Value
w_1	D_2, D_{10}
w_2	D_4, D_5, D_{10}
w_5	$D_1, D_3, D_5, D_6, D_{10}$
w_7	$D_1, D_2, D_3, D_5, D_6, D_9, D_{10}$

Figure 4.1. An example inverted index.

4.1. RR2Lev Scheme

Before explaining the RR2Lev scheme, first we will describe a basic scheme, and then extend it to the this scheme. In the setup of the basic scheme, a key K is chosen. For each keyword, the key is used to generate two keys K_1 and K_2 which are for a PRF and encryption. That means, two keys are formed per keyword in order to obtain pseudorandom labels and encrypt the document identifiers. After creating the keys, the document identifiers with this keyword are iterated. A label is associated with each document identifier by applying the PRF and the key K_1 to a keyword-specific counter that starts zero and increments by one, and the identifier is encrypted with the key K_2, and then the label/encrypted identifier pair is added to the dictionary, as seen in Figure 4.2. For example, if a keyword is contained in 3 documents, then 3 independent labels are produced for the keyword. To search for a keyword, the user generates the keys K_1 and K_2 for the searched keyword and sends to the server which computes the corresponding labels using the K_1 and PRF, retrieves the related entries from the dictionary, and decrypts them using the K_2 to get the identifiers. Hence, for example, if the keyword w_2 is searched, the server retrieves the $E(K_2^2, D_4), E(K_2^2, D_5)$ and $E(K_2^2, D_{10})$ from the dictionary, and gets D_4, D_5 and D_{10} by decrypting them.

Label	Value
$F(K_1^1, 0)$	$E(K_2^1, D_2)$
$F(K_1^1, 1)$	$E(K_2^1, D_{10})$
$F(K_1^2, 0)$	$E(K_2^2, D_4)$
$F(K_1^2, 1)$	$E(K_2^2, D_5)$
$F(K_1^2, 2)$	$E(K_2^2, D_{10})$
$F(K_1^5, 0)$	$E(K_2^5, D_1)$
$F(K_1^5, 1)$	$E(K_2^5, D_3)$
$F(K_1^5, 2)$	$E(K_2^5, D_5)$
$F(K_1^5, 3)$	$E(K_2^5, D_6)$
$F(K_1^5, 4)$	$E(K_2^5, D_{10})$
$\ldots$	$\ldots$

Figure 4.2. The basic scheme index created for the given example inverted index. Superscripts are used to distinguish the keys of different keywords.

A keyword query with r matches in the basic scheme requires r retrievals from the dictionary which is asymptotically optimal with the assumption of a dictionary retrieval as $O(1)$ cost. But, if the dictionary is kept on disk, the query time may be slow since the identifiers are stored in pseudorandom positions, and each retrieval from the dictionary results in a disk read. To decrease the number of retrievals, a certain amount of the identifiers are packed together, which means that a block size B is determined and B identifiers are packed and encrypted, and the packed ciphertext is inserted with a label, rather than adding each encrypted identifier separately. The last block of a keyword is padded to be the same size.

However, choosing a larger block size may cause too much padding in case there are many keywords matched with few documents. The solution for this is that storing the encrypted blocks of B identifiers in an array and encrypted blocks of b pointers to these blocks in the dictionary, where b and B are the block sizes of the dictionary and array.

Additionally, real datasets can have great variability in the number of matching results for different keywords. It means that there may be many keywords included in only a few documents, and there may be some keywords contained in a significant number of documents. The 2Lev scheme takes this situation into account and offers a solution to address this variability within datasets. In this thesis, we call it RR2Lev or RH2Lev depending on revealing or hiding query responses. We explain RR2Lev in this section and RH2Lev in the next section.

RR2Lev considers the result sets of keywords as (i) small, (ii) medium, or (iii) large. The result set of a keyword w, $\mathrm{DB}(w)$, contains the identifiers the keyword appears in. For simplicity, without considering encryptions, it can be said that it inserts the identifiers as follows: If the result set is small ($|\mathrm{DB}(w)| \leq b$), a block of b document identifiers are packed and stored in the dictionary. If the result set is medium ($b < |\mathrm{DB}(w)| \leq Bb$), the blocks of B identifiers are stored in the array and a block of b pointers to these blocks is stored in the dictionary. If the result set is large ($Bb < |\mathrm{DB}(w)| \leq B^2b$), a block of b pointers is stored in the dictionary, and these pointers point to the blocks of B pointers in the array, where each pointer points to the blocks of B identifiers in the array. So, the scheme uses two levels of indirections to access the identifiers for the large result sets. To search for a keyword, the user sends the keys K_1 and K_2 like the basic scheme. The server computes the label by applying the PRF to zero with K_1, accesses the corresponding value from the dictionary, and decrypts it using K_2. If the decrypted value contains the identifiers, then the server outputs them. Otherwise, the value contains the pointers and the server accesses the blocks in the array using these pointers. If those blocks include identifiers, then the server outputs the identifiers. Otherwise, the server uses the second-level pointers to access and output the identifiers.

Now, we present the dictionary and array created for the given example inverted index in Figure 4.3. Here, superscripts are removed for simplicity, however two keys are created for each keyword, as in the basic scheme. The chosen sizes for small (b) and big blocks (B) are 2 and 3, respectively. The keyword w_1 is included in 2 documents which are D_2 and D_{10}. Since its result set is small, the two identifiers are packed, the encryption of it is directly stored in the dictionary. The result sets of the keywords w_2 and w_5 are medium. The keyword w_5 is contained in 5 documents that are D_1, D_3, D_5, D_6 and D_{10}.

The identifier blocks [D_1, D_3, D_5] and [D_6, D_{10}, X] are randomly stored in the positions 67 and 20 of the array. So, the block [67, 20] is stored in the dictionary. The keyword w_7 appears in 7 documents, so its set size is large. The identifiers are separated in the blocks [D_1, D_2, D_3], [D_5, D_6, D_9] and [D_{10}, X, X]. These blocks are inserted into the positions 90, 14 and 85 in the array. The block of the pointers [90, 14, 85] is inserted in the position 6 of the array, and the block [6, X] is kept in the dictionary. If the keyword w_7 is searched, after taking the related keys from the user, the server gets the $E(K_2, 6 \parallel X)$ from the dictionary and decrypts it. Then, it accesses the position 6 in the array, gets $E(K_2, 90 \parallel 14 \parallel 85)$ and decrypts it. Next, the server accesses the positions 90, 14 and 85 in the array, and outputs a list of the identifiers $D_1, D_2, D_3, D_5, D_6, D_9$ and D_{10} after parsing and removing the padded elements.

Label	Value
$F(K_1, 0)$	$E(K_2, D_2 \parallel D_{10})$
$F(K_1, 0)$	$E(K_2, 17 \parallel X)$
$F(K_1, 0)$	$E(K_2, 67 \parallel 20)$
$F(K_1, 0)$	$E(K_2, 6 \parallel X)$

	$\ldots$
6	$E(K_2, 90 \parallel 14 \parallel 85)$
	$\ldots$
14	$E(K_2, D_5 \parallel D_6 \parallel D_9)$
	$\ldots$
17	$E(K_2, D_4 \parallel D_5 \parallel D_{10})$
	$\ldots$
20	$E(K_2, D_6 \parallel D_{10} \parallel X)$
	$\ldots$
67	$E(K_2, D_1 \parallel D_3 \parallel D_5)$
	$\ldots$
85	$E(K_2, D_{10} \parallel X \parallel X)$
	$\ldots$
90	$E(K_2, D_1 \parallel D_2 \parallel D_3)$
	$\ldots$

Figure 4.3. The RR2Lev scheme dictionary and array created for the given example inverted index. The chosen sizes for small and big blocks are 2 and 3, respectively.

4.2. RH2Lev Scheme

Response hiding schemes do not reveal the identifiers of query results to the server. To achieve this, the identifiers are encrypted using a symmetric encryption scheme, such as AES, and stored in the index instead of being stored in plaintext. So, the server outputs a list of the encrypted identifiers for a keyword search and the user decrypts the returned identifiers. In the RH2Lev scheme, an extra key K_3 is derived from the key K to encrypt the document identifiers. Therefore, an additional algorithm (resolve) is introduced, which is executed by the user to decrypt the identifiers.

The RH2Lev scheme dictionary and array created for the given index are shown in Figure 4.4. For example, if the keyword w_2 is queried, the user sends the keys K_1 and K_2 of the w_2 to the server which gets the $E(K_2, 17 \parallel X)$ from the dictionary with the K_1 and decrypts it using the K_2. Then, the server takes the element $E(K_2, E(K_3, D_4) \parallel E(K_3, D_5) \parallel E(K_3, D_{10}))$ of the array via the pointer 17 and gets a list consisting of $E(K_3, D_4), E(K_3, D_5)$ and $E(K_3, D_{10})$ by decrypting the outer encryption using the K_2 and parsing. After that, the server returns the list to the user who decrypts the encrypted identifiers with the key K_3 by running the resolve algorithm.

4.3. DynRR Scheme

Kamara and Moataz (2017) introduced a dynamic response-hiding SSE (DynRH) in the Clusion framework, which is a variant of the dynamic scheme (Cash et al. (2014)). We investigate the two types of this scheme by also adapting it to the response-revealing form (DynRR). We examine the DynRR scheme in this section and the DynRH scheme in the following section. However, prior to the DynRR scheme, we describe the dynamic scheme of Cash et al. (2014).

The dynamic property supports additions and deletions to the data after uploading it to the server. So, the basic scheme in the Section 4.1 is modified in order to allow dynamism. In the dynamic scheme, to enable the additions, an extra empty dictionary is used in addition to the static dictionary. Keyword-document pairs are added to this dictionary with addition operations. To add a document identifier for a keyword, the label is computed from the keyword and counter value specific to this keyword, and the identifier is encrypted. It means that addition operations containing keyword w need the user to know

Label	Value
$F(K_1, 0)$	$E(K_2, E(K_3, D_2) \parallel E(K_3, D_{10}))$
$F(K_1, 0)$	$E(K_2, 17 \parallel X)$
$F(K_1, 0)$	$E(K_2, 67 \parallel 20)$
$F(K_1, 0)$	$E(K_2, 6 \parallel X)$

	$\dots$
6	$E(K_2, 90 \parallel 14 \parallel 85)$
	$\dots$
14	$E(K_2, E(K_3, D_5) \parallel E(K_3, D_6) \parallel E(K_3, D_9))$
	$\dots$
17	$E(K_2, E(K_3, D_4) \parallel E(K_3, D_5) \parallel E(K_3, D_{10}))$
	$\dots$
20	$E(K_2, E(K_3, D_6) \parallel E(K_3, D_{10}) \parallel X)$
	$\dots$
67	$E(K_2, E(K_3, D_1) \parallel E(K_3, D_3) \parallel E(K_3, D_5))$
	$\dots$
85	$E(K_2, E(K_3, D_{10}) \parallel X \parallel X)$
	$\dots$
90	$E(K_2, E(K_3, D_1) \parallel E(K_3, D_2) \parallel E(K_3, D_3))$
	$\dots$

Figure 4.4. The RH2Lev scheme dictionary and array created for the given example inverted index. The chosen sizes for small and big blocks are 2 and 3, respectively.

the current value of the counter for the w. Therefore, a dictionary $\mathrm{D}_{\mathrm{count}}$ mapping a keyword to its current value is stored at the client or server, that requires keeping a state at the client or communication whose size is proportional to the number of keywords to be added or deleted if it is stored in an encrypted form at the server. To enable deletions, the server maintains a revocation list storing pseudorandom revocation identifiers each of which is computed from a deleted keyword/document pair so that the server can filter out the results of a query that are deleted. So, the user sends also another key for this to the server, that allows the server to filter out the deleted ones. Nevertheless, it makes addition operations more complicated since the server should delete the corresponding revocation identifiers from the revocation list when deleted pairs are re-added and the user increments the counter values according to the response of the server.

Now, we explain DynRR scheme. The setup algorithm of this scheme chooses a key K and allocates two dictionaries D and D$_{\text{count}}$, and a list P. The dictionary D$_{\text{count}}$ is stored at the user as state. The dictionary D and list P are stored at the server. The P is used to hold the counter values corresponding to the document identifiers to be deleted that match a keyword. When the user wants to add keyword-document pairs, the following steps are performed: For each keyword, two keyword-specific keys, K_1 and K_2, are derived. All the identifiers associated with the keyword are iterated. For each identifier, the current counter value for the keyword is taken from D$_{\text{count}}$ (zero if null), a label is produced by applying the PRF and the key K_1 to the counter value, and the identifier is encrypted with the key K_2. Then, the label/encrypted identifier pair is added to the list that will be sent to the server which adds the elements in the list to the dictionary D by running the Update algorithm. The Search algorithm is similar to that in the basic scheme, but the server also adds the counter values of the document identifiers matching the searched keyword into the list P so that they can be used during deletion. After searching a keyword, if the user wants to delete some of the identifiers for this keyword, the user produces the key K_1, selects the indices of the identifiers to be deleted from the returned result list, and transmits them to the server. Then, the server gets the counter values of the indices from the list P, produces the relevant labels via K_1 and the counter values, and deletes the related entries from the dictionary.

Let us think that the dictionary in Figure 4.5a was created for the document set of the user. If the user searches the keyword w_5, the server calculates [D_1, D_3, D_5, D_6, D_{10}] as the identifier list and [0, 1, 2, 3, 4] as the list P, and returns the identifier list to the user. After that, if the user wants to delete the identifiers D_3 and D_6 for the keyword w_5, the user chooses the indices 1 and 3, and transmits them to the server that finds the counter values 1 and 3 by getting these indices of the list P, produces the labels $F(K_1, 1)$ and $F(K_1, 3)$, and deletes them (see Figure 4.5b). If the user searches the keyword w_5 again, now the returned list is [D_1, D_5, D_{10}] and the list P is [0, 2, 4] since the P is cleared with each search operation. The user selects the index 2 to delete the identifier D_{10} for the keyword w_5 and sends it to the server. Then, the server gets the index 2 of the list P which is 4, and calculates the label $F(K_1, 4)$, and deletes the entry with this label (see Figure 4.5c).

Label	Value
$F(K_1, 0)$	$E(K_2, D_2)$
$F(K_1, 1)$	$E(K_2, D_{10})$
$F(K_1, 0)$	$E(K_2, D_4)$
$F(K_1, 1)$	$E(K_2, D_5)$
$F(K_1, 2)$	$E(K_2, D_{10})$
$F(K_1, 0)$	$E(K_2, D_1)$
$F(K_1, 1)$	$E(K_2, D_3)$
$F(K_1, 2)$	$E(K_2, D_5)$
$F(K_1, 3)$	$E(K_2, D_6)$
$F(K_1, 4)$	$E(K_2, D_{10})$
$\ldots$	$\ldots$

(a) The DynRR dictionary created for the given index. If the keyword w_5 is searched, the server locates the entries colored in gray, and the list P keeps $[0, 1, 2, 3, 4]$.

Label	Value
$F(K_1, 0)$	$E(K_2, D_2)$
$F(K_1, 1)$	$E(K_2, D_{10})$
$F(K_1, 0)$	$E(K_2, D_4)$
$F(K_1, 1)$	$E(K_2, D_5)$
$F(K_1, 2)$	$E(K_2, D_{10})$
$F(K_1, 0)$	$E(K_2, D_1)$
$F(K_1, 2)$	$E(K_2, D_5)$
$F(K_1, 4)$	$E(K_2, D_{10})$
$\ldots$	$\ldots$

(b) The DynRR dictionary created for the given index after deleting the identifiers D_3 and D_6. If the keyword w_5 is searched again, now the server locates gray entries and the list P keeps $[0, 2, 4]$.

Label	Value
$F(K_1, 0)$	$E(K_2, D_2)$
$F(K_1, 1)$	$E(K_2, D_{10})$
$F(K_1, 0)$	$E(K_2, D_4)$
$F(K_1, 1)$	$E(K_2, D_5)$
$F(K_1, 2)$	$E(K_2, D_{10})$
$F(K_1, 0)$	$E(K_2, D_1)$
$F(K_1, 2)$	$E(K_2, D_5)$
$\ldots$	$\ldots$

(c) The dictionary created for the given index after deleting the identifier D_{10}.

Figure 4.5. The DynRR scheme dictionary created for the given example inverted index before and after delete operations.

4.4. DynRH Scheme

DynRH scheme encrypts the document identifiers using a key of the user before storing them into the dictionary. So, it does not reveal the identifiers in plaintext to the server for a keyword search. As in the RH2Lev scheme, a resolve algorithm is introduced. The DynRH scheme dictionary created for the given index is shown in Figure 4.6.

Label	Value
$F(K_1, 0)$	$E(K_2, E(K_3, D_2))$
$F(K_1, 1)$	$E(K_2, E(K_3, D_{10}))$
$F(K_1, 0)$	$E(K_2, E(K_3, D_4))$
$F(K_1, 1)$	$E(K_2, E(K_3, D_5))$
$F(K_1, 2)$	$E(K_2, E(K_3, D_{10}))$
$\ldots$	$\ldots$

Figure 4.6. The DynRH scheme dictionary created for the given example inverted index.

4.5. SRMES

Ibrahim et al. (2012) proposed a secure ranked scheme (SRMES) to allow multi-keyword searches over encrypted documents stored on remote servers. In this scheme, an inverted index mapping each keyword to a posting list is created for the document collection and a score is associated with each keyword-document pair to achieve the ranking capability. They protect the inverted index by encrypting the keyword set with a key-based hash function and the document identifiers with Paillier cryptosystem, padding the posting lists to the longest list length with random identifier and zero score pairs, inserting faked keywords with faked posting lists, and using an enhanced privacy preserving mapping (PPM) scheme in which the scores are encrypted with probabilistic Paillier cryptosystem so that the duplicated scores are seen as different scores.

In the PPM scheme, integer numbers are mapped into encrypted images that can be used later for comparisons to decide the relations like $<, =, >$ for the corresponding numbers. For a set $X = \{x_1, ..., x_r\}$ consisting of r different integer numbers, the mapping list is created as follows:

$$Mlist = \{H2(H1_k(x_j) \parallel H1_k(x_i)) \mid x_i < x_j \wedge 1 \leq i, j \leq r\}, \tag{4.1}$$

where $T_i = H1_k(x_i)$ is the encrypted image of the integer x_i, and $H2$, $Mlist$ and the images are public parameters. Comparison of two images T_1 and T_2 in the mapping list gives one of the results $(0, 1$, or $-1)$ according to the Algorithm 1.

Algorithm 1 Comparison of two images (Ibrahim et al. (2012))

```
Input: The secret images $T_1$ and $T_2$;
Output: $0, 1$ or $-1$.

1: if $T_1 = T_2$, return $0$.
2: else if $H2(T_2 \parallel T_1) \in Mlist$, return $-1$.
3: else, return $1$.
```

The $Mlist$ stores $r * (r - 1)/2$ hash values so its size can be very large when the r is large. Therefore, they propose to divide the set X into subsets X_i (a set for each posting list), create an $Mlist_i$ for each subset X_i, and use a BF for representing the $Mlist_i$ in order to improve the efficiency of the PPM scheme. With this setting, for each keyword, a BF is created from the secret images of the relevance scores in the posting list associated with the keyword, and two images can be compared using the BF, which can be seen from the Algorithm 2.

Algorithm 2 CBF (Ibrahim et al. (2012))

```
Input: The secret images $T_1$ and $T_2$, and a Bloom Filter $BF$;
Output: 0, 1 or -1.

1: if $T_1 = T_2$, return $0$.
2: else if: $BF(f_l(H2(T_2 \parallel T_1))) = 1, l = 1, ..., t$, return $-1$.    ▷ $t$ hash functions
3: else, return $1$.
```

They introduce two distinct cloud servers to keep the encrypted documents and the index with the aim of hiding the access pattern. Specifically, the servers are the search server, SS and the document server, DS. Their assumption is that the servers do not collude with each other. The scheme consists of two phases that are indexing and searching phase. In the first phase, the data owner creates the index and converts it to the secure index including the posting list (encrypted identifier-encrypted score pairs) along with the BF for each keyword and encrypts the documents. In the retrieving phase, an authorized user sends the trapdoor for a multi-keyword query. The SS finds the corresponding posting lists along with BFs and calls the Ranking Algorithm (see Algorithm 3) to rank the document identifiers in each posting list. The SS combines all FileID (the encrypted identifiers) and PScore (the partial scores), and sends them to the DS which has the Paillier decryption key to decrypt the encrypted identifiers. Then, the DS calculates the scores of each identifier from the partial scores using the Equation 2.3 and returns the top-k encrypted documents to the user.

Algorithm 3 Ranking (Ibrahim et al. (2012))

```
Input: Posting List $PL$ of length $L$ and Bloom Filter $BF$;
Output: FileID and PScore

 1: Set FileID to the encrypted document identifiers in $PL$.
 2: Set Escore to the encrypted scores in $PL$.
 3: Initialize $L * L$ matrix M of zeros.
 4: for $i = 1$ to $L$ do:
 5:     for $j = 1$ to $L$ do:
 6:         if $i < j$ :
 7:             M$(i, j)$ = CBF($BF$, Escore($i$), Escore($j$))
 8:         else:
 9:             M$(i, j) = -$ M$(j, i)$
10: Set PScore to $\{sm_1, ..., sm_L\}$, where $sm_j$ is the summation of row $j$ in M
11: Output FileID and PScore.
    ▷ PScore is the set of partial scores.
```

The ranking algorithm computes the partial score of each document in a posting list by comparing *each pair* of the encrypted scores in the posting list. Thus, filling the matrix M has a high impact in the algorithm. For example, if a keyword is included in 7 documents, the length of the posting list for this keyword is 7 ($L = 7$). Each pair in the list is compared so the CBF algorithm is called $(7 * 6)/2 = 21$ times. Assume that the number of hash functions used in the Bloom Filter is 5. Then, $21 * 5 = 105$ hash functions are computed during the search.

Let us think a single-keyword query as an example to show how the matrix is calculated. Suppose the posting list for the keyword is $PL = \{(1, 2)\ (2, 6)\ (3, 4)\ (5, 4)\ (6, 12)\ (9, 6)\ (10, 3)\}$ in which the first and second numbers show the identifiers and scores, respectively. For simplicity, the list includes *plaintext* identifiers and scores. The FileID is $\{1, 2, 3, 5, 6, 9, 10\}$ and Escore is $\{2, 6, 4, 4, 12, 6, 3\}$. The SS compares each pair of scores in Escore. For instance, the SS inserts 1 to the M$(1, 2)$ since $H2(6 \parallel 2)$ is contained in the BF (6 is greater than 2), but the SS inserts -1 to the M$(2, 3)$ since $H2(6 \parallel 4)$ is contained in the BF. When the SS completes to compare each pair of the scores, the calculated matrix will be as shown in Figure 4.7 and the partial score list PScore will be $\{6, -3, 1, 1, -6, -3, 4\}$ in which the document with the partial score -6 is the most relevant to the search keyword.

	2	6	4	4	12	6	3	
2	0	1	1	1	1	1	1	$sm_1 =$ 6
6	-1	0	-1	-1	1	0	-1	$sm_2 =$ -3
4	-1	1	0	0	1	1	-1	$sm_3 =$ 1
4	-1	1	0	0	1	1	-1	$sm_4 =$ 1
12	-1	-1	-1	-1	0	-1	-1	$sm_5 =$ -6
6	-1	0	-1	-1	1	0	-1	$sm_6 =$ -3
3	-1	1	1	1	1	1	0	$sm_7 =$ 4

Figure 4.7. The matrix created in the Ranking Algorithm for an example single-keyword query associated with the 7 document identifiers.

CHAPTER 5

RANKED SEARCHABLE ENCRYPTION

We propose different approaches for ranked searchable encryption, which are (i) Sorted, (ii) OPE-Based, (iii) Paillier-Based, (iv) Embedded, and (v) Matrix-Based. The base schemes RR2Lev, RH2Lev, DynRR and DynRH explained in the Chapter 4 only support single-keyword queries without result ranking. We enhance these four schemes to enable result ranking for single- and multi-keyword searches according to these approaches. The scheme SRMES allows multi-keyword ranked searches, however we modify this static scheme to make it more efficient during searches. Table 5.1 shows which base scheme can be adapted to which approach.

Table 5.1. Proposed approaches for Ranked Searchable Encryption.

Query Type	**Base Scheme**	**Approach**				
		Sorted	**OPE**	**Paillier**	**Embedded**	**Matrix**
Single	RR2Lev	✓	✓	✓		
	RH2Lev	✓	✓	✓		
	DynRR		✓	✓		
	DynRH		✓	✓		
Multi	RR2Lev			✓		✓
	RH2Lev				✓	✓
	DynRR			✓		
	DynRH				✓	

We apply different techniques for calculation of relevance scores, that are:

1. If a ranked scheme is static and supports single keyword searches, then the score of a document to a keyword is calculated using $\text{score}(w, F_{\text{id}}) = \frac{1}{|F_{\text{id}}|}(1 + ln(f_{\text{id},w}))$. It ignores the IDF part of the Equation 2.2 because only one keyword is searched.

2. If a ranked scheme is static and supports multi-keyword searches, then the score of a document to each keyword in a query is calculated using the Equation 2.2.
3. If a ranked scheme is dynamic, then the score of a document to a keyword is directly assigned to the keyword frequency, so score$(w, F_{\text{id}}) = f_{\text{id},w}$.
4. If a ranked scheme is multi-keyword, the score of a document to a query is calculated using the Equation 2.3.

We utilize from the frequency values of the keywords in the documents in Table 5.2 as an example for the ranked schemes.

Table 5.2. The frequency values of the keywords in the documents for the given index.

	D_1	D_2	D_3	D_4	D_5	D_6	D_9	D_{10}
w_1	0	5	0	0	0	0	0	2
w_2	0	0	0	3	4	0	0	8
w_5	5	0	20	0	3	7	0	12
w_7	2	6	4	0	4	12	6	3

These approaches are based on the scenarios in which there are a single user and server, as described in the Chapter 2. In the following sections, we will explain the details of the approaches and their schemes in more detail.

5.1. Sorted Ranked SE Schemes

This approach supports only single-keyword searches and keeps the document identifiers matching a keyword in the structures in descending order of relevance scores. Therefore, in setup phase, it performs sort operation before adding the identifiers of a keyword to the structures. It means that this approach does not store the relevance scores in the structures and allows only static schemes like RR2Lev and RH2Lev because dynamic schemes can dynamically add an identifier with a lower relevance score to the dictionary after an identifier with a higher score, and do not preserve the order correctly. So, Sorted SR-RR2Lev and Sorted SR-RH2Lev schemes are proposed for this approach.

5.2. OPE-Based Ranked SE Schemes

To enable ranked search, we design the schemes in which a relevance score information is stored for each keyword-document pair. We concatenate each document identifier with a relevance score because they are related to each other. In this approach, relevance scores are first encrypted with OPE scheme and then combined with the identifiers. Additionally, random scores with the same length are added to the pointers and padded elements$-X$ in order to hide the block sizes of the dictionary and array in RR2Lev and RH2Lev schemes. As an example, Figure 5.1 illustrates the modified structures for RR2Lev, created for the given example index with the frequency values in Table 5.2.

During search operations, the server follows the similar steps of the base schemes. However, this time the server parses the concatenated strings to get the identifiers or pointers by removing the padded elements with random scores or directly random scores in case of the pointers. In this manner, the server obtains the identifiers and corresponding encrypted scores. Then, the server sorts the encrypted scores thanks to the OPE and learns the orders of the identifiers. Next, the server returns the ranked document identifiers for the searched keyword to the user. If a response-hiding scheme is used, the server returns the encrypted identifiers and the user should call the resolve algorithm to get the identifiers in plaintext. In dynamic schemes, if the user wants to delete some document identifiers for the keyword, because of the ranking, now the indices selected by the user will delete wrong document identifiers since the server uses the indices to get the counter values from the list P in DynRR and DynRH schemes. Therefore, we use a dictionary P instead of the list P, that is stored at the server and filled during search to map a document identifier (or an encrypted identifier) to its counter value that is used at the time the identifier was added for this keyword. Hence, when the user chooses the indices, the identifiers (or encrypted identifiers) corresponding to these indices are sent to the server which can get the counter values from the dictionary P and delete the correct entries.

OPE-Based SR-RR2Lev, OPE-Based SR-RH2Lev, OPE-Based SR-DynRR and OPE-Based SR-DynRH schemes are introduced for secure single keyword ranked searches. For this approach, multi-keyword schemes are not designed because we consider multi-

Label	Value
$F(K_1, 0)$	$E(K_2, \langle D_2, E(K_{ope}, 130)\rangle \parallel \langle D_{10}, E(K_{ope}, 42)\rangle)$
$F(K_1, 0)$	$E(K_2, \langle 17, R\rangle \parallel \langle X, R\rangle)$
$F(K_1, 0)$	$E(K_2, \langle 67, R\rangle \parallel \langle 20, R\rangle)$
$F(K_1, 0)$	$E(K_2, \langle 6, R\rangle \parallel \langle X, R\rangle)$

	$\dots$
6	$E(K_2, \langle 90, R\rangle \parallel \langle 14, R\rangle \parallel \langle 85, R\rangle)$
	$\dots$
14	$E(K_2, \langle D_5, E(K_{ope}, 79)\rangle \parallel \langle D_6, E(K_{ope}, 174)\rangle \parallel \langle D_9, E(K_{ope}, 279)\rangle)$
	$\dots$
17	$E(K_2, \langle D_4, E(K_{ope}, 209)\rangle \parallel \langle D_5, E(K_{ope}, 79)\rangle \parallel \langle D_{10}, E(K_{ope}, 76)\rangle)$
	$\dots$
20	$E(K_2, \langle D_6, E(K_{ope}, 147)\rangle \parallel \langle D_{10}, E(K_{ope}, 87)\rangle \parallel \langle X, R\rangle)$
	$\dots$
67	$E(K_2, \langle D_1, E(K_{ope}, 130)\rangle \parallel \langle D_3, E(K_{ope}, 199)\rangle \parallel \langle D_5, E(K_{ope}, 69)\rangle)$
	$\dots$
85	$E(K_2, \langle D_{10}, E(K_{ope}, 52)\rangle \parallel \langle X, R\rangle \parallel \langle X, R\rangle)$
	$\dots$
90	$E(K_2, \langle D_1, E(K_{ope}, 84)\rangle \parallel \langle D_2, E(K_{ope}, 139)\rangle \parallel \langle D_3, E(K_{ope}, 119)\rangle)$
	$\dots$

Figure 5.1. The OPE-Based SR-RR2Lev scheme dictionary and array created for the given example inverted index. The chosen sizes for small and big blocks are 2 and 3, respectively. R is a random value.

keyword ranked schemes as follows: if a query consists of multiple keywords, then the relevance score of a document to the query can be calculated by adding the relevance score of the document to each keyword in the query, as in the Equation 2.3. However, OPE schemes cannot be applicable to this setting. Let us explain the reason with an example. Assume that there are 2 documents whose identifiers are id_1 and id_2, and a multi-keyword query is composed of 2 keywords ($w1$ and $w2$). The document id_1 includes both of them and its scores for them are 3 and 15, respectively, and the document id_2 includes only $w2$ and its score for $w2$ is 19. The total scores of id_1 is 18 and the total score of id_2 is 19.

Hence, it can be said that id_2 is more relevant to the query than id_1. However, in OPE schemes, summation of encryptions of 3 and 15 may be greater than encryption of 19.

5.3. Paillier-Based Ranked SE Schemes

This approach also concatenates the relevance scores with the identifiers, as in our OPE-Based approach. However, the scores are encrypted using Paillier encryption scheme rather than OPE scheme. The search algorithm is similar to that in OPE-Based schemes, but the server cannot rank the document identifiers. That is why, the encrypted scores are sent to the user along with the (encrypted) identifiers. A resolve algorithm is introduced in Paillier-Based ranked schemes to decrypt the encrypted scores and rank the (encrypted) identifiers by these scores. If a response hiding scheme is used, then the encrypted identifiers are also decrypted in the resolve algorithm. In dynamic schemes, the server stores a dictionary P rather than a list P to enable deletions like OPE-Based ones. But, there is a small difference in only dynamic response-hiding scheme (DynRH) since the server does not know the identifiers in plaintext and also the order due to the ranking on the user side. A solution for this is that this time the dictionary P maps the encrypted identifiers to the counters, and when the user selects the indices to be deleted, the identifiers of these indices are encrypted at the user before calling the delete algorithm of the server.

We propose Paillier-Based SR-RR2Lev, Paillier-Based SR-RH2Lev, Paillier-Based SR-DynRR and Paillier-Based SR-DynRH schemes to be used for secure single-keyword ranked searches. Also, we introduce Paillier-Based MR-RR2Lev and Paillier-Based MR-DynRR for multi-keyword ranked searches. These schemes modify some algorithms of the single keyword schemes. The token algorithm of the multi-keyword schemes produces a list of tokens for multiple keywords instead of just one token for a single keyword. The search algorithm finds the encrypted total score of each document identifier matching the query by adding the encrypted score of the identifier with respect to each keyword in the query homomorphically. For this, the server stores a dictionary that associates the identifier with the corresponding encrypted score. Then, the server returns the dictionary to the user. The resolve algorithm of the user takes as input this result dictionary and the Paillier decryption key, gets each encrypted score from the dictionary, decrypts it using the key, and at the end, ranks the identifiers by these scores.

The response-hiding schemes cannot be adapted to multi-keyword ranked searches when the relevance scores are encrypted using Paillier scheme and there are a single user and server in the system. The reason is that the server computes the total encrypted score of each document identifier in the response-revealing ones, whereas the response-hiding schemes store the encrypted identifiers and the server sees each of them as if a different identifier because of using the keyword-specific keys. Therefore, the server cannot calculate the total scores. Embedded schemes are proposed for the response-hiding schemes in the next section to allow multi-keyword ranked searches when the system includes one user and server.

5.4. Embedded Ranked SE Schemes

As mentioned above, embedded schemes are introduced for the response-hiding schemes to support multi-keyword searches with result ranking. The issue mentioned in the Paillier-Based ranked schemes is that the server thinks two encrypted identifiers belonging to the same identifier that are encrypted with keyword-specific keys as different identifiers and cannot find the total encrypted scores. Thus, to calculate the total encrypted scores, both the identifiers and scores should be sent to the user who will find the total scores and rank the identifiers by these scores. In this way, the user adds the encrypted scores homomorphically or decrypts them and adds the scores. However, we consider that if the user performs the addition operations, the relevance scores can be directly concatenated with the identifiers since the user already decrypts the identifiers in the response-hiding schemes. With this way, the user gets the scores in plaintext for the identifiers and adds them to find the total scores. So, Embedded MR-RH2Lev and Embedded MR-DynRH schemes are proposed. Figure 5.2 illustrates an instance of this approach with the frequencies in Table 5.2.

5.5. Matrix-Based Ranked SE Schemes

This is a hybrid approach that combines a modified version of efficient and secure data structures of 2Lev scheme (Cash et al. (2014)) with an updated ranking algorithm of SRMES (Ibrahim et al. (2012)) to allow document ranking for multi-keyword searches.

Label	Value
$F(K_1^1, 0)$	$E(K_2^1, E(K_3^1, \langle D_2, 5\rangle))$
$F(K_1^1, 1)$	$E(K_2^1, E(K_3^1, \langle D_{10}, 2\rangle))$
$F(K_1^2, 0)$	$E(K_2^2, E(K_3^2, \langle D_4, 3\rangle))$
$F(K_1^2, 1)$	$E(K_2^2, E(K_3^2, \langle D_5, 4\rangle))$
$F(K_1^2, 2)$	$E(K_2^2, E(K_3^2, \langle D_{10}, 8\rangle))$
$\ldots$	$\ldots$

Figure 5.2. The Embedded MR-DynRH scheme index created for the given example inverted index.

We prefer to use structures of 2Lev scheme since SRMES may require too much padding in some cases, whereas 2Lev scheme regards locality and variabilities within datasets, mentioned in Section 4.1. We modify the structures to facilitate ranking by concatenating document identifiers with relevance scores, as in OPE-Based and Paillier-Based ranked schemes, but in this approach, the relevance scores are encrypted using a key-based hash function. SRMES uses probabilistic Paillier cryptosystem to encrypt the relevance scores so that repeated scores are shown as different scores and the information leaked to the server about the distribution of the scores is reduced. The key-based hash function that we use produces deterministic values, however, it produces different results when the same score is concatenated for different keywords thanks to keyword-specific keys. Hence, the amount of leakage is also reduced.

The ranking algorithm (see Algorithm 3) used in SRMES for ranking the documents of a keyword is modified to increase search efficiency since it requires calculation of different number of hash functions depending on the number of the documents matching the keyword. We modify this algorithm by keeping the document identifiers in the structures in descending order of relevance scores. The modified algorithm can be seen in Algorithm 4 in which the server fills the upper triangular part of the matrix by just checking the equality of two encrypted scores thanks to ranking rather than checking the hash of their concatenation in the BF. So, in the upper triangular part, if the first score is not equal to the second score, then -1 is set (which means that it is greater than the second one). Otherwise, they are equal and 0 is set. Then, if -1 is set in a cell of a row once, then it is not required to continue comparisons for this row and the remaining cells in the row are set to -1, as well. The cells in the lower triangular part of the matrix are assigned to

Algorithm 4 Modified Ranking

Input: FileID and Escore (both of length L);

Output: FileID and PScore

```
 1: Initialize $L * L$ matrix M of zeros, flag as false and pos as 0
 2: for $i = 1$ to $L$ do:
 3:     for $j = 1$ to $L$ do:
 4:         if $i < j$ :
 5:             if Escore$(i) \neq$ Escore$(j)$:
 6:                 M$(i, j) = -1$
 7:                 flag = true ; pos = $j$
 8:                 break
 9:         else:
10:             M$(i, j) = -$ M$(j, i)$
11:     if flag true:
12:         for $j = pos + 1$ to $L$ do:
13:             M$(i, j) = -1$
14:         flag = false
15: Set PScore to $\{sm_1, ..., sm_L\}$, where $sm_j$ is the summation of row $j$ in M
16: Output FileID and PScore.
    ▷ PScore is the set of partial scores.
```

the opposite values of those in the upper part. Figure 5.3 gives the matrix created in the Modified Ranking Algorithm for the example query mentioned in Section 4.5. It is easily seen that the upper part consists of only 0s and -1s.

Matrix-Based MR-RR2Lev and Matrix-Based MR-RH2Lev schemes are proposed for this approach. These schemes allow multi-keyword ranked queries. The search algorithm of them is similar to that in the Paillier-Based SR-RR2Lev scheme, since the server gets the (encrypted) identifiers (FileID) and encrypted scores (EScore) by performing several parsing operations. However, the server calls the Modified Ranking Algorithm to find the partial score of each document identifier for the searched keyword. In Matrix-Based MR-RR2Lev scheme, the server can calculate the total score of each identifier by adding the partial scores. But, in Matrix-Based MR-RH2Lev scheme, the server cannot calculate the total scores since it does not know the original identifiers. According to our scenarios described in Chapter 2, there are a single user and server in the system. Therefore, the server combines all FileID and Escore, as in SRMES, but it sends them to the user who decrypts the encrypted identifiers and adds the partial scores of the individual identifier to compute the total score of it.

$$\begin{array}{c|c|c|c|c|c|c|c|l}
 & 12 & 6 & 6 & 4 & 4 & 3 & 2 & \\
\hline
12 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & sm_1 = -6 \\
\hline
6 & 1 & 0 & 0 & -1 & -1 & -1 & -1 & sm_2 = -3 \\
\hline
6 & 1 & 0 & 0 & -1 & -1 & -1 & -1 & sm_3 = -3 \\
\hline
4 & 1 & 1 & 1 & 0 & 0 & -1 & -1 & sm_4 = 1 \\
\hline
4 & 1 & 1 & 1 & 0 & 0 & -1 & -1 & sm_5 = 1 \\
\hline
3 & 1 & 1 & 1 & 1 & 1 & 0 & -1 & sm_6 = 4 \\
\hline
2 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & sm_7 = 6 \\
\hline
\end{array}$$

Figure 5.3. The matrix created in the Modified Ranking Algorithm for an example single-keyword query associated with the 7 document identifiers.

CHAPTER 6

ANALYSIS OF BASE AND PROPOSED SCHEMES

In this chapter, the base and proposed schemes studied in the Chapter 4 and Chapter 5 are compared in terms of various metrics, such as characteristic, query functionality, leakage and efficiency, as shown in Table 6.2. The column “Char” specifies characteristic of a scheme as static or dynamic. The column “Query Func” is query functionality and can be single, multiple or ranked. Leakage is considered as index leakage “Ind”, search leakage “Search” and update leakage “Upd”. For efficiency, storage size, token size, search time, communication size, and resolve time are computed.

The notation used in Table 6.2 can be listed as follows:

- $\text{DB}(w)$ is the document identifiers that contain the keyword w.
- m is the number of unique keywords in the document collection, $m = |W|$.
- N is the total number of keyword-document pairs, $N = \sum_{w \in W} |\text{DB}(w)|$.
- r is the number of document identifiers matching the keyword w, $r = |\text{DB}(w)|$.
- q is the number of keywords in a multi-keyword query.
- r_t is the number of document identifiers matching a multi-keyword query.
- r_T is the sum of the number of matching document identifiers for each keyword in a multi-keyword query.
- $r_M = \max_w |\text{DB}(w)|$, where w is a keyword in a multi-keyword query.
- b and B are block sizes used in 2Lev-based schemes.
- S is the number of blocks in the array of 2Lev-based schemes (Cash et al. (2014)).

$$S = \sum_{w:|\text{DB}(w)|>b} \lceil |\text{DB}(w)|/B \rceil + \sum_{w:|\text{DB}(w)|>Bb} \lceil |\text{DB}(w)|/B^2 \rceil. \tag{6.1}$$

- The parameters $k, k', k_o, k'_o, k_p, k'_p, k_h$ and k'_h are used to notice differences between complexities more clearly.

- ▷ k is the size of identifiers in response-revealing schemes, and k' is the size of ciphertexts of identifiers in response-hiding schemes. So, ($k' > k$).
- ▷ k_o and k'_o specify sizes for OPE-based schemes. k_o is the sum of sizes an identifier and a score encrypted using OPE scheme. k'_o differs from k_o by considering encrypted identifiers instead of identifiers. So, ($k'_o > k_o$).
- ▷ k_p and k'_p specify sizes for Paillier-based schemes, and ($k'_p > k_p$).
- ▷ k_h and k'_h specify sizes for Matrix-based schemes, and ($k'_h > k_h$).

- D, P and PA define the complexities of symmetric decryption, Paillier decryption and Paillier addition algorithms, respectively. Costs of Paillier cryptosystem algorithms that are explained in Table 2.1 can be seen in Table 6.1.
- Sorting r document identifiers matching a keyword takes $O(r\cdot \log r)$ time.
- RelOr means that a scheme leaks the relevance order of a query to the server.
- In dynamic schemes, the server learns the size of update, and if a keyword being added was previously searched, then the server can also learn newly added identifiers for this keyword by reusing the keys of the keyword that are sent to it before. Those are not displayed in Table 6.2.
- The update pattern of (id, w) is up(id, w) and defines *when* the keyword w is added to or deleted from the document id.
- The update pattern of keyword w is UP(w) and defines the set of identifiers including w added or deleted later along with the information up(id, w).

Table 6.1. Costs of Paillier Cryptosystem Algorithms.

Paillier Decryption (P)	**Paillier Addition (PA)**
• 2 modular exponentiations • 2 subtractions • 2 divisions • 1 modular inverse • 1 multiplication • 1 mod	• 1 multiplication • 1 mod

Table 6.2. Comparison of Base and Proposed Ranked Schemes. The given analysis is for the worst-case complexity in a single-thread setting. All these schemes leak search pattern.

Scheme	**Char**	**Query Func**	**Leakage**			**Efficiency**						
			Ind	**Search**	**Upd**	**Storage Size**		**Token Size**	**Search Time**	**Comm Size**	**Resolve Time**
						User	**Server**				
RR2Lev	Sta	Sing	m, S	DB(w)	–	–	$O((m+S) \cdot B \cdot k)$	$O(1)$	$O(b \cdot B \cdot D(Bk))$	$O(r \cdot k)$	–
RH2Lev	Sta	Sing	m, S	–	–	–	$O((m+S) \cdot B \cdot k')$	$O(1)$	$O(b \cdot B \cdot D(Bk'))$	$O(r \cdot k')$	$O(r \cdot D(k'))$
DynRR	Dyn	Sing	N	DB(w), UP(w)	up(id, w)	$O(m)$	$O(N \cdot k)$	$O(1)$	$O(r \cdot D(k))$	$O(r \cdot k)$	–
DynRH	Dyn	Sing	N	UP(w)	up(id, w)	$O(m)$	$O(N \cdot k')$	$O(1)$	$O(r \cdot D(k'))$	$O(r \cdot k')$	$O(r \cdot D(k'))$
Sorted SR-RR2Lev	Sta	Sing, Rnk	m, S	DB(w), RelOr	–	–	$O((m+S) \cdot B \cdot k)$	$O(1)$	$O(b \cdot B \cdot D(Bk))$	$O(r \cdot k)$	–
Sorted SR-RH2Lev	Sta	Sing, Rnk	m, S	RelOr	–	–	$O((m+S) \cdot B \cdot k')$	$O(1)$	$O(b \cdot B \cdot D(Bk'))$	$O(r \cdot k')$	$O(r \cdot D(k'))$
OPE-Based SR-RR2Lev	Sta	Sing, Rnk	m, S	DB(w), RelOr	–	–	$O((m+S) \cdot B \cdot k_o)$	$O(1)$	$O(b \cdot B \cdot D(Bk_o)$ $+ r \cdot \log r)$	$O(r \cdot k)$	–
OPE-Based SR-RH2Lev	Sta	Sing, Rnk	m, S	RelOr	–	–	$O((m+S) \cdot B \cdot k'_o)$	$O(1)$	$O(b \cdot B \cdot D(Bk'_o)$ $+ r \cdot \log r)$	$O(r \cdot k')$	$O(r \cdot D(k'))$
OPE-Based SR-DynRR	Dyn	Sing, Rnk	N	DB(w), UP(w), RelOr	up(id, w)	$O(m)$	$O(N \cdot k_o)$	$O(1)$	$O(r \cdot D(k_o)$ $+ r \cdot \log r)$	$O(r \cdot k)$	–
OPE-Based SR-DynRH	Dyn	Sing, Rnk	N	UP(w), RelOr	up(id, w)	$O(m)$	$O(N \cdot k'_o)$	$O(1)$	$O(r \cdot D(k'_o)$ $+ r \cdot \log r)$	$O(r \cdot k')$	$O(r \cdot D(k'))$
Paillier-Based SR-RR2Lev	Sta	Sing, Rnk	m, S	DB(w)	–	–	$O((m+S) \cdot B \cdot k_p)$	$O(1)$	$O(b \cdot B \cdot D(Bk_p))$	$O(r \cdot k_p)$	$O(r \cdot P(k_p)$ $+ r \cdot \log r)$

Table 6.2. Comparison of Base and Proposed Ranked Schemes. The given analysis is for the worst-case complexity in a single-thread setting. All these schemes leak search pattern. (cont.)

Scheme	Char	Query Func	Leakage			Efficiency					
			Ind	Search	Upd	Storage Size		Token Size	Search Time	Comm Size	Resolve Time
						User	Server				
Paillier-Based SR-RH2Lev	Sta	Sing, Rnk	m, S	–	–	–	$O((m+S) \cdot B \cdot k'_p)$	$O(1)$	$O(b \cdot B \cdot D(Bk'_p))$	$O(r \cdot k'_p)$	$O(r \cdot P(k'_p)$ $+ r \cdot \log r)$
Paillier-Based SR-DynRR	Dyn	Sing, Rnk	N	DB(w), UP(w)	up(id, w)	$O(m)$	$O(N \cdot k_p)$	$O(1)$	$O(r \cdot D(k_p))$	$O(r \cdot k_p)$	$O(r \cdot P(k_p)$ $+ r \cdot \log r)$
Paillier-Based SR-DynRH	Dyn	Sing, Rnk	N	UP(w)	up(id, w)	$O(m)$	$O(N \cdot k'_p)$	$O(1)$	$O(r \cdot D(k'_p))$	$O(r \cdot k'_p)$	$O(r \cdot P(k'_p)$ $+ r \cdot \log r)$
Paillier-Based MR-RR2Lev	Sta	Mul, Rnk	m, S	DB(w)	–	–	$O((m+S) \cdot B \cdot k_p)$	$O(q)$	$O(q \cdot b \cdot B \cdot D(Bk_p)$ $+ r_T \cdot PA(k_p))$	$O(r_t \cdot k_p)$	$O(r_t \cdot P(k_p)$ $+ r_t \cdot \log r_t)$
Paillier-Based MR-DynRR	Dyn	Mul, Rnk	N	DB(w), UP(w)	up(id, w)	$O(m)$	$O(N \cdot k_p)$	$O(q)$	$O(q \cdot r_M \cdot D(k_p)$ $+ r_T \cdot PA(k_p))$	$O(r_t \cdot k_p)$	$O(r_t \cdot P(k_p)$ $+ r_t \cdot \log r_t)$
Embedded MR-RH2Lev	Sta	Mul, Rnk	m, S	–	–	–	$O((m+S) \cdot B \cdot k')$	$O(q)$	$O(q \cdot b \cdot B \cdot D(Bk'))$	$O(r_T \cdot k')$	$O(r_T \cdot D(k')$ $+ r_t \cdot \log r_t)$
Embedded MR-DynRH	Dyn	Mul, Rnk	N	UP(w)	up(id, w)	$O(m)$	$O(N \cdot k')$	$O(q)$	$O(q \cdot r_M \cdot D(k'))$	$O(r_T \cdot k')$	$O(r_T \cdot D(k')$ $+ r_t \cdot \log r_t)$
Matrix-Based MR-RR2Lev	Sta	Mul, Rnk	m, S	DB(w), RelOr	–	–	$O((m+S) \cdot B \cdot k_h)$	$O(q)$	$O(q \cdot (b \cdot B \cdot D(Bk_h)$ $+r^2 \cdot k_h) + r_t \cdot \log r_t)$	$O(r_t \cdot k)$	–
Matrix-Based MR-RH2Lev	Sta	Mul, Rnk	m, S	–	–	–	$O((m+S) \cdot B \cdot k'_h)$	$O(q)$	$O(q \cdot (b \cdot B \cdot D(Bk'_h)$ $+r^2 \cdot k'_h))$	$O(r_T \cdot k')$	$O(r_T \cdot D(k')$ $+ r_t \cdot \log r_t)$

[*] In "Search Leakage" column of multi-keyword schemes, w means that the leakage exists for each keyword in a query.
[*] In "Upd Leakage" column of dynamic schemes, the leakage up(id, w) exists for each keyword/document pair being added/deleted.

CHAPTER 7

EXPERIMENTAL RESULTS

In this chapter, we evaluate the performance of our proposed schemes which are implemented in Java. Our experiments are executed on an Intel Core i7 8750 2.2 GHz CPU with 16GB RAM running Windows 10. The experiments are conducted on a real dataset, RFC, from which 8262 text files with a total size of 440 MB are randomly chosen. Lucene (The Apache Software Foundation (Luc)) is used to extract keywords from each RFC file by tokenizing the text, removing the stopwords, converting the keywords to lower-case, and reducing the keywords to their root form (stemming). Thus, when a query is searched, initially all the pre-processing steps are followed for (each keyword of) the query, and then the corresponding token(s) is/are computed. The big and small block sizes are chosen as 100 and 10 for 2Lev variants according to the limits in Section 4.1.

For the symmetric encryption AES in CTR mode with a 256-bit key and for the keyed hash function HMAC-SHA512 are instantiated, and the Bouncy Castle library is used to implement them. The Clusion framework (Kamara and Moataz (2017)) is used for the base schemes which are extended to achieve the document ranking for single- and multi-keywords. The ciphertext lengths of OPE scheme and Paillier encryption scheme are defined as 64-bit and 1024-bit, respectively.

To measure the performance of the certain pieces of our code correctly, we utilize from Java Microbenchmark Harness (JMH) which is a powerful tool to build, run and analyze microbenchmarks. We write benchmark codes for search and resolve algorithms, and execute them by specifying parameters, such as the number of warmup and measurement iterations, the benchmark mode, and so on. All time results are based on 100 executions.

The schemes are compared in terms of the following evaluation metrics which are: (i) query efficiency, (ii) communication overhead, and (iii) storage overhead.

7.1. Query Efficiency

The execution times of the search and resolve algorithms in the schemes are measured as follows:

Table 7.1. Keywords to measure the query time of the single keyword schemes.

Keyword	Number of documents matching the keyword
oversimplified	10
blackhole	50
phenomenon	100
lookup	1000
response	5000

- For single keyword schemes, keywords that are included in varying numbers of documents are chosen to see the effect of the number of documents matching the keyword on search time, resolve time and total query time of each scheme. The chosen keywords and their matching numbers are illustrated in Table 7.1.
- For multi-keyword schemes, queries having different numbers of keywords, so different result sizes, are chosen to understand how the result size influences search time, resolve time and total query time. The keywords and their matching numbers are listed in Table 7.2, and the queries created from these keywords and their result sizes are given in Table 7.3. The queries are abbreviated as Q1, Q2, Q3, Q4 and Q5.

Figure 7.1 shows the benchmark results of schemes based on RR2Lev for single keyword searches. Specifically, the schemes are Sorted SR-RR2Lev, OPE-Based SR-RR2Lev and Paillier-Based SR-RR2Lev. Sorted SR-RR2Lev just keeps the identifiers in order, so its search time is the same as that of the base scheme RR2Lev. According to the

Table 7.2. Keywords used to create multi-keyword queries.

Keyword	Number of documents matching the keyword	Keyword	Number of documents matching the keyword
phenomenon	100	stuff	107
fluctuate	105	pretend	105
pairwise	102	agenda	105
sigma	104	callback	109
arrow	109	scratch	104

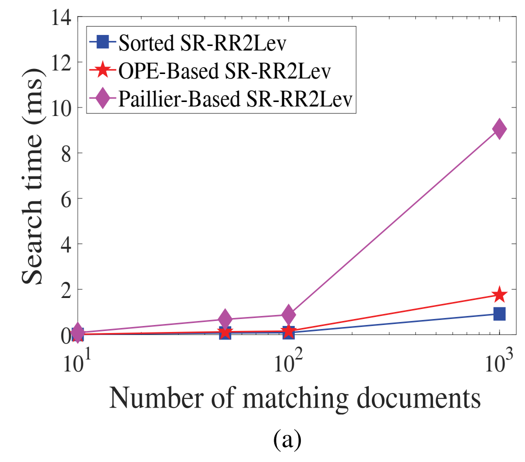
Table 7.3. Queries to measure the query time of the multi-keyword schemes.

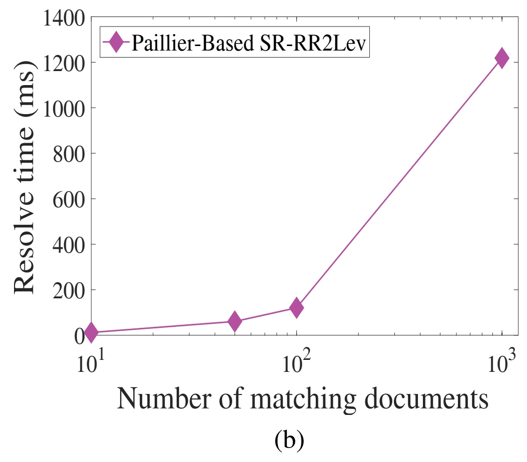
Query	Abbr	Result Size
phenomenon fluctuate	Q1	199
phenomenon fluctuate pairwise sigma	Q2	397
phenomenon fluctuate pairwise sigma arrow stuff	Q3	588
phenomenon fluctuate pairwise sigma arrow stuff pretend agenda	Q4	759
phenomenon fluctuate pairwise sigma arrow stuff pretend agenda callback scratch	Q5	933

Figure 7.1, the followings can be deduced:

- Search times of all the schemes increase with the number of documents matching the keyword since the server performs more decryption and parsing operations.
- The search time of OPE-Based SR-RR2Lev is about *twice* that of RR2Lev for all result sizes since the OPE-Based scheme parses the identifiers and ranks them by the scores, that is also another reason for the increase in the search time as the number of matching documents increases.
- The dictionary and array created for OPE-Based SR-RR2Lev and Paillier-Based SR-RR2Lev are similar, but only their encryption schemes used to encrypt the score are different. Therefore, their search algorithms are similar until the server obtains all the identifiers and corresponding scores. However, increase rate of the search time is much higher in the Paillier-Based scheme in which the server cannot rank the identifiers and returns them along with the encrypted scores to the user.
- The resolve algorithm in the Paillier-Based scheme includes decryptions of the Paillier encrypted scores and ranking of the original scores, that dramatically increases with the number of matching identifiers.
- The total query time is equal to the search time for the Sorted SR-RR2Lev and OPE-Based SR-RR2Lev due to not doing any operations on the user side.

Figure 7.2 displays the benchmark results of schemes based on RH2Lev for single keyword searches. The results can be explained as follows:





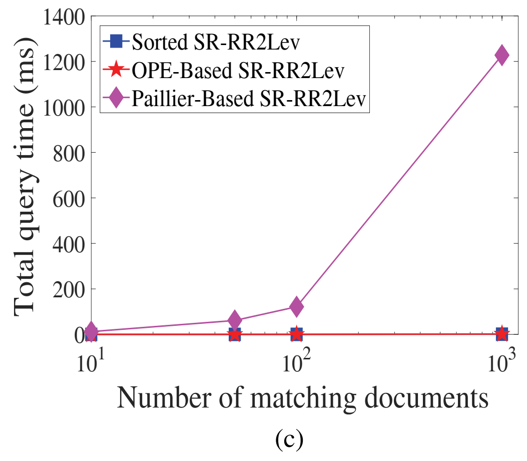
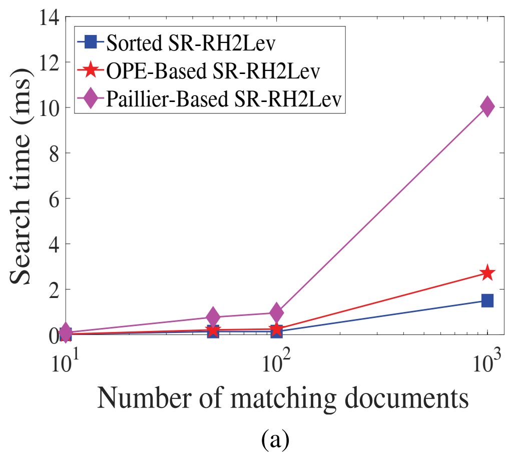
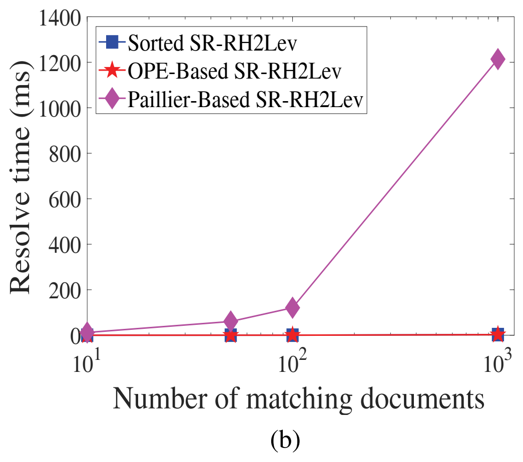


Figure 7.1. (a) Search time, (b) Resolve time, (c) Total query time of schemes based on RR2Lev for single keyword queries that match different numbers of documents.





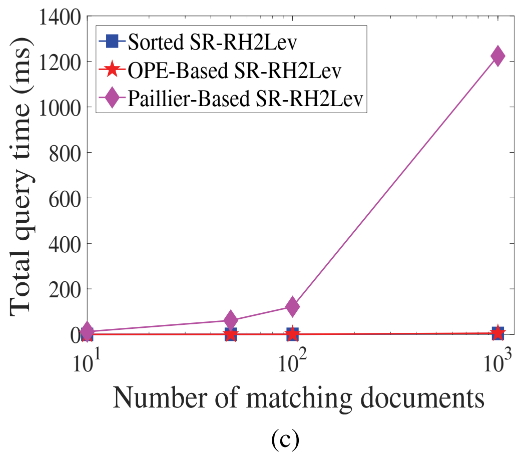


Figure 7.2. (a) Search time, (b) Resolve time, (c) Total query time of schemes based on RH2Lev for single keyword queries that match different numbers of documents.

- Search times of these schemes have similar trends to those of the RR2Lev based schemes in Figure 7.1, but encrypting the identifiers increases the search times.

- Each scheme, as a response hiding scheme, has a resolve algorithm to decrypt the encrypted identifiers. These schemes are more secure than the response revealing ones, but they require more processing on the user side.

- Total query times of these schemes increase due to increases in search times and adding also resolve times.

Figure 7.3 illustrates the benchmark results of schemes based on DynRR for single keyword searches. It can be seen from the figure that the search algorithms of DynRR and OPE-Based SR-DynRR are approximately *5-times* slower than those of RR2Lev and OPE-Based SR-RR2Lev whose results can be seen from Figure 7.1 since 2Lev is designed by considering locality and packing the several identifiers into a block. Especially, when the structures are stored on the disk, the gap between search results can grow even more. However, although the search time of Paillier-Based SR-DynRR immediately rises up as the number of matching identifiers increases, its difference from Paillier-Based SR-RR2Lev is lower than even *2 times*.

Figure 7.4 illustrates the benchmark results of schemes based on DynRH for single keyword searches. As in RH2Lev based schemes, search times of these schemes increase according to the DynRR based schemes because of encrypting the identifiers (see Figure 7.3), and they also require the user to decrypt the identifiers in the resolve algorithm after each search operation.

For RR2Lev scheme, we propose 2 multi-keyword ranked schemes that are Paillier-Based MR-RR2Lev and Matrix-Based MR-RR2Lev. Figure 7.5 gives the benchmark results of these schemes for multi-keyword searches that have different result sizes, such as 199 to 933 identifiers.

- The dictionary and array of these schemes are similar (the scores are encrypted using Paillier scheme or a key based hash function). Therefore, their search algorithms are also similar to search a keyword until the server gets all the matching identifiers and relevant encrypted scores for the keyword.

- After that, in the Paillier-Based one, the server adds the encrypted score of each identifier to its total encrypted score (homomorphic addition). When the server

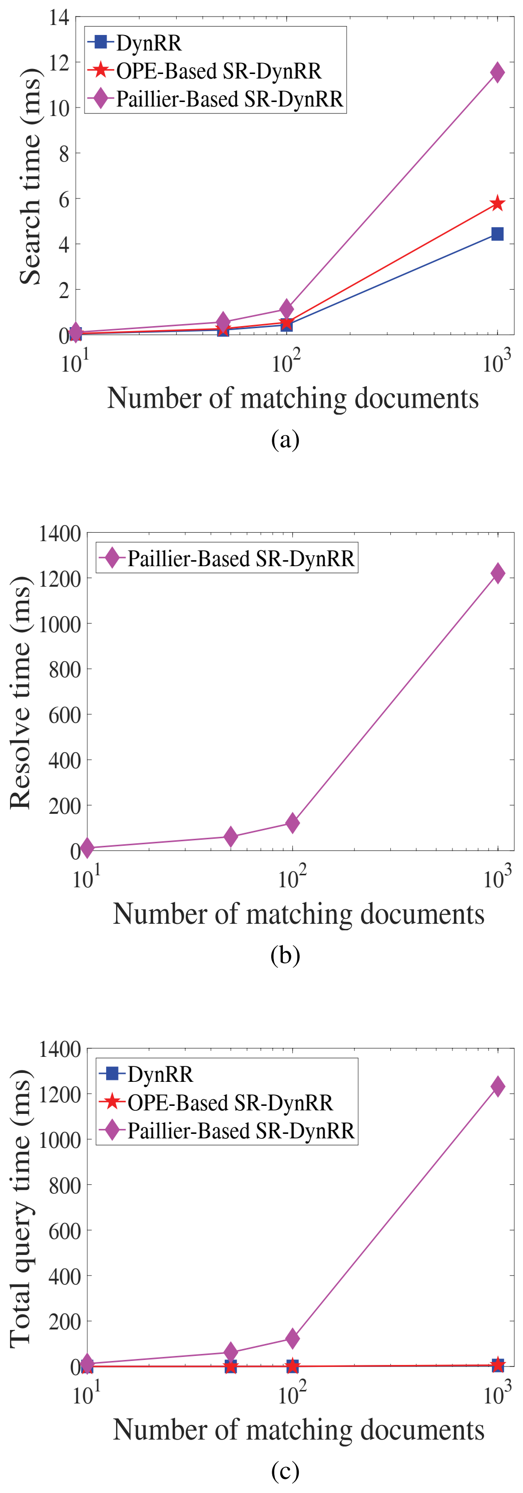


Figure 7.3. (a) Search time, (b) Resolve time, (c) Total query time of schemes based on DynRR for single keyword queries that match different numbers of documents.

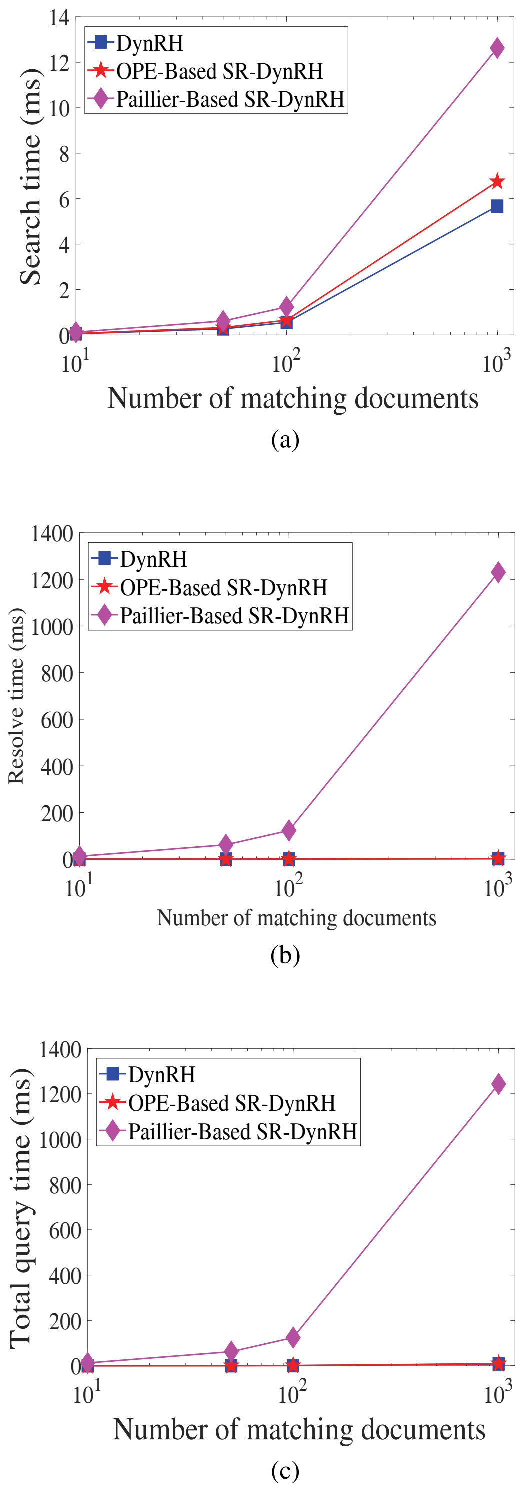


Figure 7.4. (a) Search time, (b) Resolve time, (c) Total query time of schemes based on DynRH for single keyword queries that match different numbers of documents.

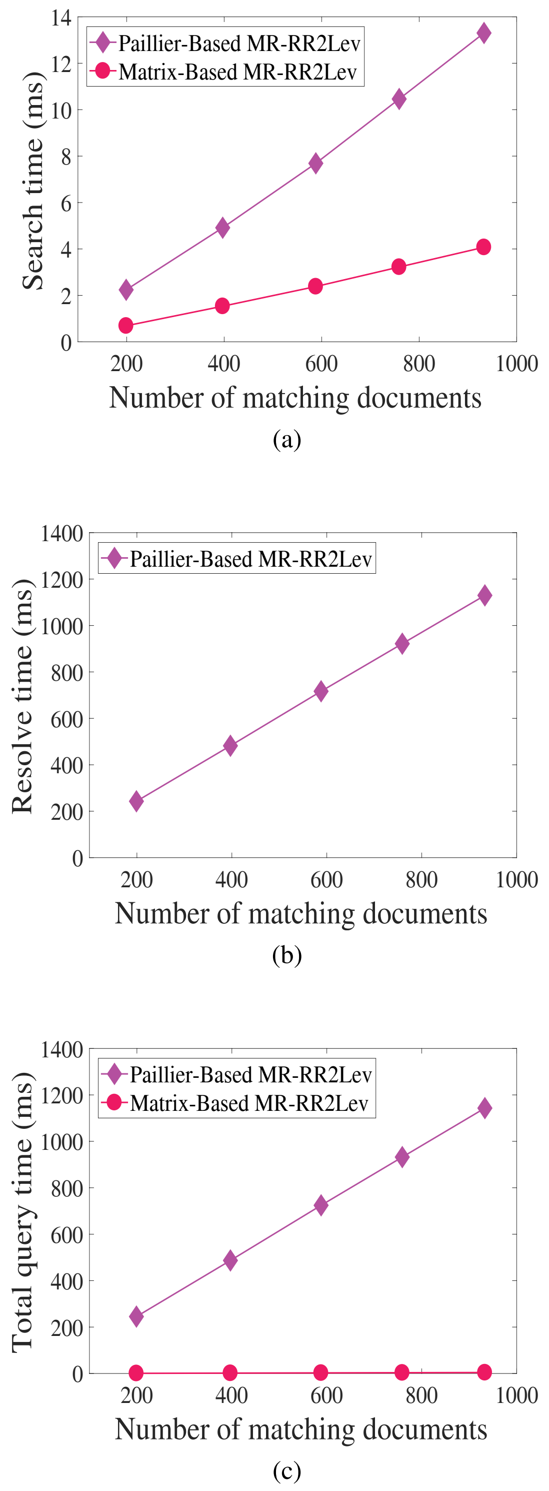


Figure 7.5. (a) Search time, (b) Resolve time, (c) Total query time of schemes based on RR2Lev for multi-keyword queries that match different numbers of documents.

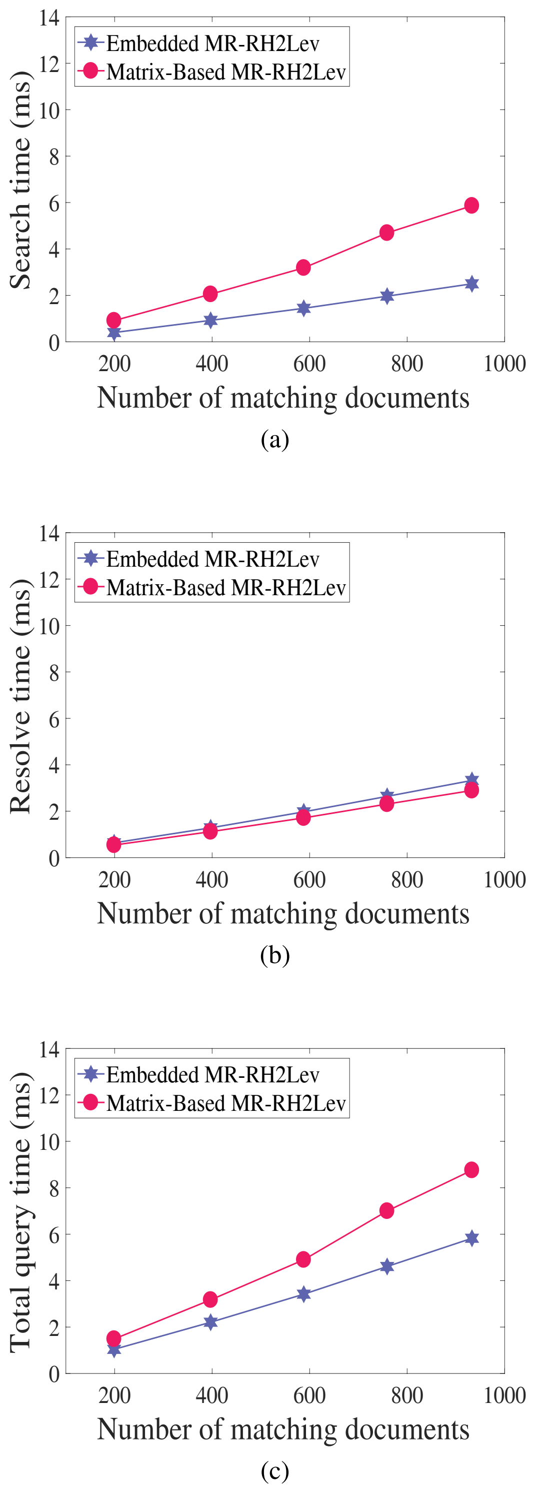


Figure 7.6. (a) Search time, (b) Resolve time, (c) Total query time of schemes based on RH2Lev for multi-keyword queries that match different numbers of documents.

completes searching all the keywords in the query, then it sends the identifiers and encrypted scores to the user.

- On the other hand, the Matrix-Based one calls the Modified Ranking Algorithm (Algorithm 4) for each keyword in the query to find the partial score of each identifier by filling the matrix. Then, it adds the score of each identifier to its total score. At the end, it sorts the identifiers according to the total scores, and sends the ranked identifiers to the user.

- It can be viewed from the figure that the search algorithm of the Matrix-Based one is slightly *3-times* more efficient than the Paillier-Based one. Considering the operations explained above, it is expected.

- In the Matrix-Based MR-RR2Lev, since the server already ranks the identifiers, there is no resolve algorithm inside it.

The proposed schemes to extend RH2Lev scheme for providing multi-keyword queries are Embedded MR-RH2Lev and Matrix-Based MR-RH2Lev whose benchmark results can be shown in Figure 7.6. Actually, before comparing them, it is readily seen that the search algorithm of Matrix-Based MR-RH2Lev is about *1,5 times* that of Matrix-Based MR-RR2Lev and it is only caused by encrypting the identifiers. The figure gives the followings:

- The Embedded one outperforms the Matrix-Based one roughly *2-times* since it just collects all the encrypted concatenations that match a keyword in the query, each includes the identifier and plaintext score, whereas the Matrix-Based one calls the Modified Ranking Algorithm after each keyword search to find the local scores.

- In the resolve algorithm, the Matrix-Based one decrypts the encrypted identifiers and adds the local score of each identifier to its total score, while the Embedded one decrypts the concatenations, gets the identifiers and scores, and also sums up the scores of each identifier. That means, the small difference between them is due to parsing the concatenations, but it increases the processing time of the user for each multi-keyword query.

Paillier-Based MR-DynRR is a variant of DynRR for ranked multi-keyword queries. Its results are displayed in Figure 7.7. When they are compared with the those of Paillier-Based MR-RR2Lev, the two results are nearly the same.

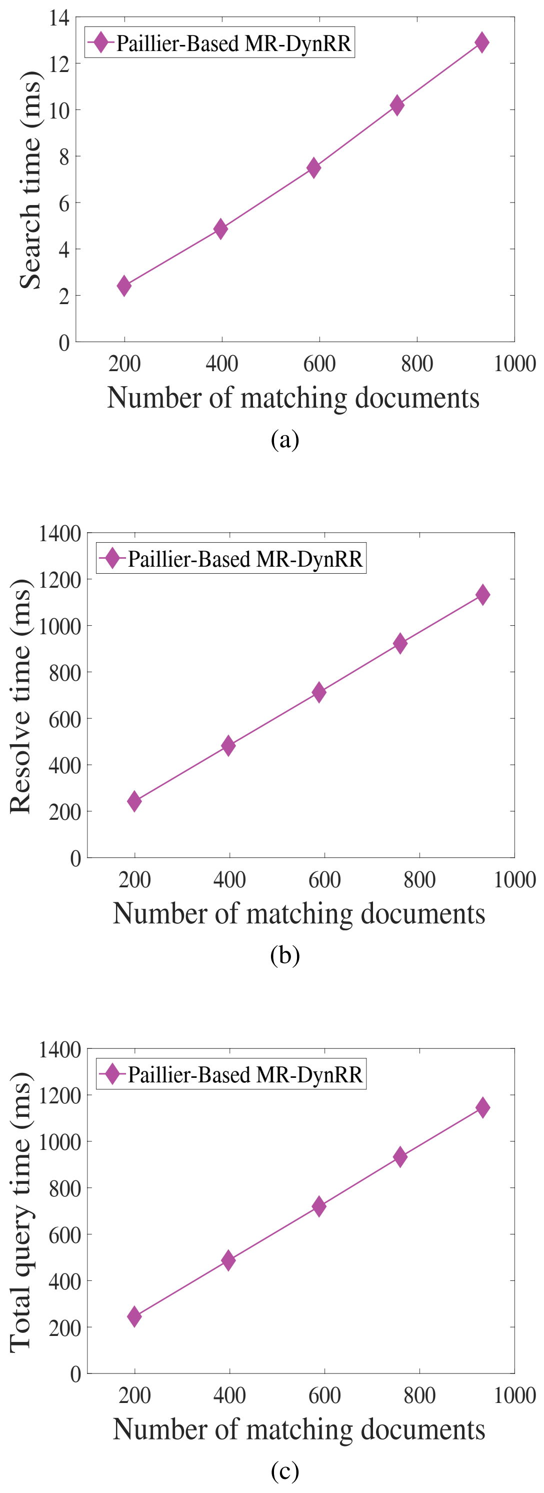


Figure 7.7. (a) Search time, (b) Resolve time, (c) Total query time of schemes based on DynRR for multi-keyword queries that match different numbers of documents.

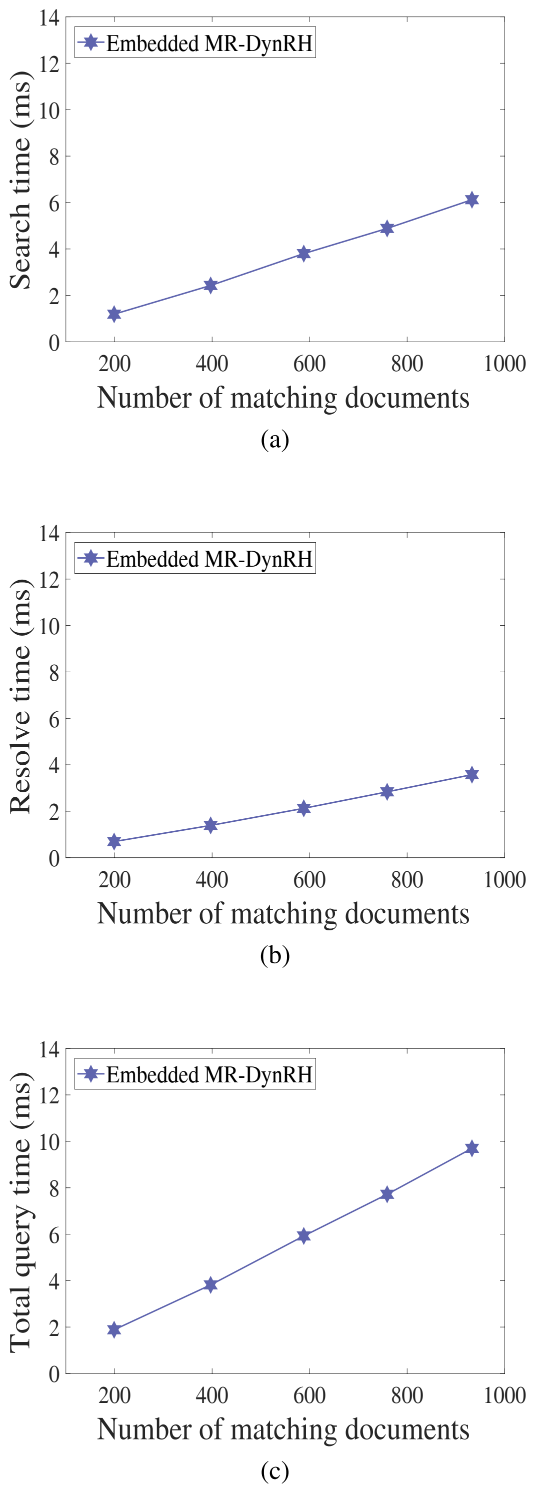


Figure 7.8. (a) Search time, (b) Resolve time, (c) Total query time of schemes based on DynRH for multi-keyword queries that match different numbers of documents.

Finally, the benchmark results of Embedded MR-DynRH are illustrated in Figure 7.8, which indicates the encryption of identifiers leads to an increase in the search time according to the Embedded MR-DynRR.

To summarize the operations of the user and server for multi-keyword queries, we create a summary table (see Table 7.4). The detailed numbers of the benchmark results are also given in Appendix A.

7.2. Communication Overhead

It refers to the data size that the server transmits to the user after a single- or multi-keyword query. First, we explain how the communication overhead of the schemes is calculated, and then find it for the schemes using an example single- and multi-keyword queries. As a single-keyword query, ”phenomenon”, and as a multi-keyword query, ”phenomenon, fluctuate” are chosen. According to Table 7.2 and Table 7.3, the (”phenomenon”), (”fluctuate”) and (”phenomenon, fluctuate”) are included in 100, 105 and 199 documents, respectively. Further, the sizes of an identifier, a symmetric-encrypted identifier, and a Paillier ciphertext are 20, 100, and 309 digits, in order.

- For single-keyword schemes, the communication overhead is calculated as follows:
  - In Paillier-Based schemes, it is (the number of document identifiers associated with the search keyword $*$ the sum of sizes of an (encrypted) identifier and a Paillier-encrypted score). If the scheme is response-revealing, it is $100 * (20 + 309)$, or if response-hiding, it is $100 * (100 + 309)$.
  - In Sorted and OPE-Based schemes, it is (the number of document identifiers associated with the search keyword $*$ the size of an (encrypted) identifier). It is $100 * 20$ for response-revealing, and $100 * 100$ for response-hiding.
- For multi-keyword schemes, the communication overhead is calculated as follows:
  - In Paillier-Based schemes, such as Paillier-Based MR-RR2Lev and Paillier-Based MR-DynRR, it is (the result set size $*$ the sum of sizes of an identifier and a Paillier-encrypted score). It is $199 * (20 + 309)$.
  - In Matrix-Based MR-RR2Lev scheme, it is (the result set size $*$ the size of an identifier). It is $199 * 20$.

Table 7.4. A summary for operations of multi-keyword ranked schemes.

Scheme	**Search (Server)**	**Resolve (User)**
• Paillier-Based MR-RR2Lev • Paillier-Based MR-DynRR	• Searches for each keyword • Adds *encrypted* scores for each identifier • Collects each identifier and its encrypted score	• Decrypts each Paillier-encrypted score • Ranks the identifiers
• Embedded MR-RH2Lev • Embedded MR-DynRH	• Searches for each keyword • Collects all the encrypted concatenations, each consists of the identifier and score	• Decrypts each concatenation (AES) • Adds *scores* for each identifier • Ranks the identifiers
• Matrix-Based MR-RR2Lev	• Searches for each keyword • Calls the Modified Ranking Algorithm to find the local scores • Adds the local scores to calculate the total score for each identifier • Ranks the identifiers	
• Matrix-Based MR-RH2Lev	• Searches for each keyword • Calls the Modified Ranking Algorithm to find the local scores • Collects each encrypted identifier and its local score	• Decrypts each identifier (AES) • Adds the local scores to calculate the total score for each identifier • Ranks the identifiers

  - In Matrix-Based MR-RH2Lev scheme, it is (the sum of the number of matching documents for each keyword in the query $*$ the sum of sizes of an encrypted identifier and a partial score). It is about $205 * 100$.
  - In Embedded schemes, it is (the sum of the number of matching documents for each keyword in the query $*$ the size of an encrypted concatenation of an identifier and score). It is roughly $205 * 100$.

7.3. Storage Overhead

The storage overhead of each scheme is measured and demonstrated in Figure 7.9. The followings can be said from the figure:

- The Embedded schemes have nearly the same storage overhead as their base schemes.
- Due to using larger ciphertext length in the Paillier-Based schemes, their storage overhead is more than the OPE-Based and Matrix-Based schemes.
- Encrypting the identifiers using a symmetric encryption scheme also causes an increase in storage.

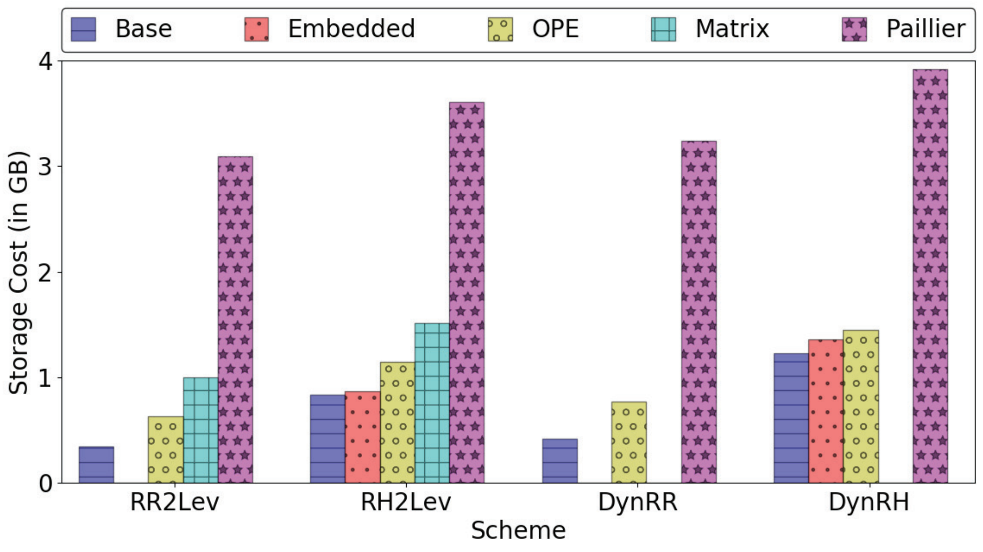


Figure 7.9. Storage cost of the base and proposed schemes.

CHAPTER 8

CONCLUSION

Due to the increase in the amount of the data and opportunities offered by cloud storage, many companies and individual users outsource their data to the untrusted servers. To guarantee the data confidentiality, the data is encrypted prior to sending to the servers. Therefore, it is necessary to design and implement efficient schemes that allow searching on the encrypted data.

In this thesis, we focus on Searchable Symmetric Encryption schemes. In addition to searching, ranking of query results is also necessary to reduce the evaluation time of the users. For this, we propose several approaches to provide secure single- and multi-keyword ranked searches, that are Sorted, OPE-Based, Paillier-Based, Embedded and Matrix-Based. We extend the base schemes (RR2Lev and RH2Lev by Cash et al. (2014), DynRR and DynRH by Kamara and Moataz (2017)) according to the proposed approaches. However, the existing structures of the schemes cannot be directly applied. That is why, we modify them by adding the ranking capability without leaking any additional information, except for the relevance order of the documents in some approaches. Moreover, we present a hybrid approach (Matrix-Based approach) that combines a modified structure of the static scheme (Cash et al. (2014)) with an updated ranking algorithm (Ibrahim et al. (2012)) so that it supports multi-keyword ranked searches in a secure and efficient way.

The approaches have different properties, such as supporting static or dynamic settings, single-keyword or multi-keyword searches, keeping or not keeping the relevance scores in the encrypted index, and using varying cryptographic tools. Therefore, the properties of the extended schemes also differ.

For future work, secure semantic ranked search schemes can also be implemented to improve search accuracy by considering the terms semantically related to the query.

REFERENCES

Agrawal, R., J. Kiernan, R. Srikant, and Y. Xu (2004). Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pp. 563–574. ACM.

Apache Lucene. [Online]. Available: `https://lucene.apache.org`.

Baldimtsi, F. and O. Ohrimenko (2015). Sorting and searching behind the curtain. In *International Conference on Financial Cryptography and Data Security*, pp. 127–146. Springer.

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

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

Bost, R., B. Minaud, and O. Ohrimenko (2017). Forward and backward private searchable encryption from constrained cryptographic primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1465–1482. ACM.

Brakerski, Z. (2012). Fully homomorphic encryption without modulus switching from classical gapsvp. In *Advances in cryptology–crypto 2012*, pp. 868–886. Springer.

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

Cash, D., J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner (2014). Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS*, Volume 14, pp. 23–26. Citeseer.

Cash, D., S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner (2013). Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology–CRYPTO 2013*, pp. 353–373. Springer.

Chang, Y.-C. and M. Mitzenmacher (2005). Privacy preserving keyword searches on remote encrypted data. In *International Conference on Applied Cryptography and Network Security*, pp. 442–455. Springer.

Chase, M. and S. Kamara (2010). Structured encryption and controlled disclosure. In *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 577–594. Springer.

Curtmola, R., J. Garay, S. Kamara, and R. Ostrovsky (2006). Searchable symmetric encryption: Improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*. ACM.

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

Hahn, F. and F. Kerschbaum (2014). Searchable encryption with secure and efficient updates. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 310–320. ACM.

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

Jiang, X., J. Yu, J. Yan, and R. Hao (2017). Enabling efficient and verifiable multi-keyword ranked search over encrypted cloud data. *Information Sciences 403*, 22–41.

JMH, Java Microbenchmark Harness. [Online]. Available: `http://openjdk.java.net/projects/code-tools/jmh/`.

Kamara, S. (2015). Encrypted search. *XRDS: Crossroads, The ACM Magazine for Students 21*(3), 30–34.

Kamara, S. and T. Moataz (2017). Clusion Framework. [Online]. Available: `http://esl.cs.brown.edu/blog/clusion/`.

Kamara, S. and C. Papamanthou (2013). Parallel and dynamic searchable symmetric encryption. In *International Conference on Financial Cryptography and Data Security*, pp. 258–274. Springer.

Kamara, S., C. Papamanthou, and T. Roeder (2012). Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 965–976. ACM.

Katz, J. and Y. Lindell (2007). Introduction to modern cryptography (chapman 8 hall/crc cryptography and network security series).

Kerschbaum, F. (2015). Frequency-hiding order-preserving encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 656–667. ACM.

Kerschbaum, F. and A. Schroepfer (2014). Optimal average-complexity ideal-security order-preserving encryption. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 275–286. ACM.

Naveed, M., M. Prabhakaran, and C. A. Gunter (2014). Dynamic searchable encryption via blind storage. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pp. 639–654. IEEE.

Orencik, C., A. Selcuk, E. Savas, and M. Kantarcioglu (2016). Multi-keyword search over encrypted data with scoring and search pattern obfuscation. *International Journal of Information Security 15*(3), 251–269.

Ozmen, M. O., T. Hoang, and A. A. Yavuz (2017). Forward-private dynamic searchable symmetric encryption with effcient search. *IACR Cryptology ePrint Archive*, 1222.

Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic*

Techniques, pp. 223–238. Springer.

Poh, G. S., J.-J. Chin, W.-C. Yau, K.-K. R. Choo, and M. S. Mohamad (2017). Searchable symmetric encryption: designs and challenges. *ACM Computing Surveys (CSUR) 50*(3), 40.

Popa, R. A., F. H. Li, and N. Zeldovich (2013). An ideal-security protocol for order-preserving encoding. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pp. 463–477. IEEE.

RFC, Request For Comments. [Online]. Available: `https://www.rfc-editor.org/retrieve/bulk/`.

Shen, Y. and P. Zhang (2017). Ranked searchable symmetric encryption supporting conjunctive queries. In *International Conference on Information Security Practice and Experience*, pp. 350–360. Springer.

Song, D. X., D. Wagner, and A. Perrig (2000). Practical techniques for searches on encrypted data. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pp. 44–55. IEEE.

Stefanov, E., C. Papamanthou, and E. Shi (2014). Practical dynamic searchable encryption with small leakage. In *NDSS*, Volume 71, pp. 72–75.

Strizhov, M. and I. Ray (2014). Multi-keyword similarity search over encrypted cloud data. In *IFIP International Information Security Conference*, pp. 52–65. Springer.

Sun, W., B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li (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*, pp. 71–82. ACM.

Tekin, L. and S. Şahin (2017). Implementation and evaluation of improved secure index scheme using standard and counting bloom filters. *International Journal of Information Security Science 6*(4), 46–56.

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

Xia, Z., X. Wang, X. Sun, and Q. Wang (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.

Yavuz, A. A. and J. Guajardo (2015). Dynamic searchable symmetric encryption with minimal leakage and efficient updates on commodity hardware. In *International Conference on Selected Areas in Cryptography*, pp. 241–259. Springer.

Yi, X., R. Paulet, and E. Bertino (2014). *Homomorphic encryption and applications*, Volume 3. Springer.

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

APPENDIX A

BENCHMARK RESULTS

Table A.1. Benchmark results of schemes for single keyword searches.

Scheme	Keyword	Search Time (ms)	Resolve Time (ms)	Total Time (ms)
Sorted SR-RR2Lev	oversimplified	0,010		0,010
	blackhole	0,077		0,077
	phenomenon	0,088		0,088
	lookup	0,916		0,916
	response	4,094		4,094
OPE-Based SR-RR2Lev	oversimplified	0,018		0,018
	blackhole	0,127		0,127
	phenomenon	0,158		0,158
	lookup	1,753		1,753
	response	7,829		7,829
Paillier-Based SR-RR2Lev	oversimplified	0,088	12,105	12,193
	blackhole	0,676	60,495	61,171
	phenomenon	0,875	120,594	121,469
	lookup	9,057	1218,462	1227,519
	response	41,874	6071,328	6113,202
Sorted SR-RH2Lev	oversimplified	0,016	0,026	0,042
	blackhole	0,140	0,248	0,388
	phenomenon	0,139	0,249	0,388
	lookup	1,500	2,758	4,258
	response	6,498	12,594	19,092

(Cont. on next page)

Table A.1. Benchmark results of schemes for single keyword searches. (cont.)

Scheme	Keyword	Search Time (ms)	Resolve Time (ms)	Total Time (ms)
OPE-Based SR-RH2Lev	oversimplified	0,026	0,026	0,052
	blackhole	0,213	0,124	0,337
	phenomenon	0,250	0,248	0,498
	lookup	2,715	2,489	5,204
	response	12,124	12,307	24,431
Paillier-Based SR-RH2Lev	oversimplified	0,096	12,130	12,226
	blackhole	0,771	60,456	61,227
	phenomenon	0,962	120,857	121,819
	lookup	10,044	1213,750	1223,794
	response	46,053	6089,372	6135,425
SR-DynRR	oversimplified	0,042		0,042
	blackhole	0,215		0,215
	phenomenon	0,433		0,433
	lookup	4,437		4,437
	response	22,792		22,792
OPE-Based SR-DynRR	oversimplified	0,053		0,053
	blackhole	0,268		0,268
	phenomenon	0,546		0,546
	lookup	5,777		5,777
	response	29,641		29,641
Paillier-Based SR-DynRR	oversimplified	0,111	12,204	12,315
	blackhole	0,562	61,202	61,764
	phenomenon	1,128	121,446	122,574
	lookup	11,544	1220,254	1231,798
	response	58,982	6084,800	6143,782

(Cont. on next page)

Table A.1. Benchmark results of schemes for single keyword searches. (cont.)

Scheme	Keyword	Search Time (ms)	Resolve Time (ms)	Total Time (ms)
SR-DynRH	oversimplified	0,053	0,025	0,078
	blackhole	0,270	0,125	0,395
	phenomenon	0,551	0,248	0,799
	lookup	5,669	2,466	8,135
	response	29,096	12,458	41,554
OPE-Based SR-DynRH	oversimplified	0,064	0,025	0,089
	blackhole	0,324	0,123	0,447
	phenomenon	0,654	0,248	0,902
	lookup	6,752	2,470	9,222
	response	34,945	12,483	47,428
Paillier-Based SR-DynRH	oversimplified	0,123	12,311	12,434
	blackhole	0,615	61,535	62,150
	phenomenon	1,235	122,968	124,203
	lookup	12,630	1230,321	1242,951
	response	64,509	6154,880	6219,389

Table A.2. Benchmark results of schemes for multi-keyword searches.

Scheme	Query	Search Time (ms)	Resolve Time (ms)	Total Time (ms)
Paillier-Based MR-RR2Lev	Q1	2,240	242,798	245,038
	Q2	4,914	481,923	486,837
	Q3	7,689	716,659	724,348
	Q4	10,460	921,618	932,078
	Q5	13,303	1129,725	1143,028
Matrix-Based MR-RR2Lev	Q1	0,689		0,689
	Q2	1,534		1,534
	Q3	2,381		2,381
	Q4	3,222		3,222
	Q5	4,074		4,074
Embedded MR-RH2Lev	Q1	0,400	0,639	1,039
	Q2	0,924	1,284	2,208
	Q3	1,443	1,973	3,416
	Q4	1,967	2,641	4,608
	Q5	2,499	3,326	5,825
Matrix-Based MR-RH2Lev	Q1	0,913	0,543	1,486
	Q2	2,052	1,119	3,171
	Q3	3,186	1,712	4,898
	Q4	4,690	2,309	6,999
	Q5	5,862	2,894	8,756
Paillier-Based MR-DynRR	Q1	2,410	242,537	244,947
	Q2	4,861	482,016	486,877
	Q3	7,494	712,013	719,507
	Q4	10,188	922,595	932,783
	Q5	12,897	1132,396	1145,293
Embedded MR-DynRH	Q1	1,190	0,693	1,883
	Q2	2,427	1,389	3,816
	Q3	3,799	2,125	5,924
	Q4	4,884	2,835	7,719
	Q5	6,122	3,576	9,698