Secure and Reliable Data Outsourcing in Cloud Computing

by

Ning Cao

A Dissertation
Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Degree of Doctor of Philosophy
in
Electrical and Computer Engineering

July 2012

Approved:

Professor Wenjing Lou
ECE Department
Dissertation Advisor

Professor Xinming Huang
ECE Department
Dissertation Committee

Professor Berk Sunar
ECE Department
Dissertation Committee

Professor Joshua D. Guttman
CS Department
Dissertation Committee

Abstract

The many advantages of cloud computing are increasingly attracting individuals and organizations to outsource their data from local to remote cloud servers. In addition to cloud infrastructure and platform providers, such as Amazon, Google, and Microsoft, more and more cloud application providers are emerging which are dedicated to offering more accessible and user friendly data storage services to cloud customers. It is a clear trend that cloud data outsourcing is becoming a pervasive service. Along with the widespread enthusiasm on cloud computing, however, concerns on data security with cloud data storage are arising in terms of reliability and privacy which raise as the primary obstacles to the adoption of the cloud. To address these challenging issues, this dissertation explores the problem of secure and reliable data outsourcing in cloud computing. We focus on deploying the most fundamental data services, e.g., data management and data utilization, while considering reliability and privacy assurance.

The first part of this dissertation discusses secure and reliable cloud data management to guarantee the data correctness and availability, given the difficulty that data are no longer locally possessed by data owners. We design a secure cloud storage service which addresses the reliability issue with near-optimal overall performance. By allowing a third party to perform the public integrity verification, data owners are significantly released from the onerous work of periodically checking data integrity. To completely free the data owner from the burden of being online after data outsourcing, we propose an exact repair solution so that no metadata needs to be generated on the fly for the repaired data.

The second part presents our privacy-preserving data utilization solutions supporting two categories of semantics – keyword search and graph query. For protecting data privacy, sensitive data has to be encrypted before outsourcing, which obsoletes traditional data utilization based on plaintext keyword search. We define and solve the challenging problem of privacy-preserving multi-keyword ranked search over encrypted data in cloud computing. We establish a set of strict privacy requirements for such a secure cloud data utilization system to become a reality. We first propose a basic idea for keyword search based on secure inner product computation, and then give two improved schemes to achieve various stringent privacy requirements in two different threat models. We also investigate some further enhancements of our ranked search mechanism, including supporting more search semantics, i.e., TF $\times$ IDF, and dynamic data operations.

As a general data structure to describe the relation between entities, the graph has been increasingly used to model complicated structures and schemaless data, such as the personal social network, the relational database, XML documents and chemical compounds. In the case that these data contains sensitive information and need to be encrypted before outsourcing to the cloud, it is a very challenging task to effectively utilize such graph-structured data after encryption. We define and solve the problem of privacy-preserving query over encrypted graph-structured data in cloud computing. By utilizing the principle of filtering-and-verification, we pre-build a feature-based index to provide feature-related information about each encrypted data graph, and then choose the efficient inner product as the pruning tool to carry out the filtering procedure.

To my beloved family

Acknowledgements

Without the generous support of many people this dissertation would not have been possible. First, I want to thank my advisor, Dr. Wenjing Lou, who guided me into the research area of cloud computing and security, discussed about my ideas, challenged me with sharping questions, gave suggestions about the structure of the paper and did proofread with great patience. I appreciate the insightful suggestions Dr. Lou has given to me and the strong belief she has put in me. Her hard working and passion for research also has set an example that I would like to follow.

I want to thank my thesis committee, Dr. Xinming Huang, Dr. Berk Sunar and Dr. Joshua D. Guttman, for serving on my dissertation committee, and for their good questions and insightful comments on my work. I also want to thank Dr. Kui Ren for discussing with me on the technique details, helping me think through the difficult problems and kind support to my job application.

My sincere gratitude further goes to current and former fellow graduate students in the Cyber Security lab, including Kai Zeng, Shucheng Yu, Zhenyu Yang, Ming Li, Hanfei Zhao, Qiben Yan, Yao Zheng, Ning Zhang, Wenhai Sun, Bing Wang, and Tingting Jiang for their friendship and support. Besides, I would like to extend my thanks to my fellow collaborators, Cong Wang and Qian Wang, for their invaluable discussions and the kind friendship. I also wish to take this opportunity to say thank you to my fellow friends at WPI: Sisi Luo, Qi Ao, Xiaodong Huang, Shiquan He, Xiaolin Hu, Ya Zhang, Yuqin Yao and many others, for their accompany during my study here.

Finally, I want to thank my family. I am in debt to my parents, Fengjin Cao and Enying Liang, who always encouraged me to pursue what I wanted. I want to thank my younger sister, Bo Cao, giving me courage when I face difficulties. I am very indebted to my dearest grandmother, who passed away during my Ph.D study. I also want to thank my dearest wife, Ruijun Fu. To accompany me studying abroad, Ruijun sacrificed many chances of studying and working. Without her support, encouragement and love, I cannot imagine how I can go through this four-year study. She is the best gift I have ever received in my life. Special thanks also go to my parents in-law, Genfa Fu and Yali Yang. Thanks for their support and love in Ruijun and me. This thesis is dedicated to all my dearest family members.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Cloud computing is the long dreamed vision of computing as a utility, where cloud customers can remotely store their data into the cloud so as to enjoy the high quality networks, servers, applications and services from a shared pool of configurable computing resources [103]. The advantages of cloud computing include on-demand self-service, ubiquitous network access, location independent resource pooling, rapid resource elasticity, usage-based pricing, transference of risk, etc. [11] Its great flexibility and economic savings are motivating both individuals and enterprises to outsource their local complex data management system into the cloud.

Along with the widespread enthusiasm on cloud computing, however, concerns on data security with cloud storage are arising due to unreliability of the service and malicious attacks from hackers. Recently more and more events on cloud service outage or server corruption with major cloud infrastructure providers are reported [2–4, 117]. Data breaches of noteworthy cloud services also appear from time to time [84, 99, 100]. Besides, the cloud service providers may also voluntarily examine customers' data for various motivations. Therefore, we argue that the cloud is intrinsically neither secure nor reliable from the view point of the cloud customers. Without providing strong security, privacy and reliability guarantee, it would be hard to expect cloud customers to turn over control of their data to cloud servers solely based on economic savings and service flexibility. To address these concerns and thus motivate the wide adoption of data outsourcing in cloud, in this thesis we will explore the problem of secure and reliable data outsourcing. We aim at deploying the most fundamental data services including data management and data utilization, with built-in reliability and privacy assurance as well as high level service performance, usability, and scalability.

Firstly, in addition to major cloud infrastructure providers, such as Amazon, Google, and Microsoft, more and more third-party cloud data service providers are emerging which are dedicated to offering more accessible and user friendly storage services to cloud customers [11]. Examples include Dropbox [1] which already has millions of users. It is a clear trend that cloud storage is becoming a pervasive service. With the increasing adoption of cloud computing for data storage, assuring data service reliability, in terms of data correctness and availability, has been outstanding. While existing solutions address the reliability issue by adding data redundancy to multiple servers, the problem becomes challenging in the "pay-as-you-use" cloud paradigm where we always want to efficiently resolve it for both corruption detection and data repair. Prior distributed storage systems based on erasure codes or replication techniques have either high decoding computational cost for data users, or too much burden of data storage and repair cost for data owners. Recently Chen et al. [37] proposed a network coding-based storage system which provides a decent solution for efficient data repair. This scheme, based on previous work [43, 44, 78, 121], reduces the communication cost for data repair to the information theoretic minimum. This is achieved by recoding encoded packets in the healthy servers during the repair procedure. However, as network coding utilizes Gaussian elimination for decoding, the data retrieval in terms of computation cost is more expensive than erasure codes-based systems. Hence, new secure and reliable storage solutions with the efficiency consideration of both data repair and data retrieval are entailed in the cloud computing.

Secondly, to protect data privacy and combat unsolicited accesses in the cloud and beyond, sensitive data, e.g., emails, personal health records, tax documents, financial transactions, etc., may have to be encrypted by data owners before outsourcing to the commercial public cloud [58]; this, however, obsoletes the traditional data utilization service based on plaintext keyword search. The trivial solution of downloading all the data and decrypting locally is clearly impractical, due to the huge amount of bandwidth cost in cloud systems. Moreover, aside from eliminating the local storage management, storing data into the cloud serves no purpose unless they can be easily searched and utilized. Thus, exploring privacy-preserving and effective search service over encrypted cloud data is of paramount importance. Considering the potentially large number of on-demand data users and huge amount of outsourced data documents in the cloud, this problem is particularly challenging as it is extremely difficult to meet also the requirements of performance, system usability and scalability. Related works on searchable encryption focus on single keyword search or Boolean keyword search, and rarely sort the search results. How to design an efficient encrypted data search mechanism that supports multi-keyword ranking semantics without privacy breaches still remains a challenging open problem.

Thirdly, we further explore the data search within another category of search semantics in cloud computing. As a general data structure to describe the relation between entities, the graph has been increasingly used to model complicated structures and schemaless data, such as the personal social network (the social graph), the relational database, XML documents and chemical compounds studied by research labs [38, 91, 92, 128, 132, 133]. Images in the personal album can also be modeled as the attributed relational graph (ARG) [20]. For the protection of users' privacy, these sensitive data also have to be encrypted before outsourcing to the cloud. Moreover, some data are supposed to be shared among trusted partners. For example, the lab director and members are given the authorization to access the entire lab data. Authorized users are usually planning to retrieve some portion of data they are interested rather than the entire dataset, mostly because of the "pay-for-use" billing rule in the cloud computing paradigm. Considering the large amount of data centralized in the cloud datacenter, the key challenge here is to realize an efficient encrypted query design which supports graph semantics without privacy breaches.

1.2 Contributions

In this dissertation, the fundamental problem of secure and reliable data outsourcing in Cloud Computing is tackled. The detailed and challenging research tasks we solved are outlined below:

Secure and Reliable Cloud Storage In Chapter 2, we address the problem of secure and reliable cloud storage with efficiency consideration of both data repair and data retrieval. By utilizing a near-optimal erasure codes, specifically LT codes, our designed storage service has faster decoding during data retrieval than existing solutions. To minimize the data repair complexity, we employ the exact repair method to efficiently recover the exact form of any corrupted data. Such a design also reduces the data owner's cost during data repair since no verification tag needs to be generated (old verification tags can be recovered as same as data recovery). Our proposed cloud storage service provides a better overall efficiency of data retrieval and repair than existing counterparts. It also completely releases the data owner from the burden of being online by enabling public integrity check and exact repair. Portions of the work studied in this chapter were presented as extended abstract at the 31th IEEE Conference on Computer Communications (INFOCOM'12) [32].

Privacy-Preserving Multi-Keyword Ranked Search over Encrypted Cloud Data In Chapter 3, we address the problem of privacy-preserving multi-keyword ranked search over encrypted data in cloud computing. We establish a set of strict privacy requirements for such a secure cloud data utilization system. Among various multi-keyword semantics, we choose the efficient similarity measure of "coordinate matching", i.e., as many matches as possible, to capture the relevance of data documents to the search query. We further use "inner product similarity" to quantitatively evaluate such similarity measure. We first propose a basic idea for ranked keyword search based on secure inner product computation, and then give two improved schemes to achieve various stringent privacy requirements in two different threat models. We also investigate some further enhancements of our ranked search mechanism, including supporting more search semantics, i.e., TF $\times$ IDF, and dynamic data operations. Portions of the work studied in this chapter were presented as extended abstract at the 30th IEEE Conference on Computer Communications (INFOCOM'11) [30].

Privacy-Preserving Query over Encrypted Graph-Structured Cloud Data In Chapter 4, we address the problem of privacy-preserving query over encrypted graph-structured data in cloud computing. Our work utilizes the principle of "filtering-and-verification". We pre-build a feature-based index to provide feature-related information about each encrypted data graph, and then choose the efficient inner product as the pruning tool to carry out the filtering procedure. To meet the challenge of supporting graph query without privacy breaches, we improve the secure inner product computation to achieve various privacy requirements under the known-background threat model. Portions of the work studied in this chapter was presented as extended abstract at the 31th International Conference on Distributed Computing Systems (ICDCS'11) [31].

1.3 Roadmap

The organization of this dissertation is as follows.

Chapter 2 presents our solution for secure and reliable cloud storage. Section 2.1 describes the problem as well as the main idea of our solution. In Section 2.2, we formulate the problem by introducing the system model, the threat model, our design goals, and the preliminary. Section 2.3 gives our design rationale, followed by section 2.4, which describes the proposed scheme. Section 2.5 and 2.6 discuss security analysis and performance analysis, respectively. We discuss related work on both network coding-based distributed storage and remote data integrity check in Section 2.7, and conclude the chapter in Section 2.8.

Chapter 3 presents our proposed privacy-preserving multi-keyword ranked search over encrypted cloud data. Section 3.1 describes the problem as well as the main idea of our solution. In Section 3.2, we introduce the system model, the threat model, our design goals, and the preliminary. Section 3.3 describes the framework and privacy requirements, followed by section 3.4, which describes the proposed schemes. Section 3.5 discusses supporting more search semantics and dynamic operation. Section 3.6 presents simulation results. We discuss related work on both single and Boolean keyword searchable encryption in Section 3.7, and conclude the chapter in Section 3.8.

Chapter 4 presents our proposed privacy-preserving query over encrypted graph-structured cloud data. Section 4.1 describes the problem as well as the main idea of our solution. In Section 4.2, we introduce the system model, the threat model and our design goals. Section 4.3 gives preliminaries, and section 4.4 describes the framework and privacy requirements, followed by section 4.5, which gives our proposed scheme. Section 4.6 presents evaluation results. We discuss related work on both keyword searchable encryption and graph containment query in Section 4.7, and conclude the chapter in Section 4.8.

Chapter 5 concludes this dissertation and presents several directions for future work.

Chapter 2

Secure and Reliable Cloud Storage

2.1 Introduction

The many advantages of cloud computing are increasingly attracting individuals and organizations to move their data from local to remote cloud servers [58]. In addition to major cloud infrastructure providers [11], such as Amazon, Google, and Microsoft, more and more third-party cloud data service providers are emerging which are dedicated to offering more accessible and user friendly storage services to cloud customers. It is a clear trend that cloud storage is becoming a pervasive service. Along with the widespread enthusiasm on cloud computing, however, concerns on data security with cloud storage are arising due to unreliability of the service. For example, recently more and more events on cloud service outage or server corruption with major cloud infrastructure providers are reported [3, 4], be it caused by Byzantine failures and/or malicious attacks. Such a reality demands for reliable data storage to tolerate certain outage/corruption. In particular, the cloud storage service should offer cloud customers with capabilities of: 1) timely detection of any server (and hence data) corruption event, 2) correct retrieval of data even if a limited number of servers are corrupted, and 3) repair of corrupted data from uncorrupted data. Although existing techniques have provided solutions for them individually, the main challenge for cloud storage service is to simultaneously provide these capabilities at minimal cost. This is because in cloud computing both data storage and transmission are charged in the “pay-as-you-use” manner. Solutions of high cost will discourage user engagement and be of less practical use. Moreover, it is important to set cloud customers free by minimizing the complexity imposed on them in terms of computation/communication cost and burden of being online.

Existing solutions address the reliability issue by adding data redundancy to multiple servers. These techniques can be categorized into replication-based solutions and erasure codes-based ones. Data replication is the most straightforward way of adding redundancy. The advantage of replication is its simplicity in data management. Repair of data on corrupted servers is also straightforward by simply copying the entire data from a healthy server. The main drawback of replication is its high storage cost. Moreover, replication-based solutions cannot satisfy the high-throughput requirement in distributed storage service like cloud computing, where a large number of users may access the service concurrently. This is because different users may want to access different pieces of data on a server, which would cause less cache hits but frequent disk I/O requests. [126] provides a detailed analysis on this drawback.

As compared to its replication-based counterparts, erasure codes-based solutions can achieve the required reliability level with much less data redundancy [116]. Different from replication-based solutions, erasure codes-based ones are more suitable for distributed storage systems with concurrent user access. This is because every block of data on a server is useful for decoding the original data, which leads to a high cache hit rate of the system. There have been a large number of related works on erasure codes-based distributed storage systems [60,61,126]. The main drawback of existing optimal erasure codes-based systems, however, is the high communication cost needed for repairing a corrupted storage server. It is commonly believed that the communication cost is equal to the size of the entire original data [43]. For example, Reed-Solomon codes [79] usually need to reconstruct all the original packets in order to generate a fragment of encoded packets. Taking into consideration the large amount of data outsourced, the entire data reconstruction is expensive which makes this solution less attractive. Similarly, existing distributed storage systems based on near-optimal erasure codes [126] do not have an efficient solution for the data repair problem or pay no attention to it.

Recently Chen et al. [37] proposed a network coding-based storage system which provides a decent solution for efficient data repair. This scheme, based on previous work [43, 44, 78, 121], reduces the communication cost for data repair to the information theoretic minimum. This is achieved by recoding encoded packets in the healthy servers during the repair procedure. However, as network coding utilizes Gaussian elimination for decoding, the data retrieval in terms of computation cost is more expensive than erasure codes-based systems. Moreover, [37] adopts so-called functional repair for data repair, i.e., corrupted data is recovered to a correct form, but not the exact original form. While this is good for reducing data repair cost, it requires the data owner to produce new verification tags, e.g., cryptographic message authentication code, for newly generated data blocks. As the computational cost of generating verification tags is linear to the number of data blocks, this design will inevitably introduce heavy computation/communication cost on the data owner. Moreover, the data owner has to stay online during data repair.

In this chapter, we explore the problem of secure and reliable storage in the "pay-as-you-use" cloud computing paradigm, and design a cloud storage service with the efficiency consideration of both data repair and data retrieval. By utilizing a near-optimal erasure codes, specifically LT codes, our designed storage service has faster decoding during data retrieval than existing solutions. To minimize the data repair complexity, we employ the exact repair method to efficiently recover the exact form of any corrupted data. Such a design also reduces the data owner's cost during data repair since no verification tag needs to be generated (old verification tags can be recovered as data recovery). By enabling public integrity check, our designed LT codes based secure cloud storage service (LTCS) completely releases the data owner from the burden of being online. Our contributions are summarized as follows,

1) We are among the first to explore the problem of secure and reliable cloud storage with the efficiency consideration for both data repair and data retrieval.

2) Our proposed cloud storage service provides a better overall efficiency of data retrieval and repair than existing counterparts. It also greatly reduces cost and burden of being online for the data owner by enabling public integrity check and exact repair.

3) The advantages of our proposed service are validated via both numerical analysis and experimental results.

2.2 Problem Formulation

2.2.1 The System Model

Considering a cloud data storage service which provides both secure data outsourcing service and efficient data retrieval and repair service, including four different entities: the data owner, the data user, the cloud server, and the third party server. The data owner outsources the encoded fragments of the file $\mathcal{M}$ to n cloud servers denoted as storage servers. If the data owner requires to keep the data content confidential, the file $\mathcal{M}$ can be first encrypted before encoding. Outsourced data are attached by some metadata like verification tags to provide integrity check capability. After the data outsourcing, a data user can select any k storage servers to retrieve encoded segments, and recover the file $\mathcal{M}$, which can be further decrypted in case the file is encrypted. Meanwhile, the third party server periodically checks the integrity of data stored in cloud servers. Failed cloud servers can be repaired with the help of other healthy cloud servers.

2.2.2 The Threat Model

The cloud server is considered as "curious-and-vulnerable". Specifically, the cloud server is vulnerable to Byzantine failures and external attacks. While Byzantine failures may be made by hardware errors or the cloud maintenance personnel's misbehaviors, external attacks could be ranging from natural disasters, like fire and earthquake, to adversaries' malicious hacking. After the adversary gains the control of the cloud server, it may launch the pollution attack or the replay attack which aims to break the linear independence among encoded data, by replacing the data stored in corrupted cloud server with old encoded data. If the cloud server is not corrupted, it correctly follows the designated protocol specification, but it will try to infer and analyze data in its storage and interactions during the protocol execution so as to learn additional information. This represents a threat to the privacy of cloud users' data stored on the server.

2.2.3 Design Goals

To provide secure and reliable cloud data storage services, our design should simultaneously achieve performance guarantees during data retrieval and repair.

- **Availability and Reliability:** By accessing any k-combination of n storage servers, the data user could successfully retrieve encoded data and recover all the original data. The data retrieval service remains functional when up to $n - k$ storage servers are corrupted in one round, and corrupted servers can be repaired from other healthy servers.

- **Security:** The designed storage service protects the data confidentiality and periodically checks the integrity of data in cloud servers to prevent data dropout or corruption.

- **Offline Data Owner:** Data owners can go offline immediately after data outsourcing, which means they are not required to be involved in tasks such as data integrity check and repair at a later stage.

- **Efficiency:** Above goals should be achieved with low storage, computation and communication cost for the data owner, data users and cloud servers.

2.2.4 Notations

- $\mathcal{M}$: the outsourced file, consisting of m original packets, $\mathcal{M} = (M_1, \ldots, M_m)$.

- S_l : the l-th storage server, $1 \le l \le n$.

- C_{li} : the i-th encoded packet stored in the l-th storage server, $1 \le i \le \alpha$.

- Δ_{li} : the coding vector of the encoded packet C_{li}.

- φ_l : the coding tag, used to verify all the coding vectors Δ_{li} in S_l.

- ϕ_{li} : the retrieval tag, used to verify C_{li} in the retrieval and repair.

- σ_{lij} : the verification tag, used to verify C_{li} in the integrity check, $1 \le j \le t$.

2.2.5 Preliminary on LT Codes

LT codes [69] has a typical property that the encoding procedure can generate unlimited number of encoded packets, each of which is generated by conducting bitwise XOR operation on a subset of original packets. LT codes can recover m original packets from any $m + O(\sqrt{m}\ln^2(m/\delta))$ coded packets with probability $1-\delta$. The decoding procedure is performed by the efficient Belief Propagation decoder [70] with complexity $O(m\ln(m/\delta))$. Code degree d is defined as the number of original packets that are combined into one coded packet. In LT codes, the distribution of code degree is defined by Ideal Soliton distribution or Robust Soliton distribution. The Ideal Soliton distribution is $\rho(i)$, i.e., $P\{d=i\}$, where $\sum_{i=1}^{m}\rho(i) = 1$ and

$$\rho(i) = P\{d=i\} = \begin{cases} 1/m & \text{if } i = 1 \\ 1/i(i-1) & \text{if } i = 2, \dots, m. \end{cases}$$

Robust Soliton distribution is $\mu(i)$, where $\mu(i) = (\rho(i)+\tau(i))/\beta$ and $\beta = \sum_{i=1}^{m}\rho(i)+\tau(i)$. Let $R = c \cdot \ln(m/\delta)\sqrt{m}$, and define $\tau(i)$ as follows,

$$\tau(i) = \begin{cases} R/im & \text{if } i = 1, \dots, m/R-1 \\ R\ln(R/\delta)/m & \text{if } i = m/R \\ 0 & \text{if } i = m/R+1, \dots, m. \end{cases}$$

2.3 LTCS: Design Rationale

2.3.1 Enabling Reliability and Availability

To ensure the data reliability in distributed storage systems, various data redundancy techniques can be employed, such as replication, erasure codes, and network

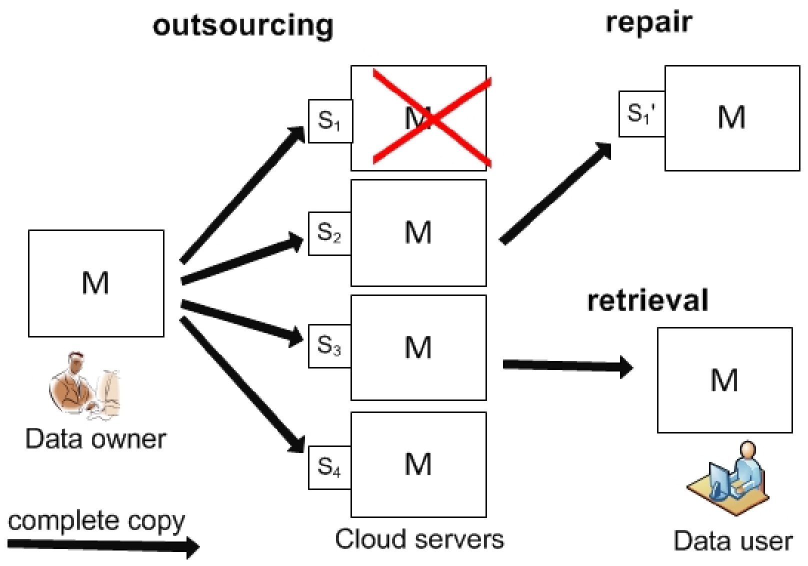


Figure 2.1: Distributed storage systems based on replication.

coding. Replication as shown in Fig. 2.1 is the most straightforward way of adding data redundancy where each of n storage servers stores a complete copy of the original data. Data users can retrieve the original data by accessing any one of the storage servers, and the corrupted server can be repaired by simply copying the entire data from a healthy server.

Given the same level of redundancy, the optimal erasure codes based distributed storage system as shown in Fig. 2.2 is more reliable by many orders of magnitude than the replication-based system [116]. Data users can recover the entire m original packets by retrieving the same number of encoded packets from any k-combination of n servers, and therefore every server only needs to store m/k encoded packets which is regarded as the property of optimal redundancy-reliability tradeoff. However, its quadratic decoding complexity makes it very inefficient for data users to recover data during data retrieval. Moreover, the communication cost to repair a failed storage server is equal to the size of the entire original data in the optimal erasure codes-based distributed storage system [43, 44]. For example, as a typical optimal

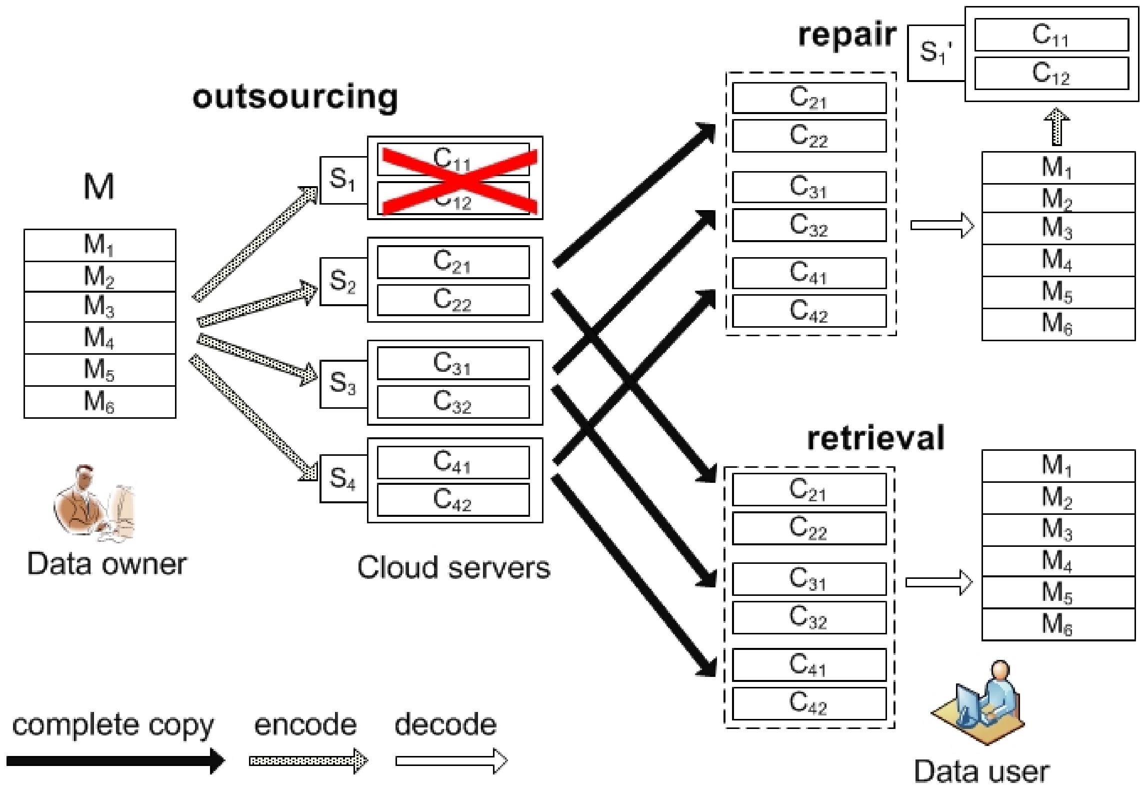


Figure 2.2: Distributed storage systems based on optimal erasure codes.

erasure codes, Reed-Solomon codes [79] usually need to reconstruct all the original packets in order to generate a fragment of encoded packets. In other words, one has to retrieve m encoded packets in order to generate only m/k encoded packets for the corrupted server.

Network coding-based storage codes [43, 44, 78, 121] as shown in Fig. 2.3 reduce the repair communication cost to the information theoretic minimum by combining encoded packets in the healthy servers during the repair procedure, where only m/k recoded packets are needed to generate the corrupted m/k encoded packets. Each server needs to store $2m/(k+1)$ encoded packets, which is more than optimal erasure codes, to guarantee that data users can retrieve m linearly independent encoded packets from any k-combination of n servers. Besides, the network coding-based storage codes have the similar inefficient decoding problem as optimal erasure

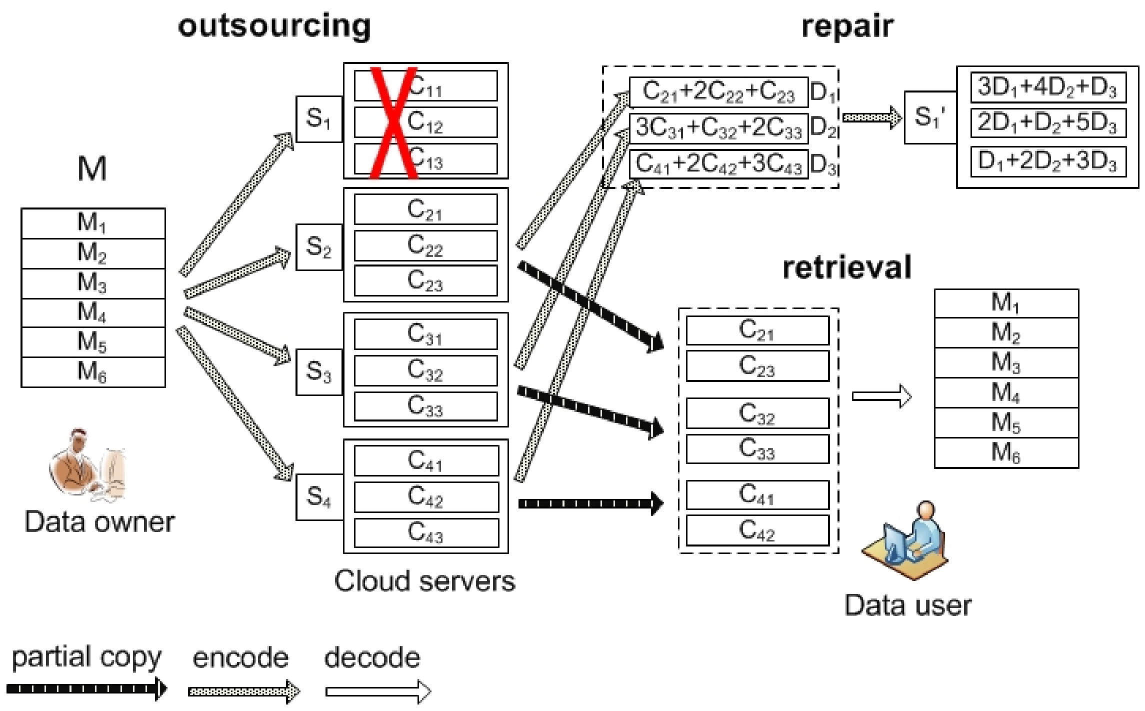


Figure 2.3: Distributed storage systems based on network coding.

codes due to the utilization of Gaussian elimination decoder.

To meet the efficient decoding requirement in the cloud data storage scenario where the data owner outsources huge amount of data for sharing with data users, our design is based on the near-optimal erasure codes, specifically LT codes, to store low-complexity encoded packets over n distributed servers. The fast Belief Propagation decoding for LT codes can be used during data retrieval in our LT codes based secure cloud storage service (LTCS). Data users can efficiently recover all the m of original packets from any $m(1+\epsilon)$ encoded packets which can be retrieved from any k-combination of n servers. To achieve so, every server needs to store at least $m(1+\varepsilon)/k$ encoded packets which is larger than the erasure codes but smaller than the network coding based storage codes.

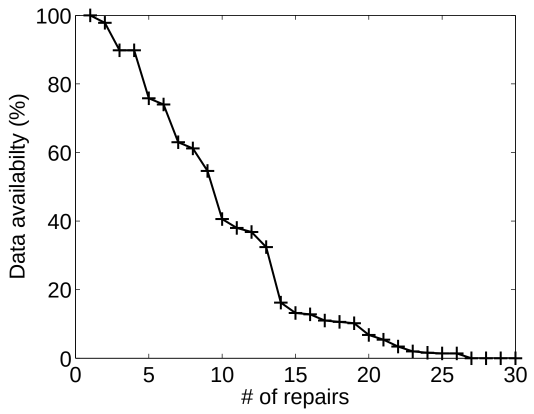


Figure 2.4: Data availabity after functional repair as in LTNC.

2.3.2 Reducing Maintenance Cost

To prevent data dropout or corruption, the integrity of data stored in each server needs to be periodically checked. In [37], the data owner raises a challenge for every encoded packet to cloud servers. Taking into consideration the large number of encoded packets with substantial data redundancy in cloud servers, the cost of such private integrity check is somehow burdensome in terms of both computation and communication for data owners. LTCS utilizes the public integrity verification which enables the data owner to delegate the integrity check task to a third party server. Once there is a server failing to pass the integrity check, the third party server immediately reports it to the administrator of the cloud server who will then activate the repair process.

The repair task in our LT codes based storage service is accomplished by generating the exactly same packets as those previously stored in corrupted storage servers. Such repair method does not introduce any additional linear dependence among newly generated packets and those packets stored in healthy storage servers, and therefore maintains the data availability. Furthermore, we run the decoding over the encoded packets before outsourcing to guarantee the reliable data retrieval and recovery. Unlike the exact repair in our designed service, the functional repair is the other category of data repair, where the repair procedure generates correct encoded packets, but not the exactly same packets as those corrupted. Attempts to apply functional repair in the LT codes based distributed storage should first solve how to recode packets, because the random linear recoding in the functional repair of network coding-based storage codes cannot satisfy the degree distribution in LT codes. It seems that this problem can be solved by utilizing the recently proposed LT network codes (LTNC) which provides efficient decoding at the cost of slightly more communication in the single-source broadcasting scenario [33]. However, after several rounds of repair with same recoding operations regulated in LT network codes, data users experience decoding failure with high probability, as illustrated in Fig. 2.4, where data availability is the probability that data users could recover original data from any k-combination of n storage servers. The major reason is that recoding operations with the degree restriction in LT network codes introduce inneglectable linear dependence among recoded packets and existing packets in LT codes based storage service. Therefore, the functional repair is not suitable for LT codes-based storage service.

2.3.3 Offline Data Owner

In the repair procedure, network coding-based storage systems with functional repair generate new encoded packets to substitute corrupted data in the failed server. The data owner needs to stay online for generating necessary tags for these new packets [37]. In LTCS, all newly generated packets for the corrupted storage server in the repair procedure are exactly the same as old ones previously stored in the server, which means their corresponding metadata are also same. Like the distributed storage of data packets, these metadata can be stored in multiple servers and recovered in case of repairing corrupted servers. The replication or erasure codes (like Reed-Solomon codes) can be adopted to reliably backup these metadata. Hence, without the burden of generating tags and checking integrity, the data owner can stay offline immediately after outsourcing the data which makes LTCS more practical to be deployed in the cloud paradigm.

2.4 LTCS: The Proposed Secure and Reliable Cloud Storage Service

In this section, we present the LT codes-based secure and reliable cloud storage service (LTCS), where n storage servers $\{S_l\}_{1 \leq l \leq n}$ are utilized to provide the data storage service for data owner and data users. Our data integrity technique is partially adapted from the BLS signature in POR [87].

2.4.1 Setup

Let $e : G \times G \rightarrow G_T$ be a bilinear map, where g is the generator of G, with a BLS hash function $H : \{0,1\}^* \rightarrow G$. The data owner generates a random number $\eta \leftarrow \mathbb{Z}_p$ and s random numbers $u_1, \ldots, u_s \leftarrow G$. The secret key sk is $\{\eta\}$, and the public key is pk $= \{u_1, \ldots, u_s, v\}$, where $v \leftarrow g^\eta$.

2.4.2 Data Outsourcing

The data outsourcing is to pre-process data and distribute them to multiple cloud servers. The file $\mathcal{M}$ is first equally split into m original packets, $M_1, \ldots, M_m$, with the same size of $\frac{|\mathcal{M}|}{m}$ bits. Following the Robust Soliton degree distribution in LT codes, m original packets are combined by exclusive-or (XOR) operations to generate $n\alpha$ encoded packets, where α is the number of packets outsourced to each storage server and set to $m/k \cdot (1+\varepsilon)$. For protecting data confidentiality, sensitive data could be encrypted before the encoding process. Existing data access control mechanisms [129] can be employed to prevent the cloud server from prying into outsourced data.

According to LT codes, all the m original packets can be recovered from any $m(1+\varepsilon)$ of encoded packets with probability $1-\delta$ by on average $O(m \cdot \ln(m/\delta))$ packet operations. However, the availability requirement specifies that data recovery should be always successful by accessing any k of healthy storage servers. To achieve this goal, the data owner checks the decodability of these encoded packets before outsourcing by executing the decoding algorithm. Specifically, all the $n\alpha$ encoded packets are divided into n groups, each of which consists of α packets, $\{\{C_{li}\}_{1\leq i\leq\alpha}\}_{1\leq l\leq n}$. The Belief Propagation decoding algorithm is then run on every k-combination of n groups. If the decoding fails in any combination, the data owner re-generates encoded packets and re-checks the decodability until every k-combination can recover all the m original packets. Once the encoding configuration successfully passes the decodability detection, it can be reused for all the storage services that specifies the same n and k.

For each encoded packet C_{li}, $1 \leq l \leq n$, $1 \leq i \leq \alpha$, three kinds of auxiliary data are attached, i.e., the coding vector, the retrieval tag, and verification tags. The coding vector Δ_{li} is a m-bit vector, where each bit represents whether the corresponding original packet is combined into C_{li} or not. The retrieval tag ϕ_{li}, computed by Eq. 2.1, is to verify the encoded packet C_{li} in data retrieval, and also in data repair if necessary.

$$\phi_{li} \leftarrow (H(l||i||C_{li}))^{\eta} \in G \tag{2.1}$$

To generate the verification tag for the purpose of integrity check, each encoded packet C_{li} is split into t segments, $\{C_{li1}, \ldots, C_{lit}\}$. Each segment C_{lij} includes s symbols in $\mathbb{Z}_p : \{C_{lij1}, \ldots, C_{lijs}\}$. For each segment C_{lij}, we generate a verification tag σ_{lij}, $1 \leq j \leq t$, in Eq. 2.2.

$$\sigma_{lij} \leftarrow (H(l||i||j) \cdot \prod_{\ell=1}^{s} u_\ell^{C_{lij\ell}})^\eta \in G \tag{2.2}$$

These data are outsourced to the l-th storage server in the form of $\{l, \{i, C_{li}, \Delta_{li}, \phi_{li}, \{\sigma_{lij}\}_{1\leq j\leq t}\}_{1\leq i\leq \alpha}, \varphi_l\}$, where φ_l is the coding tag to validate all the previously coding vectors. The computation of φ_l is shown in Eq. 2.3.

$$\varphi_l \leftarrow (H(l||\Delta_{l1}|| \ldots ||\Delta_{l\alpha}))^\eta \in G \tag{2.3}$$

2.4.3 Data Retrieval

Data users can recover original data by accessing any k of n cloud servers in the data retrieval. The data user first retrieves all the coding vectors and the coding tags stored in the selected k cloud servers, and performs the verification in Eq. 2.4. If the verification operation on any coding tag fails, the data user sends reports to the third party server and accesses one substitutive storage server.

$$e(\varphi_l, g) \stackrel{?}{=} e(H(l||\Delta_{l1}|| \ldots ||\Delta_{l\alpha}), v) \tag{2.4}$$

Once all the coding tags from k storage servers pass the validation, the data user partially executes the Belief Propagation decoding algorithm only with coding vectors, and records ids of coding vectors that are useful for the decoding. Meanwhile, the data user retrieves those corresponding useful encoded packets and their retrieval tags from corresponding storage servers, and verifies the integrity of encoded packets as shown in Eq. 2.5.

$$e(\phi_{li}, g) \overset{?}{=} e(H(l||i||C_{li}), v) \tag{2.5}$$

All the original packets in $\mathcal{M}$ can be recovered by performing the same XOR operations on encoded packets as those on coding vectors. Finally, the data user can decrypt the $\mathcal{M}$ and get the plaintext data if the file is encrypted before encoding. Note that if there exist some verification tags that fail in the integrity check procedure, the data user also reports them to the third party server and retrieves data from one substitutive storage server. When the third party server receives any failure reports from data users about either coding tags or verification tags, it will immediately challenge the corresponding server (details on challenge will be given in the following section).

2.4.4 Integrity Check

To monitor the integrity of data stored in the storage servers, the third party server periodically performs the integrity check over every storage server. The third party server first randomly picks $\alpha + t$ numbers, $a_1, \dots, a_\alpha, b_1, \dots, b_t \leftarrow \mathbb{Z}_p$, and then sends them to every storage server. The l-th storage server will compute s integrated symbols $\{\mu_{l\ell}\}_{1 \le \ell \le s}$ and one integrated tag ς_l in Eq. 2.6. Note that a_i corresponds to the i-th encoded packet in every storage server, and b_j corresponds to the j-th segment in each encoded packet.

$$\mu_{l\ell} = \sum_{i=1}^{\alpha} \sum_{j=1}^{t} a_i b_j C_{lij\ell}, \ \varsigma_l = \prod_{i=1}^{\alpha} \prod_{j=1}^{t} {\sigma_{lij}}^{a_i b_j} \tag{2.6}$$

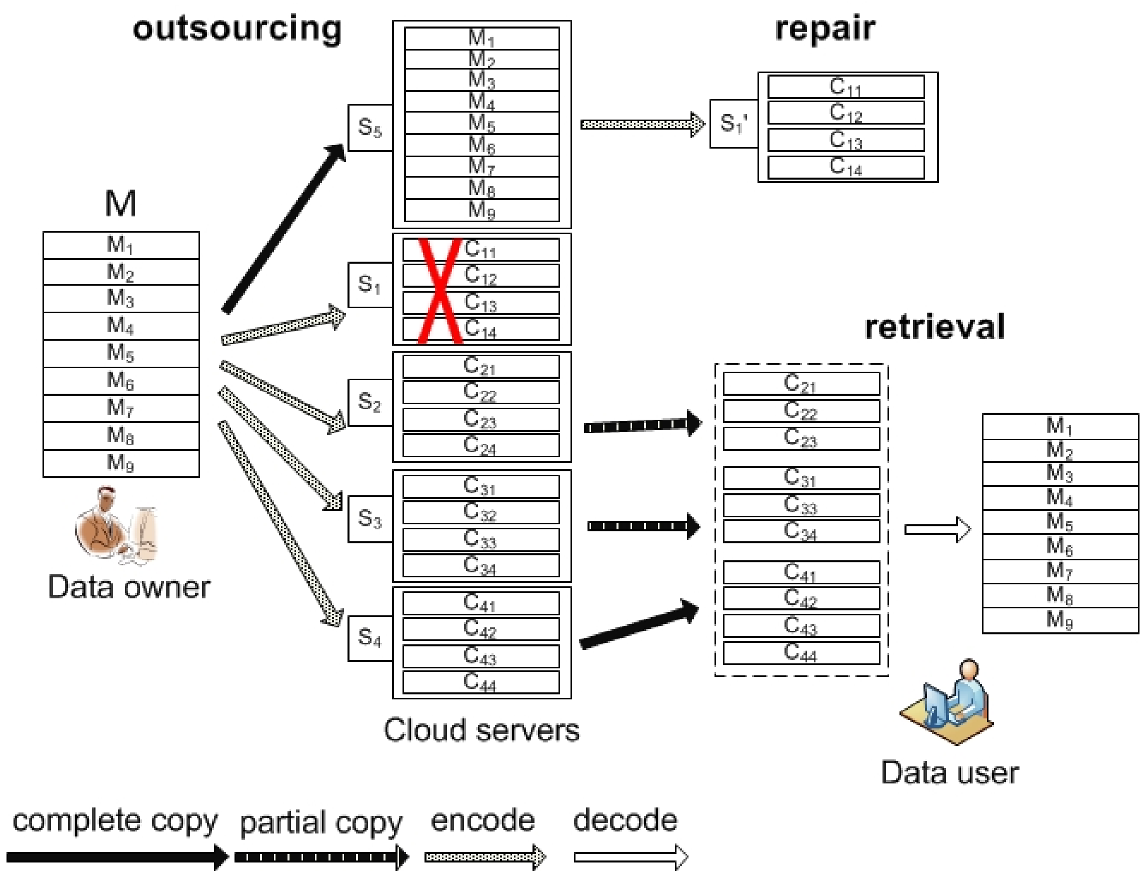


Figure 2.5: LT codes-based cloud storage service (LTCS).

The third party server verifies these received integrated symbols $\{\mu_{l\ell}\}_{1\leq\ell\leq s}$ and the integrated verification tag ς_l, as shown in Eq. 2.7.

$$e(\varsigma_l, g) \overset{?}{=} e(\prod_{i=1}^{\alpha}\prod_{j=1}^{t} H(l||i||j)^{a_i b_j} \cdot \prod_{\ell=1}^{s} u_\ell^{\mu_{l\ell}}, v) \tag{2.7}$$

If the verification fails, the third party server reports it to the data center, and the administrator of the storage server will reset the server software and start the data repair procedure.

2.4.5 Data Repair

It is commonly believed that all existing coding constructions must access the original data to generate coded packets, which means the communication cost of data repair for erasure codes is equal to the size of the entire original data [43]. A straightforward data repair method is therefore to recover all the original data packets whenever a storage server is corrupted. But such method will introduce much cost of both computation and communication. In LTCS as illustrated in Fig. 2.5, one repair server S_{n+1} is deployed to efficiently repair corrupted storage servers. Although other storage services based on optimal erasure codes or network coding can also integrate the repair server, they still introduce more computational cost during data retrieval (and storage cost for network coding-based service) than LTCS, which will be validated in section 2.6.

To accommodate the repair server, the data owner outsources all the original packets to the repair server S_{n+1} during data outsourcing. Each original packet is also attached by the verification tag which is generated in the same way as shown in Eq. 2.2. Besides, all the auxiliary data of storage servers are stored in the repair server as a backup. Similarly with the distributed data storage, the metadata including verification tags for original packets need to be reliably stored in n storage servers. Compared with the large size of encoded data, auxiliary data are quite small such that we can employ the simple replication or erasure codes to add redundancy.

To deal with the failure on the l-th storage server, the repair server uses all the corresponding coding vectors $\{\Delta_{li}\}_{1 \leq i \leq \alpha}$ to generate encoded packets $\{C_{li}\}_{1 \leq i \leq \alpha}$. Specifically, C_{li} is generated by the XOR combination of $|\Delta_{li}|$ original packets, as illustrated in Eq. 2.8, where $j_{li1}, \ldots, j_{li|\Delta_{li}|} \leftarrow \{1, \ldots, m\}$ correspond to the nonzero bits in the coding vector Δ_{li}. The repair server sends to S_l all the encoded packets

Table 2.1: Performance complexity analysis of storage services based on different redundancy techniques.

	Network Coding	Reed-Solomon	LTCS
Server storage	$O((2n/(k+1)) \cdot \lvert\mathcal{M}\rvert)$	$O((1+n/k) \cdot \lvert\mathcal{M}\rvert)$	$O((1+n(1+\varepsilon)/k) \cdot \lvert\mathcal{M}\rvert)$
Encoding comp.	$O(2nm^2/(k+1))$	$O(nm^2/k)$	$O((nm(1+\varepsilon)\ln m)/k)$
Retrieval comm.	$O(\lvert\mathcal{M}\rvert)$	$O(\lvert\mathcal{M}\rvert)$	$O(\lvert\mathcal{M}\rvert)$
Retrieval comp.	$O(m^2)$	$O(m^2)$	$O(m \ln m)$
Repair comm.	$O(2T/(k+1) \cdot \lvert\mathcal{M}\rvert)$	$O(T(1/k+1/n) \cdot \lvert\mathcal{M}\rvert)$	$O(T((1+\varepsilon)/k+1/n) \cdot \lvert\mathcal{M}\rvert)$

with their tags in the form of $\{l, \{i, C_{li}, \Delta_{li}, \phi_{li}, \{\sigma_{lij}\}_{1\le j\le t}\}_{1\le i\le\alpha}, \varphi_l\}$.

$$C_{li} = M_{j_{li1}} \oplus \ldots \oplus M_{j_{li|\Delta_{li}|}} \tag{2.8}$$

The repaired server S_l authenticates received encoded packets $\{C_{li}\}_{1\le i\le\alpha}$ and auxiliary tags as in the data retrieval and integrity check. If the authentication fails, the repair server itself may be corrupted and need repair.

The third party server also challenges the repair server S_{n+1} to check the integrity of original packets. Since there are m packets stored in S_{n+1}, instead of α in storage servers, the third party server should generate $m+t$ random numbers, $a_1, \ldots, a_m, b_1, \ldots, b_t \leftarrow \mathbb{Z}_p$. The integrated symbols $\{\mu_{(n+1)\ell}\}_{1\le\ell\le s}$ are then generated from the m original packets, $\mu_{(n+1)\ell} = \sum\limits_{i=1}^{m}\sum\limits_{j=1}^{t} a_i b_j C_{(n+1)ij\ell}$, where $C_{(n+1)i} = M_i$. There are similar changes in the generation of the integrated verification tag, $\varsigma_{n+1} = \prod\limits_{i=1}^{m}\prod\limits_{j=1}^{t} \sigma_{(n+1)ij}{}^{a_i b_j}$. The repair server is less likely to be corrupted than storage servers, since it does not participate in the data retrieval service for data users. Even when the repair server is found to be corrupted and needs repair, all the original packets and auxiliary data can be recovered by performing data retrieval from any d of healthy storage servers. Therefore, there is no single point of failure.

2.5 Security Analysis

2.5.1 Protection of Data Confidentiality and Integrity

For protecting data confidentiality, existing encryption techniques or data access control schemes [129] can be utilized before the encoding process, which prevent the cloud server from prying into outsourced data. With respect to the data integrity, LTCS utilizes various cryptographic tags to resist the pollution attack during the data repair and retrieval procedures. LTCS is also secure against the replay attack which is presented in the network coding-based distributed storage system [37]. To lunch the replay attack, the adversary first corrupts some storage servers and backups encoded packets stored in these servers. After several rounds of data repair, the adversary corrupts the same storage servers as before, and then substitutes new encoded packets with specific old packets. Since the verification tag only binds the storage server id and the packet id, not the freshness of the packet, the substituted old packets could pass the integrity verification. As a result, such substitution makes encoded packets stored in specific k-combinations of n storage servers linearly dependable, and the data recovery would fail when all other $n - k$ storage servers are corrupted. Actually, if the data repair mechanism is designed to generate new packets which are different from the old packets stored in the same storage server, any coding-based distributed storage system is somehow vulnerable to such kind of attack. In other words, the functional repair itself has the possibility to break the decodability. By contrast, LTCS employs the exact repair method where the newly generated packets are the same as those previously stored packets. The replay attack becomes invalid since there is no difference between old and new packets in the same storage server. Furthermore, LTCS examines the data decodability from any k-combination of storage servers before outsourcing, which guarantees that original data could be recovered even when the adversary corrupts both the repair server and at most $n-k$ storage servers in one round.

2.5.2 Verification Correctness in Integrity Check

The verification correctness in Eq. 2.7 is proved in Eq. 2.9.

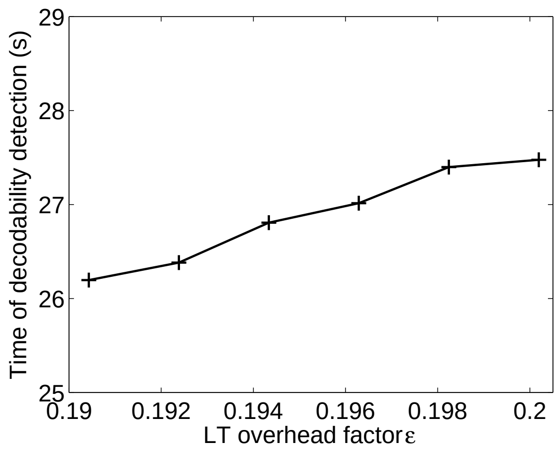
$$\begin{aligned}
e(\varsigma_l, g) &= e(\prod_{i=1}^{\alpha}\prod_{j=1}^{t} {\sigma_{lij}}^{a_i b_j}, g) \\
&= e(\prod_{i=1}^{\alpha}\prod_{j=1}^{t} (H(l||i||j)^{a_i b_j} \cdot \prod_{\ell=1}^{s} u_\ell^{a_i b_j C_{lij\ell}}), g)^{\eta} \\
&= e(\prod_{i=1}^{\alpha}\prod_{j=1}^{t} H(l||i||j)^{a_i b_j} \cdot \prod_{\ell=1}^{s}\prod_{i=1}^{\alpha}\prod_{j=1}^{t} u_\ell^{a_i b_j C_{lij\ell}}, v) \\
&= e(\prod_{i=1}^{\alpha}\prod_{j=1}^{t} H(l||i||j)^{a_i b_j} \cdot \prod_{\ell=1}^{s} u_\ell^{\sum_{i=1}^{\alpha}\sum_{j=1}^{t} a_i b_j C_{lij\ell}}, v) \\
&= e(\prod_{i=1}^{\alpha}\prod_{j=1}^{t} H(l||i||j)^{a_i b_j} \cdot \prod_{\ell=1}^{s} u_\ell^{\mu_{l\ell}}, v). \qquad (2.9)
\end{aligned}$$

2.6 Performance Analysis

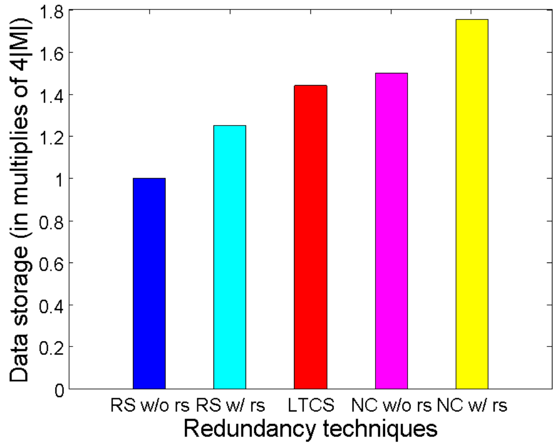
In this section, we demonstrate the performance of storage services based on different redundancy techniques by both theoretical complexity analysis and experimental evaluation. We set the same desired reliability level as network coding-based distributed storage system RDC-NC [37], where $n = 12$, $k = 3$. Other parameters are set from the consideration of specific properties of network coding (NC), Reed-Solomon codes (RS), and LT codes. For LTCS, $m = 3072, \alpha = m(1+\varepsilon)/k, d_s = 1, d_r = k, \beta = \alpha, \delta = 1, c = 0.1$, where $\varepsilon \sim O(\ln^2(m/\delta)/\sqrt{m})$ is the LT overhead factor. d_s and d_r represent the number of cloud servers participating in the repair of corrupted storage server and corrupted repair server, respectively. β represents the number of packets retrieved from each participating server during repair. For Reed-Solomon codes based storage system, $m = 6\ or\ 12, \alpha = m/k, d = k, \beta = \alpha$; for network coding based storage system, $m = 6\ or\ 12, \alpha = 2m/(k+1), d = k, \beta = \alpha/d$. The whole experiment system is implemented by C language on a Linux Server with Intel Xeon Processor 2.93GHz. Besides, the performance of network coding and Reed-Solomon codes is optimized by employing table lookup in the multiplication and division over $GF(2^8)$, and we evaluate their performance with or without repair server (rs), respectively. The performance complexity comparison among storage services based on different redundancy techniques with repair server is shown in Tab. 2.1, where T is the number of corrupted storage servers in one round, $0 \leq T \leq n - k$.

2.6.1 Outsourcing

As described in section 2.4.2, the data owner detects the decodability in the encoding procedure to guarantee data availability. To check all k-combinations of n groups, the data owner has to execute $\binom{n}{k}$ times of the Belief Propagation decoding algorithm. For the efficiency purpose, this decoding process can be partially executed where only coding vectors follow the decoding steps and data packets are not involved. If there exists a combination that cannot recover all the original packets, the data owner will re-generate $n\alpha$ coding vectors according to LT codes and re-detect them, where α is equal to $m(1 + \varepsilon)/k$. Once all the $\binom{n}{k}$ combinations successfully pass the decodability detection, corresponding coding vectors can be reused for all the storage services that specifies the same n and k. As illustrated in Fig. 2.6(a), the larger ε makes the decodability detection more costly because of the



(a) Time of detecting decodability



(b) Data storage cost

Figure 2.6: Outsourcing performance with different ε. n = 12, k = 3, m = 3072.

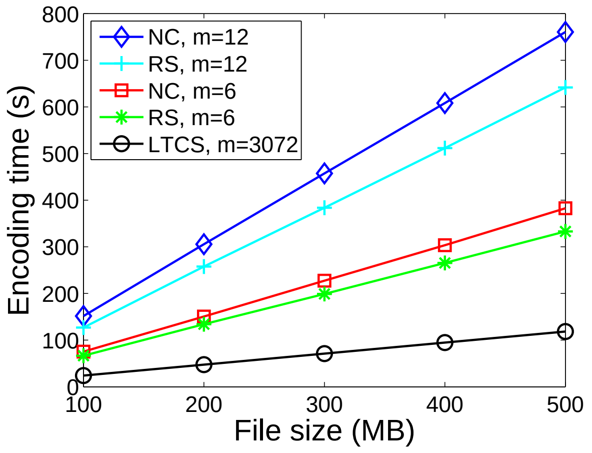
linear relation between ε and α, namely the number of coding vectors in each group. Considering that the larger ε leads to more storage cost and repair communication cost, the following evaluations are conducted by setting ε to the smallest one as 0.1904, which corresponds to $\alpha = 1219$.

Once one set of $n\alpha$ coding vectors pass the decodability detection, encoding operations are performed on real data packets via the XOR combination. Although the number of encoded packets in LTCS, $n\alpha$, is several hundreds times larger than in other storage service based on network coding or Reed-Solomon codes, the computational cost of encoding in LTCS is much less than the later, as illustrated in Fig. 2.7 (a). The main reason for such big advantage is that the average degree of encoded packets is $O(ln(m/\delta))$ and $O(m)$ in two services, respectively. Furthermore, the combination for encoding is the efficient XOR in LTCS while the linear combination in network coding or Reed-Solomon codes involves the multiplication operations with coefficients. The total number of encoded packets in Reed-Solomon codes-based service is less than network coding-based one so the encoding procedure introduces different computational cost in two services.

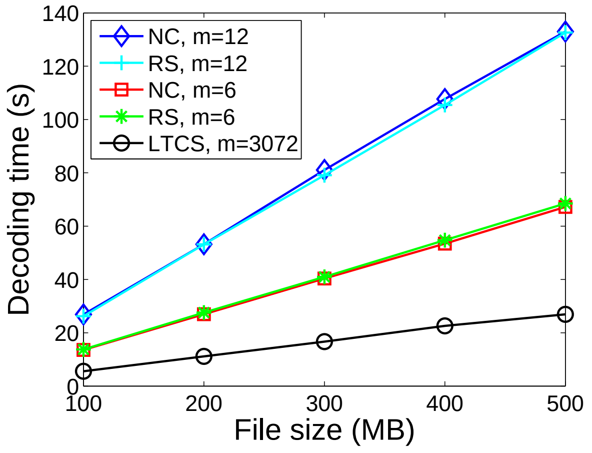
As for the data storage in LTCS, every storage server stores α encoded packets, each of which has the size of $|\mathcal{M}|/m$. And the repair server stores all the m original packets with the same size. The total data storage in cloud servers is the sum of all encoded packets in n storage servers and all original packets in the repair server, which is $O(n\alpha \cdot |\mathcal{M}|/m + |\mathcal{M}|)$, i.e., $O([1 + n(1 + \varepsilon)/k] \cdot |\mathcal{M}|)$. Since LT codes is a near-optimal erasure codes in terms of redundancy-reliability tradeoff, the data storage cost in LTCS is larger than Reed-Solomon codes-based storage service which introduces theoretical minimum storage cost as $4|\mathcal{M}|$ in our evaluation setting. By contrast, the total data storage in existing network coding-based storage service is $O(|\mathcal{M}|[2n/(k+1)])$ as illustrated in Fig. 2.6(b). If we integrate the repair server into this service, the storage cost will be $O(|\mathcal{M}|[1 + 2n/(k + 1)])$ which is much larger than LTCS.

2.6.2 Data Retrieval

The availability in data retrieval is guaranteed by the decodability detection before data outsourcing and the exact repair of corrupted data. Recall that the data user first retrieves $k\alpha$, i.e. $m(1 + \varepsilon)$, of coding vectors from k storage servers, and



(a) Encoding



(b) Decoding

Figure 2.7: Encoding and decoding time for different size of file. n=12, k=3.

then only retrieve m of encoded packets that are useful for decoding. Therefore, the communication cost during data retrieval in LTCS is the same $O(|\mathcal{M}|)$ as the network coding-based storage system where any m of encoded packets are linearly independent with high probability.

The computational complexity of Belief Propagation decoding in LTCS is $O(m \cdot \ln(m/\delta))$ for the data user, where δ is set to 1. By contrast, the other storage services based on network coding or Reed-Solomon codes usually use the costly decoding algorithms with higher computational complexity, $O(m^2)$. Although the total number of original packets, m, may be smaller in the other two storage services than in LTCS, the decoding process for the data user in LTCS performs at least two times faster than in the other two storage services, as illustrated in 2.7(b). This efficient decoding process demonstrates that LTCS is more appropriate than other redundancy-based storage services in the cloud storage paradigm, where data retrieval is a routine task for data users.

2.6.3 Integrity Check

To check the integrity of data stored in a storage server, the third party server needs to perform one integrated challenge in LTCS, which means only two bilinear maps in Eq. 2.7 are executed in order to check α encoded packets. Network coding-based service has to perform α times of challenges for each storage server where 2α bilinear maps are executed to check α of encoded packets. Similarly, the communication cost between the third party server and each storage server during one round of *Integrity Check* in network coding-based service is almost α times more than that in LTCS.

2.6.4 Data Repair

When the repair server is corrupted, LTCS first retrieve β encoded packets from each of d_r healthy storage servers to recover all the original packets. In such case, the communication complexity from d_r healthy storage servers to the repair server is $O(d_r \cdot \beta \cdot |\mathcal{M}|/m)$, i.e., $O((1+\varepsilon) \cdot |\mathcal{M}|)$, where $d_r = k$, $\beta = \alpha$. If the repair server is not corrupted or has been repaired, the data repair of storage servers in LTCS is simply accomplished by the repair server generating β encoded packets for each corrupted storage server, where $d_s = 1$, $\beta = \alpha$. Assume the number of corrupted storage

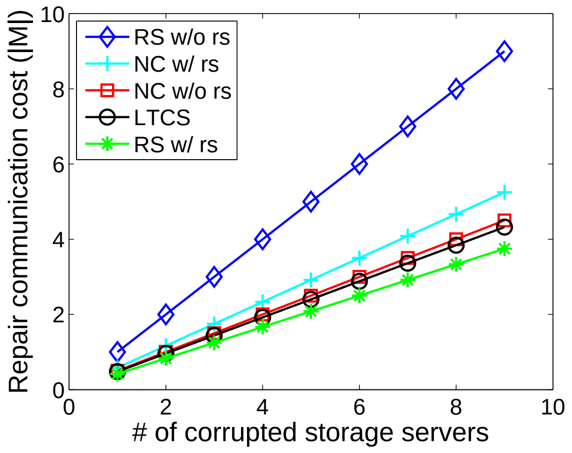


Figure 2.8: Communication cost of repair. n=12, k=3.

servers in one round is T, $0 \leq T \leq n-k$. The repair communication complexity in such scenario is $O(T\alpha \cdot |\mathcal{M}|/m)$, i.e., $O(T(1+\varepsilon)/k \cdot |\mathcal{M}|)$, where $|\mathcal{M}|/m$ is the size of each encoded packet.

Assume the corruption probability of the repair server is the same as storage servers, i.e., T/n. The total repair communication complexity is then calculated as $O(T(1+\varepsilon)/k \cdot |\mathcal{M}| + T/n \cdot |\mathcal{M}|)$, i.e., $O(T((1+\varepsilon)/k + 1/n) \cdot |\mathcal{M}|)$. As illustrated in Fig. 2.8, to repair different number of corrupted storage servers T, the communication cost in LTCS is only 15 percent more than Reed-Solomon codes-based service integrated with repair server, but smaller than that in network coding-based service.

2.7 Related work

2.7.1 Network Coding-based Distributed Storage

As a new data transmitting technique, network coding is different with traditional store-and-forward methods. Instead of simply forwarding previously received packets, network coding allows intermediate nodes to recode received packets before forwarding. It has been proved that random linear network coding over a sufficiently large finite field can achieve the multicast capacity [53, 82]. Since the data repair problem in the distributed storage is claimed to be mapped to a multicasting problem on the information flow graph [43], many network coding-based storage codes [42–44, 46, 66, 78, 90, 101, 121–125] have been proposed to take advantage of this property of capacity achievability. By recoding encoded packets in healthy servers during the repair procedure, the repair communication cost is reduced to the information theoretical minimum. The achievable region of functional repair is characterized in [37], but a large part of the achievable region of exact repair remains open [44]. Furthermore, since network coding utilizes Gaussian elimination decoding algorithm, the data retrieval is more expensive than erasure codes-based system [37]. Therefore, these designs are only suitable in "read-rarely" storage scenarios, and cannot be efficiently deployed in the cloud storage system where data retrieval is a routine operation.

2.7.2 Remote Data Integrity Check

The remote data integrity check problem has been explored in many works [12, 13, 25, 26, 34, 41, 47, 57, 71, 74, 85, 89, 105, 107, 109–113]. However, existing works do not have an efficient solution for the data repair problem or pay no attention to it. Portions of the work studied in this chapter were presented as extended abstract at the 31th IEEE Conference on Computer Communications (INFOCOM'12) [32].

Juels et al. [57] described a formal proof of retrievability (POR) model for ensuring the remote data integrity. Their scheme combines spot-checking and error-correcting code to ensure both possession and retrievability of files on archive service systems. Shacham et al. [87] built on this model and constructed a random linear function based homomorphic authenticator which enables unlimited number of challenges and requires less communication overhead due to its usage of relatively small size of BLS signature. Bowers et al. [26] proposed an improved framework for POR protocols that generalizes both Juels and Shachams work. Later in their subsequent work, Bowers et al. [25] extended POR model to distributed systems. However, all these schemes are focusing on static data. The effectiveness of their schemes rests primarily on the preprocessing steps that the user conducts before outsourcing the data file. Recently, Dodis et al. [45] gave theoretical studies on generalized framework for different variants of existing POR work.

Ateniese et al. [12] defined the provable data possession (PDP) model for ensuring possession of file on untrusted storages. Their scheme utilized public key based homomorphic tags for auditing the data file. In their subsequent work, Ateniese et al. [13] described a PDP scheme that uses only symmetric key based cryptography. This method introduces lower overhead than their previous scheme and allows for block updates, deletions and appends to the stored file. However, their scheme focuses on single server scenario and does not provide data availability guarantee against server failures, leaving both the distributed scenario and data error recovery issue unexplored. Wang et al. [112] proposed to combine BLS based homomorphic authenticator with Merkle Hash Tree to support fully data dynamics, while Erway et al. [47] developed a skip list based scheme to enable provable data possession with fully dynamics support. The incremental cryptography work done by Bellare et al. [19] also provides a set of cryptographic building blocks such as hash, MAC, and signature functions that may be employed for storage integrity verification while supporting dynamic operations on data. However, this branch of work falls into the traditional data integrity protection mechanism, where local copy of data has to be maintained for the verification. It is not yet clear how the work can be adapted to cloud storage scenario where users no longer have the data at local sites but still need to ensure the storage correctness efficiently in the cloud.

In other related work, Curtmola et al. [41] aimed to ensure data possession of multiple replicas across the distributed storage system. They extended the PDP scheme to cover multiple replicas without encoding each replica separately, providing guarantee that multiple copies of data are actually maintained. Lillibridge et al. [68] presented a P2P backup scheme which can detect data loss from free-riding peers, but does not ensure all data is unchanged. Filho et al. [49] proposed to verify data integrity using RSA-based hash to demonstrate uncheatable data possession in peer-to- peer file sharing networks. However, their proposal requires exponentiation over the entire data file, which is clearly impractical for the server whenever the file is large. Shah et al. [88, 89] proposed allowing a TPA to keep online storage honest by first encrypting the data then sending a number of pre-computed symmetric-keyed hashes over the encrypted data to the auditor. However, their scheme only works for encrypted files, and auditors must maintain long-term state. Schwarz et al. [85] proposed to ensure static file integrity across multiple distributed servers, using erasure-coding and block-level file integrity checks. Very recently, Wang et al. [107] gave a study on many existing solutions on remote data integrity checking, and discussed their pros and cons under different design scenarios of secure cloud storage services.

2.8 Conclusions

In this chapter, for the first time, we explore the problem of secure and reliable cloud storage with the efficiency consideration of both data repair and data retrieval, and design a LT codes-based cloud storage service (LTCS). To enable efficient decoding for data users in the data retrieval procedure, we adopt a low complexity LT codes for adding data redundancy in distributed cloud servers. Our proposed LTCS provides efficient data retrieval for data users by utilizing the fast Belief Propagation decoding algorithm, and releases the data owner from the burden of being online by enabling public data integrity check and employing exact repair. The performance analysis and experimental results show that LTCS has a comparable storage and communication cost, but a much faster data retrieval than the erasure codes-based solutions. It introduces less storage cost, much faster data retrieval, and comparable communication cost comparing to network coding-based storage services.

Chapter 3

Privacy-Preserving Multi-Keyword Ranked Search

3.1 Introduction

Cloud computing is the long dreamed vision of computing as a utility, where cloud customers can remotely store their data into the cloud so as to enjoy the on-demand high quality applications and services from a shared pool of configurable computing resources [31, 32, 103]. Its great flexibility and economic savings are motivating both individuals and enterprises to outsource their local complex data management system into the cloud. To protect data privacy and combat unsolicited accesses in the cloud and beyond, sensitive data, e.g., emails, personal health records, photo albums, tax documents, financial transactions, etc., may have to be encrypted by data owners before outsourcing to the commercial public cloud [58]; this, however, obsoletes the traditional data utilization service based on plaintext keyword search. The trivial solution of downloading all the data and decrypting locally is clearly impractical, due to the huge amount of bandwidth cost in cloud scale systems.

Moreover, aside from eliminating the local storage management, storing data into the cloud serves no purpose unless they can be easily searched and utilized. Thus, exploring privacy-preserving and effective search service over encrypted cloud data is of paramount importance. Considering the potentially large number of on-demand data users and huge amount of outsourced data documents in the cloud, this problem is particularly challenging as it is extremely difficult to meet also the requirements of performance, system usability and scalability.

On the one hand, to meet the effective data retrieval need, the large amount of documents demand the cloud server to perform result relevance ranking, instead of returning undifferentiated results. Such ranked search system enables data users to find the most relevant information quickly, rather than burdensomely sorting through every match in the content collection [96]. Ranked search can also elegantly eliminate unnecessary network traffic by sending back only the most relevant data, which is highly desirable in the "pay-as-you-use" cloud paradigm. For privacy protection, such ranking operation, however, should not leak any keyword related information. On the other hand, to improve the search result accuracy as well as to enhance the user searching experience, it is also necessary for such ranking system to support multiple keywords search, as single keyword search often yields far too coarse results. As a common practice indicated by today's web search engines (e.g., Google search), data users may tend to provide a set of keywords instead of only one as the indicator of their search interest to retrieve the most relevant data. And each keyword in the search request is able to help narrow down the search result further. "Coordinate matching" [119], i.e., as many matches as possible, is an efficient similarity measure among such multi-keyword semantics to refine the result relevance, and has been widely used in the plaintext information retrieval (IR) community. However, how to apply it in the encrypted cloud data search system remains a very challenging task because of inherent security and privacy obstacles, including various strict requirements like the data privacy, the index privacy, the keyword privacy, and many others (see section 3.3.2).

In the literature, searchable encryption [7,18,22,23,35,40,51,65,86,98] is a helpful technique that treats encrypted data as documents and allows a user to securely search through a single keyword and retrieve documents of interest. However, direct application of these approaches to the secure large scale cloud data utilization system would not be necessarily suitable, as they are developed as crypto primitives and cannot accommodate such high service-level requirements like system usability, user searching experience, and easy information discovery. Although some recent designs have been proposed to support Boolean keyword search [16, 24, 29, 52, 55, 59, 63, 67, 93] as an attempt to enrich the search flexibility, they are still not adequate to provide users with acceptable result ranking functionality (see section 3.7). Our early works [104, 106] have been aware of this problem, and provide solutions to the secure ranked search over encrypted data problem but only for queries consisting of a single keyword. How to design an efficient encrypted data search mechanism that supports multi-keyword semantics without privacy breaches still remains a challenging open problem.

In this chapter, for the first time, we define and solve the problem of multi-keyword ranked search over encrypted cloud data (MRSE) while preserving strict system-wise privacy in the cloud computing paradigm. Among various multi-keyword semantics, we choose the efficient similarity measure of "coordinate matching", i.e., as many matches as possible, to capture the relevance of data documents to the search query. Specifically, we use "inner product similarity" [119], i.e., the number of query keywords appearing in a document, to quantitatively evaluate such similarity measure of that document to the search query. During the index construction, each document is associated with a binary vector as a subindex where each bit represents whether corresponding keyword is contained in the document. The search query is also described as a binary vector where each bit means whether corresponding keyword appears in this search request, so the similarity could be exactly measured by the inner product of the query vector with the data vector. However, directly outsourcing the data vector or the query vector will violate the index privacy or the search privacy. To meet the challenge of supporting such multi-keyword semantic without privacy breaches, we propose a basic idea for the MRSE using secure inner product computation, which is adapted from a secure k-nearest neighbor (kNN) technique [120], and then give two significantly improved MRSE schemes in a step-by-step manner to achieve various stringent privacy requirements in two threat models with increased attack capabilities. Our contributions are summarized as follows,

1. For the first time, we explore the problem of multi-keyword ranked search over encrypted cloud data, and establish a set of strict privacy requirements for such a secure cloud data utilization system.

2. We propose two MRSE schemes based on the similarity measure of "coordinate matching" while meeting different privacy requirements in two different threat models.

3. We investigate some further enhancements of our ranked search mechanism to support more search semantics and dynamic data operations.

4. Thorough analysis investigating privacy and efficiency guarantees of the proposed schemes is given, and experiments on the real-world dataset further show the proposed schemes indeed introduce low overhead on computation

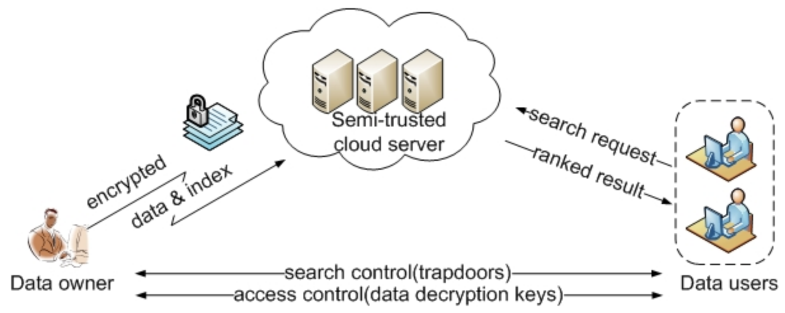

Semi-trusted cloud server
encrypted
data & index
search request
ranked result
Data owner
search control(trapdoors)
access control(data decryption keys)
Data users

Figure 3.1: Architecture of the search over encrypted cloud data

and communication.

The remainder of this chapter is organized as follows. In Section 3.2, we introduce the system model, the threat model, our design goals, and the preliminary. Section 3.3 describes the MRSE framework and privacy requirements, followed by section 3.4, which describes the proposed schemes. Section 3.5 discusses supporting more search semantics and dynamic operation. Section 3.6 presents simulation results. We discuss related work on both single and Boolean keyword searchable encryption in Section 3.7, and conclude the chapter in Section 3.8.

3.2 Problem Formulation

3.2.1 System Model

Considering a cloud data hosting service involving three different entities, as illustrated in Fig. 3.1: the data owner, the data user, and the cloud server. The data owner has a collection of data documents $\mathcal{F}$ to be outsourced to the cloud server in the encrypted form $\mathcal{C}$. To enable the searching capability over $\mathcal{C}$ for effective data utilization, the data owner, before outsourcing, will first build an encrypted searchable index $\mathcal{I}$ from $\mathcal{F}$, and then outsource both the index $\mathcal{I}$ and the encrypted document collection $\mathcal{C}$ to the cloud server. To search the document collection for t given keywords, an authorized user acquires a corresponding trapdoor T through search control mechanisms, e.g., broadcast encryption [40]. Upon receiving T from a data user, the cloud server is responsible to search the index $\mathcal{I}$ and return the corresponding set of encrypted documents. To improve the document retrieval accuracy, the search result should be ranked by the cloud server according to some ranking criteria (e.g., coordinate matching, as will be introduced shortly). Moreover, to reduce the communication cost, the data user may send an optional number k along with the trapdoor T so that the cloud server only sends back top-k documents that are most relevant to the search query. Finally, the access control mechanism [129] is employed to manage decryption capabilities given to users and the the data collection can be updated in terms of inserting new documents, updating existing documents and deleting existing documents.

3.2.2 Threat Model

The cloud server is considered as "honest-but-curious" in our model, which is consistent with related works on cloud security [111, 129]. Specifically, the cloud server acts in an "honest" fashion and correctly follows the designated protocol specification. However, it is "curious" to infer and analyze data (including index) in its storage and message flows received during the protocol so as to learn additional information. Based on what information the cloud server knows, we consider two threat models with different attack capabilities as follows.

Known Ciphertext Model In this model, the cloud server is supposed to only know encrypted dataset $\mathcal{C}$ and searchable index $\mathcal{I}$, both of which are outsourced from the data owner.

Known Background Model In this stronger model, the cloud server is supposed to possess more knowledge than what can be accessed in the known ciphertext model. Such information may include the correlation relationship of given search requests (trapdoors), as well as the dataset related statistical information. As an instance of possible attacks in this case, the cloud server could use the known trapdoor information combined with document/keyword frequency [131] to deduce/identify certain keywords in the query.

3.2.3 Design Goals

To enable ranked search for effective utilization of outsourced cloud data under the aforementioned model, our system design should simultaneously achieve security and performance guarantees as follows.

- **Multi-keyword Ranked Search:** To design search schemes which allow multi-keyword query and provide result similarity ranking for effective data retrieval, instead of returning undifferentiated results.
- **Privacy-Preserving:** To prevent the cloud server from learning additional information from the dataset and the index, and to meet privacy requirements specified in section 3.3.2.
- **Efficiency:** Above goals on functionality and privacy should be achieved with low communication and computation overhead.

3.2.4 Notations

- $\mathcal{F}$ – the plaintext document collection, denoted as a set of m data documents $\mathcal{F} = (F_1, F_2, \ldots, F_m)$.

- $\mathcal{C}$ – the encrypted document collection stored in the cloud server, denoted as $\mathcal{C} = (C_1, C_2, \ldots, C_m)$.
- $\mathcal{W}$ – the dictionary, i.e., the keyword set consisting of n keyword, denoted as $\mathcal{W} = (W_1, W_2, \ldots, W_n)$.
- $\mathcal{I}$ – the searchable index associated with $\mathcal{C}$, denoted as $(I_1, I_2, \ldots, I_m)$ where each subindex I_i is built for F_i.
- $\widetilde{\mathcal{W}}$ – the subset of $\mathcal{W}$, representing the keywords in a search request, denoted as $\widetilde{\mathcal{W}} = (W_{j_1}, W_{j_2}, \ldots, W_{j_t})$.
- $T_{\widetilde{\mathcal{W}}}$ – the trapdoor for the search request $\widetilde{\mathcal{W}}$.
- $\mathcal{F}_{\widetilde{\mathcal{W}}}$ – the ranked id list of all documents according to their relevance to $\widetilde{\mathcal{W}}$.

3.2.5 Preliminary on Coordinate Matching

As a hybrid of conjunctive search and disjunctive search, "coordinate matching" [119] is an intermediate similarity measure which uses the number of query keywords appearing in the document to quantify the relevance of that document to the query. When users know the exact subset of the dataset to be retrieved, Boolean queries perform well with the precise search requirement specified by the user. In cloud computing, however, this is not the practical case, given the huge amount of outsourced data. Therefore, it is more flexible for users to specify a list of keywords indicating their interest and retrieve the most relevant documents with a rank order.

3.3 Framework and Privacy Requirements for MRSE

In this section, we define the framework of multi-keyword ranked search over encrypted cloud data (MRSE) and establish various strict system-wise privacy requirements for such a secure cloud data utilization system.

3.3.1 MRSE Framework

For easy presentation, operations on the data documents are not shown in the framework since the data owner could easily employ the traditional symmetric key cryptography to encrypt and then outsource data. With focus on the index and query, the MRSE system consists of four algorithms as follows.

- $\mathsf{Setup}(1^{\ell})$ *Taking a security parameter ℓ as input, the data owner outputs a symmetric key as SK.*

- $\mathsf{BuildIndex}(\mathcal{F}, SK)$ *Based on the dataset $\mathcal{F}$, the data owner builds a searchable index $\mathcal{I}$ which is encrypted by the symmetric key SK and then outsourced to the cloud server. After the index construction, the document collection can be independently encrypted and outsourced.*

- $\mathsf{Trapdoor}(\widetilde{\mathcal{W}})$ *With t keywords of interest in $\widetilde{\mathcal{W}}$ as input, this algorithm generates a corresponding trapdoor $T_{\widetilde{\mathcal{W}}}$.*

- $\mathsf{Query}(T_{\widetilde{\mathcal{W}}}, k, \mathcal{I})$ *When the cloud server receives a query request as ($T_{\widetilde{\mathcal{W}}}$, k), it performs the ranked search on the index $\mathcal{I}$ with the help of trapdoor $T_{\widetilde{\mathcal{W}}}$, and finally returns $\mathcal{F}_{\widetilde{\mathcal{W}}}$, the ranked id list of top-k documents sorted by their similarity with $\widetilde{\mathcal{W}}$.*

Neither the search control nor the access control is within the scope of this dissertation. While the former is to regulate how authorized users acquire trapdoors, the later is to manage users' access to outsourced documents.

3.3.2 Privacy Requirements for MRSE

The representative privacy guarantee in the related literature, such as searchable encryption, is that the server should learn nothing but search results. With this general privacy description, we explore and establish a set of strict privacy requirements specifically for the MRSE framework.

As for the *data privacy*, the data owner can resort to the traditional symmetric key cryptography to encrypt the data before outsourcing, and successfully prevent the cloud server from prying into the outsourced data. With respect to the *index privacy*, if the cloud server deduces any association between keywords and encrypted documents from index, it may learn the major subject of a document, even the content of a short document [131]. Therefore, the searchable index should be constructed to prevent the cloud server from performing such kind of association attack. While data and index privacy guarantees are demanded by default in the related literature, various *search privacy* requirements involved in the query procedure are more complex and difficult to tackle as follows.

Keyword Privacy As users usually prefer to keep their search from being exposed to others like the cloud server, the most important concern is to hide what they are searching, i.e., the keywords indicated by the corresponding trapdoor. Although the trapdoor can be generated in a cryptographic way to protect the query keywords, the cloud server could do some statistical analysis over the search result to make an estimate. As a kind of statistical information, *document frequency* (i.e., the number of documents containing the keyword) is sufficient to identify the keyword with high probability [130]. When the cloud server knows some background information of the dataset, this keyword specific information may be utilized to reverse-engineer the keyword.

Trapdoor Unlinkability The trapdoor generation function should be a randomized one instead of being deterministic. In particular, the cloud server should not be able to deduce the relationship of any given trapdoors, e.g., to determine whether the two trapdoors are formed by the same search request. Otherwise, the deterministic trapdoor generation would give the cloud server advantage to accumulate frequencies of different search requests regarding different keyword(s), which may further violate the aforementioned keyword privacy requirement. So the fundamental protection for trapdoor unlinkability is to introduce sufficient nondeterminacy into the trapdoor generation procedure.

Access Pattern Within the ranked search, the access pattern is the sequence of search results where every search result is a set of documents with rank order. Specifically, the search result for the query keyword set $\widetilde{\mathcal{W}}$ is denoted as $\mathcal{F}_{\widetilde{\mathcal{W}}}$, consisting of the id list of all documents ranked by their relevance to $\widetilde{\mathcal{W}}$. Then the access pattern is denoted as $(\mathcal{F}_{\widetilde{\mathcal{W}}_1}, \mathcal{F}_{\widetilde{\mathcal{W}}_2}, \ldots)$ which are the results of sequential searches. Although a few searchable encryption works, e.g., [29] has been proposed to utilize private information retrieval (PIR) technique [56, 62, 75, 75, 114, 118], to hide the access pattern, our proposed schemes are not designed to protect the access pattern for the efficiency concerns. This is because any PIR based technique must “touch” the whole dataset outsourced on the server which is inefficient in the large scale cloud system.

3.4 Privacy-Preserving and Efficient MRSE

To efficiently achieve multi-keyword ranked search, we propose to employ "inner product similarity" [119] to quantitatively evaluate the efficient similarity measure "coordinate matching". Specifically, D_i is a binary data vector for document F_i where each bit $D_i[j] \in \{0, 1\}$ represents the existence of the corresponding keyword W_j in that document, and Q is a binary query vector indicating the keywords of interest where each bit $Q[j] \in \{0, 1\}$ represents the existence of the corresponding keyword W_j in the query $\widetilde{\mathcal{W}}$. The similarity score of document F_i to query $\widetilde{\mathcal{W}}$ is therefore expressed as the inner product of their binary column vectors, i.e., $D_i \cdot Q$. For the purpose of ranking, the cloud server must be given the capability to compare the similarity of different documents to the query. But, to preserve strict system-wise privacy, data vector D_i, query vector Q and their inner product $D_i \cdot Q$ should not be exposed to the cloud server. In this section, we first propose a basic idea for the MRSE using secure inner product computation, which is adapted from a secure k-nearest neighbor (kNN) technique, and then show how to significantly improve it to be privacy-preserving against different threat models in the MRSE framework in a step-by-step manner.

3.4.1 Secure Inner Product Computation

3.4.1.1 Secure kNN Computation

In the secure k-nearest neighbor (kNN) scheme [120], Euclidean distance between a data record p_i and a query vector q is used to select k nearest database records. The secret key is composed of one $(d+1)$-bit vector as S and two $(d+1) \times (d+1)$ invertible matrices as $\{M_1, M_2\}$, where d is the number of fields for each record p_i. First, every data vector p_i and query vector q are extended to $(d+1)$-dimension vectors as $\vec{p_i}$ and

$\vec{q}$, where the $(d+1)$-th dimension is set to $-0.5||p_i^2||$ and 1, respectively. Besides, the query vector $\vec{q}$ is scaled by a random number $r > 0$ as (rq, r). Then, $\vec{p_i}$ is split into two random vectors as $\{\vec{p_i}', \vec{p_i}''\}$, and $\vec{q}$ is also split into two random vectors as $\{\vec{q}\,', \vec{q}\,''\}$. Note here that vector S functions as a splitting indicator. Namely, if the j-th bit of S is 0, $\vec{p_i}'[j]$ and $\vec{p_i}''[j]$ are set as the same as $\vec{p_i}[j]$, while $\vec{q}\,'[j]$ and $\vec{q}\,''[j]$ are set to two random numbers so that their sum is equal to $\vec{q}[j]$; if the j-th bit of S is 1, the splitting process is similar except that $\vec{p_i}$ and $\vec{q}$ are switched. The split data vector pair $\{\vec{p_i}', \vec{p_i}''\}$ is encrypted as $\{p_{ia}, p_{ib}\}$, where $p_{ia} = M_1^T \vec{p_i}'$ and $p_{ib} = M_2^T \vec{p_i}''$; the split query vector pair $\{\vec{q}\,', \vec{q}\,''\}$ is encrypted as $\{q_a, q_b\}$, where $q_a = M_1^{-1}\vec{q}\,'$ and $q_b = M_2^{-1}\vec{q}\,''$. In the query step, the product of data vector pair and query vector pair, i.e., $-0.5r(||p_i||^2 - 2p_i \cdot q)$, is serving as the indicator of Euclidean distance $(||p_i||^2 - 2p_i \cdot q + ||q||^2)$ to select k nearest neighbors.

Security Analysis in Known Ciphertext Model Similarly with [120], let the knowledge of the attacker be the encrypted data record and query vector. For any data record p_i, by definition, the attacker knows the encrypted values $\{p_{ia}, p_{ib}\}$. If the attacker does not know the splitting configuration, he has to model as two random (d+1)-dimensional vectors. The equations for solving the transformation matrices are $M_1^T \vec{p_i}' = p_{ia}$ and $M_2^T \vec{p_i}'' = p_{ib}$, where M_1 and M_2 are two $(d+1) \times (d+1)$ unknown matrices. There are $2(d+1)$ unknowns in p_{ia} and p_{ib} and $2(d+1)^2$ unknowns in M_1 and M_2. Since there are only $2(d+1)$ equations, which are less than the number of unknowns, the attacker does not have sufficient information to solve for the transformation matrices. Hence, we believe this kNN computation scheme is secure in the known ciphertext model.

3.4.1.2 Secure Inner Product Computation

As the MRSE is using the inner product similarity instead of the Euclidean distance, we need to do some modifications on the secure kNN computation scheme to fit the MRSE framework. One way to do that is by eliminating the dimension extension, the final result changes to be the inner product as $rp_i \cdot q$.

Efficiency Analysis While the encryption of either data record or query vector involves two multiplications of a $d \times d$ matrix and a d-dimension vector with complexity $O(d^2)$, the final inner product computation involves two multiplications of two d-dimension vectors with complexity $O(d)$.

Security Analysis In the known ciphertext model, the splitting vector S is unknown, so $\vec{p_i}'$ and $\vec{p_i}''$ are considered as two random d-dimensional vectors. To solve the linear equations created by the encryption of data vectors, we have $2dm$ unknowns in m data vectors and $2d^2$ unknowns in $\{M_1, M_2\}$. Since we have only $2dm$ equations, which are less than the number of unknowns, there is no sufficient information to solve either data vectors or $\{M_1, M_2\}$. Similarly, $\vec{q}\,'$ and $\vec{q}\,''$ are also considered as two random d-dimensional vectors. To solve the linear equations created by the encryption of query vectors, we have $2d$ unknowns in two query vectors and $2d^2$ unknowns in $\{M_1, M_2\}$. Since we have only $2d$ equations here, which are less than the number of unknowns, there is no sufficient information to solve either query vectors or $\{M_1, M_2\}$. Therefore, we believe that without prior knowledge of secret key, neither data vector nor query vector, after such a series of processes like splitting and multiplication, can be recovered by analyzing their corresponding ciphertexts.

3.4.2 MRSE_I: Privacy-Preserving Scheme in Known Ciphertext Model

The adapted secure inner product computation scheme is not good enough for our MRSE design. The major reason is that the only randomness involved is the scale factor r in the trapdoor generation, which does not provide sufficient nondeterminacy in the overall scheme as required by the trapdoor unlinkability requirement as well as the keyword privacy requirement. To provide a more advanced design for the MRSE, we now provide our MRSE_I scheme as follows.

3.4.2.1 MRSE_I Scheme

In our more advanced design, instead of simply removing the extended dimension in the query vector as we plan to do at the first glance, we preserve this dimension extending operation but assign a new random number t to the extended dimension in each query vector. Such a newly added randomness is expected to increase the difficulty for the cloud server to learn the relationship among the received trapdoors. In addition, as mentioned in the keyword privacy requirement, randomness should also be carefully calibrated in the search result to obfuscate the document frequency and diminish the chances for re-identification of keywords. Introducing some randomness in the final similarity score is an effective way towards what we expect here. More specifically, unlike the randomness involved in the query vector, we insert a dummy keyword into each data vector and assign a random value to it. Each individual vector D_i is extended to $(n+2)$-dimension instead of $(n+1)$, where a random variable ε_i representing the dummy keyword is stored in the extended dimension. The whole scheme to achieve ranked search with multiple keywords over encrypted data is as follows.

- Setup The data owner randomly generates a $(n+2)$-bit vector as S and two $(n+2) \times (n+2)$ invertible matrices $\{M_1, M_2\}$. The secret key SK is in the form of a 3-tuple as $\{S, M_1, M_2\}$.

- BuildIndex$(\mathcal{F}, SK)$ The data owner generates a binary data vector D_i for every document F_i, where each binary bit $D_i[j]$ represents whether the corresponding keyword W_j appears in the document F_i. Subsequently, every plaintext subindex $\vec{D_i}$ is generated by applying dimension extending and splitting procedures on D_i. These procedures are similar with those in the secure kNN computation except that the $(n+1)$-th entry in $\vec{D_i}$ is set to a random number ε_i, and the $(n+2)$-th entry in $\vec{D_i}$ is set to 1 during the dimension extending. $\vec{D_i}$ is therefore equal to $(D_i, \varepsilon_i, 1)$. Finally, the subindex $I_i = \{M_1^T \vec{D_i}', M_2^T \vec{D_i}''\}$ is built for every encrypted document C_i.

- Trapdoor$(\widetilde{\mathcal{W}})$ With t keywords of interest in $\widetilde{\mathcal{W}}$ as input, one binary vector Q is generated where each bit $Q[j]$ indicates whether $W_j \in \widetilde{\mathcal{W}}$ is true or false. Q is first extended to $n+1$-dimension which is set to 1, and then scaled by a random number $r \neq 0$, and finally extended to a $(n+2)$-dimension vector as $\vec{Q}$ where the last dimension is set to another random number t. $\vec{Q}$ is therefore equal to (rQ, r, t). After applying the same splitting and encrypting processes as above, the trapdoor $T_{\widetilde{\mathcal{W}}}$ is generated as $\{M_1^{-1}\vec{Q}', M_2^{-1}\vec{Q}''\}$.

- Query$(T_{\widetilde{\mathcal{W}}}, k, \mathcal{I})$ With the trapdoor $T_{\widetilde{\mathcal{W}}}$, the cloud server computes the similarity scores of each document F_i as in equation 3.1. WLOG, we assume $r > 0$. After sorting all scores, the cloud server returns the top-k ranked id list $\mathcal{F}_{\widetilde{\mathcal{W}}}$.

With t brought into the query vector and ε_i brought into each data vector, the final similarity scores would be:

$$\begin{aligned} I_i \cdot T_{\widetilde{\mathcal{W}}} &= \{M_1^T \vec{D}_i{}', M_2^T \vec{D}_i{}''\} \cdot \{M_1^{-1}\vec{Q}', M_2^{-1}\vec{Q}''\} \\ &= \vec{D}_i{}' \cdot \vec{Q}\,' + \vec{D}_i{}'' \cdot \vec{Q}\,'' \\ &= \vec{D}_i \cdot \vec{Q} \\ &= (D_i, \varepsilon_i, 1) \cdot (rQ, r, t) \\ &= r(D_i \cdot Q + \varepsilon_i) + t. \end{aligned} \tag{3.1}$$

Note that in the original case, the final score is simply $rD_i \cdot Q$, which preserves the scale relationship for two queries on the same keywords. But such an issue is no longer valid in our improved scheme due to the randomness of both t and ε_i, which clearly demonstrates the effectiveness and improved security strength of our MSRE_I mechanism.

3.4.2.2 Analysis

We analyze this MRSE_I scheme from three aspects of design goals described in section 3.2.

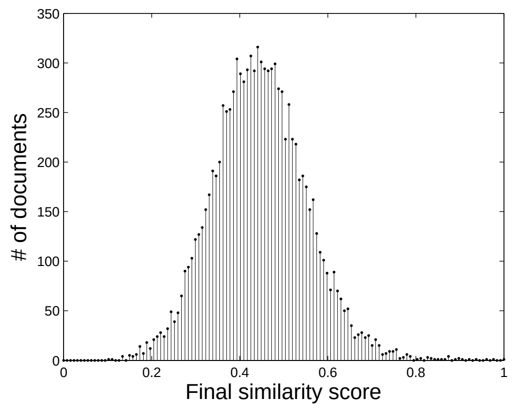
Functionality and Efficiency Assume the number of query keywords appearing in a document F_i is $x_i = D_i \cdot Q$. From equation 3.1, the final similarity score as $y_i = I_i \cdot T_{\widetilde{\mathcal{W}}} = r(x_i + \varepsilon_i) + t$ is a linear function of x_i, where the coefficient r is set as a positive random number. However, because the random factor ε_i is introduced as a part of the similarity score, the final search result on the basis of sorting similarity scores may not be as accurate as that in original scheme. For the consideration of search accuracy, we can let ε_i follow a normal distribution $N(\mu, \sigma^2)$, where the standard deviation σ functions as a flexible trade-off parameter among search accuracy and security. From the consideration of effectiveness, σ is expected to be smaller so as to obtain high precision indicating the good purity of retrieved documents. To quantitatively evaluate the search accuracy, we set a measure as precision P_k to capture the fraction of returned top-k documents that are included in the real top-k list. Detailed accuracy evaluation on the real-world dataset will be given in section 3.6.

As for the efficiency, our inner product based MRSE scheme is an outstanding approach from the performance perspective. In the steps like BuildIndex or Trapdoor, the generation procedure of each subindex or trapdoor involves two multiplications of a $(n+2) \times (n+2)$ matrix and a $(n+2)$-dimension vector with complexity $O(n^2)$. In the Query, the final similarity score is computed through two multiplications of two $(n+2)$-dimension vectors with complexity $O(n)$.

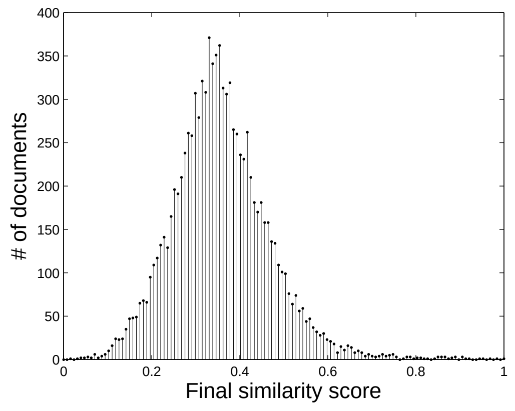
Privacy As for the *data privacy*, traditional symmetric key encryption techniques could be properly utilized here and is not within the scope of this dissertation.

The *index privacy* is well protected if the secret key SK is kept confidential since such vector encryption method has been proved to be secure in the known ciphertext model [120]. We add two more dimensions to the vectors compared to the adapted secure inner product computation described in Section 3.4.1.2. In the encryption of data vectors, the number of equations as $2(n+2)m$ in $M_1^T \vec{D_i}' = I_i'$ and $M_2^T \vec{D_i}'' = I_i''$ is still less than the number of unknowns as the sum of $2(n+2)m$ unknowns in m data vectors and $2(n+2)^2$ unknowns in $\{M_1, M_2\}$. As a result, the attacker cannot solve the equations. Note that the addition of dimensions will only increase the security of the scheme [120].

With the randomness introduced by the splitting process and the random numbers r, and t, our basic scheme can generate two totally different trapdoors for the same query $\widetilde{\mathcal{W}}$. This nondeterministic trapdoor generation can guarantee the *trap-*



(a) $\sigma = 1$



(b) $\sigma = 0.5$

Figure 3.2: Distribution of final similarity score with different standard deviations, 10k documents, 10 query keywords.

door unlinkability which is an unsolved privacy leakage problem in related symmetric key based searchable encryption schemes because of the deterministic property of trapdoor generation [40]. Moreover, with properly selected parameter σ for the random factor ε_i, even the final score results can be obfuscated very well, preventing the cloud server from learning the relationships of given trapdoors and the corresponding keywords. Note that although σ is expected to be small from the effectiveness point of view, the small one will introduce small obfuscation into the the final similarity scores, which may weaken the protection of keyword privacy and trapdoor unlinkability. As shown in Fig. 3.2, the distribution of the final similarity scores with smaller σ will enable the cloud server to learn more statistical information about the original similarity scores, and therefore σ should be set large enough from the consideration of privacy.

3.4.3 MRSE_II: Privacy-Preserving Scheme in Known Background Model

When the cloud server has knowledge of some background information on the outsourced dataset, e.g., the correlation relationship of two given trapdoors, certain keyword privacy may not be guaranteed anymore by the MRSE_I scheme. This is possible in the known background model because the cloud server can use scale analysis as follows to deduce the keyword specific information, e.g., document frequency, which can be further combined with background information to identify the keyword in a query at high probability. After presenting how the cloud server uses scale analysis attack to break the keyword privacy, we propose a more advanced MRSE scheme to be privacy-preserving in the known background model.

3.4.3.1 Scale Analysis Attack

Given two correlated trapdoors T_1 and T_2 for query keywords $\{K_1, K_2\}$ and $\{K_1, K_2, K_3\}$ respectively, there will be two special cases when searching on any three documents as listed in Tab. 3.1 and Tab. 3.2. In any of these two cases, there exists a system

Table 3.1: K_3 appears in every document

Doc	Query for $\{K_1, K_2, K_3\}$	Query for $\{K_1, K_2\}$
1	$x_1 = 3, y_1 = r(3 + \varepsilon_1) + t$	$x'_1 = 2, y'_1 = r'(2 + \varepsilon_1) + t'$
2	$x_2 = 2, y_2 = r(2 + \varepsilon_2) + t$	$x'_2 = 1, y'_2 = r'(1 + \varepsilon_2) + t'$
3	$x_3 = 1, y_3 = r(1 + \varepsilon_3) + t$	$x'_3 = 0, y'_3 = r'(0 + \varepsilon_3) + t'$

of equations among final similarity scores y_i for T_1 and y'_i for T_2 as follows,

$$\begin{cases} y_1 - y_2 = & r(1 + \varepsilon_1 - \varepsilon_2); \\ y'_1 - y'_2 = & r'(1 + \varepsilon_1 - \varepsilon_2); \\ y_2 - y_3 = & r(1 + \varepsilon_2 - \varepsilon_3); \\ y'_2 - y'_3 = & r'(1 + \varepsilon_2 - \varepsilon_3); \\ y_1 - y_3 = & r(2 + \varepsilon_1 - \varepsilon_3); \\ y'_1 - y'_3 = & r'(2 + \varepsilon_1 - \varepsilon_3). \end{cases} \tag{3.2}$$

To this end, although the exact value of x_i is encrypted as y_i, the cloud server could deduce that whether all the three documents contain K_3 or none of them contain K_3 through checking the following equivalence relationship among all final similarity scores in two queries,

$$\frac{y_1 - y_2}{y'_1 - y'_2} = \frac{y_2 - y_3}{y'_2 - y'_3} = \frac{y_1 - y_3}{y'_1 - y'_3}. \tag{3.3}$$

By extending three documents to the whole dataset, the cloud server could further deduce two possible values of document frequency of keyword K_3. In the known background model, the server can identify the keyword K_3 by referring to the keyword specific document frequency information about the dataset.

Table 3.2: K_3 does not appear in either document

Doc	Query for $\{K_1, K_2, K_3\}$	Query for $\{K_1, K_2\}$
1	$x_1 = 2, y_1 = r(2 + \varepsilon_1) + t$	$x'_1 = 2, y'_1 = r'(2 + \varepsilon_1) + t'$
2	$x_2 = 1, y_2 = r(1 + \varepsilon_2) + t$	$x'_2 = 1, y'_2 = r'(1 + \varepsilon_2) + t'$
3	$x_3 = 0, y_3 = r(0 + \varepsilon_3) + t$	$x'_3 = 0, y'_3 = r'(0 + \varepsilon_3) + t'$

3.4.3.2 MRSE_II Scheme

The privacy leakage shown above is caused by the fixed value of random variable ε_i in data vector D_i. To eliminate such fixed property in any specific document, more dummy keywords instead of only one should be inserted into every data vector D_i. All the vectors are extended to $(n+U+1)$-dimension instead of $(n+2)$, where U is the number of dummy keywords inserted. Improved details in the MRSE_II scheme is presented as follows.

- $\mathsf{Setup}(1^n)$ The data owner randomly generates a $(n + U + 1)$-bit vector as S and two $(n + U + 1) \times (n + U + 1)$ invertible matrices $\{M_1, M_2\}$.

- $\mathsf{BuildIndex}(\mathcal{F}, SK)$ The $(n + j + 1)$-th entry in $\vec{D}_i$ where $j \in [1, U]$ is set to a random number $\varepsilon^{(j)}$ during the dimension extending.

- $\mathsf{Trapdoor}(\widetilde{\mathcal{W}})$ By randomly selecting V out of U dummy keywords, the corresponding entries in Q are set to 1.

- $\mathsf{Query}(T_{\widetilde{\mathcal{W}}}, k, \mathcal{I})$ The final similarity score computed by cloud server is equal to $r(x_i + \sum \varepsilon_i^{(v)}) + t_i$ where the v-th dummy keyword is included in the V selected ones.

3.4.3.3 Analysis

Assume the probability of two $\sum \varepsilon_i^{(v)}$ having the same value should be less than $1/2^\omega$, it then means there should be at least 2^ω different values of $\sum \varepsilon_i^{(v)}$ for each data vector. The number of different $\sum \varepsilon_i^{(v)}$ is not larger than $\binom{U}{V}$, which is maximized when $\frac{U}{V} = 2$. Besides, considering $\binom{U}{V} \geq (\frac{U}{V})^V = 2^V$, it is greater than 2^ω when $U = 2\omega$ and $V = \omega$. So every data vector should include at least 2ω dummy entries, and every query vector will randomly select half dummy entries. Here ω can be considered as a system parameter for the tradeoff between efficiency and privacy. With properly setting the value of ω, the MRSE_II scheme is secure against scale analysis attack, and provides various expected privacy guarantees within the known ciphertext model or the known background model.

Moreover, every $\varepsilon^{(j)}$ is assumed to follow the same uniform distribution $M(\mu' - c, \mu' + c)$, where the mean is μ' and the variance as σ'^2 is $c^2/3$. According to the central limit theorem, the sum of ω independent random variables $\varepsilon^{(j)}$ follows the Normal distribution, where the mean is $\omega\mu'$ and the variance is $\omega\sigma'^2 = \omega c^2/3$. To make $\sum \varepsilon_i^{(v)}$ follow the Normal distribution $N(\mu, \sigma^2)$ as above, the value of μ' is set as μ/ω and the value of c is set as $\sqrt{\frac{3}{\omega}}\sigma$ so that $\omega\mu' = \mu$ and $\omega\sigma'^2 = \sigma^2$. With such parameter setting, search accuracy is statistically the same as that in MRSE_I scheme.

3.5 Discussion

3.5.1 Supporting More Search Semantics

In the ranking principle "coordinate matching", the presence of keyword in the document or the query is shown as 1 in the data vector or the query vector. Actually, there are more factors which could make impact on the search usability. For example, when one keyword appears in most documents in the dataset, the importance of this keyword in the query is less than other keywords which appears in less documents. Similarly, if one document contains a query keyword in multiple locations, the user may prefer this to the other document which contains the query keyword in only one location. To capture these information in the search process, we use the TF $\times$ IDF weighting rule within the vector space model to calculate the similarity, where TF (or term frequency) is the number of times a given term or keyword (we will use them interchangeably hereafter) appears within a file (to measure the importance of the term within the particular file), and IDF (or inverse document frequency) is obtained by dividing the number of files in the whole collection by the number of files containing the term (to measure the overall importance of the term within the whole collection). Among several hundred variations of the TF $\times$ IDF weighting scheme, no single combination of them outperforms any of the others universally [134]. Thus, without loss of generality, we choose an example formula that is commonly used and widely seen in the literature (see Chapter 4 in [96]) for the relevance score calculation,

$$Score(F_i, Q) = \frac{1}{|F_i|} \sum_{W_j \in \widetilde{\mathcal{W}}} (1 + \ln f_{i,j}) \cdot \ln(1 + \frac{m}{f_j}). \tag{3.4}$$

Here $f_{i,j}$ denotes the TF of keyword W_j in file F_i; f_j denotes the number of files that contain keyword W_j which is called document frequency; m denotes the total number of files in the collection; and $|F_i|$ is the Euclidean length of file F_i, obtained by $\sqrt{\sum_{j=1}^{n} (1 + \ln f_{i,j})^2}$, functioning as the normalization factor.

In order to calculate the relevance score as shown in Eq. 3.4 on the server side, we propose a new search mechanism MRSE_I_TF as follows which modify related data structures in the previous scheme MRSE_I. As for the dictionary $\mathcal{W}$, the document frequency f_j is attached to every keyword W_j, which will be used in the generation of query vector. In BuildIndex, for every keyword W_j appearing in the document F_i, the corresponding entry $D_i[j]$ in the data vector D_i is changed from a binary value 1 to the normalized term frequency, i.e., $\frac{1+\ln f_{i,j}}{|F_i|}$. Similarly, the query vector Q changes corresponding entries from 1 to $\ln(1+\frac{m}{f_j})$. Finally, the similarity score is as follows,

$$\begin{aligned} I_i \cdot T_{\widetilde{\mathcal{W}}} &= r(D_i \cdot Q + \varepsilon_i) + t \\ &= r(\sum_{W_j \in Q} \frac{1+\ln f_{i,j}}{|F_i|} \cdot \ln(1+\frac{m}{f_j}) + \varepsilon_i) + t \\ &= r(Score(F_i, Q) + \varepsilon_i) + t. \end{aligned} \tag{3.5}$$

Therefore, the similarity of the document and the query in terms of the cosine of the angle between the document vector and the query vector could be evaluated by computing the inner product of subindex I_i and trapdoor $T_{\widetilde{\mathcal{W}}}$. Although this similarity measurement introduces more computation cost during the index construction and trapdoor generation, it captures more related information on the content of documents and query which returns better results of users' interest. As we will see in section 3.6, the additional cost of this measurement in BuildIndex and Trapdoor is relatively small compared to the whole cost. Besides, BuildIndex is a one-time computation for the whole scheme.

Here, although some entries in D_i have been changed from binary value 1 to normalized term frequency, the scale analysis attack presented in section 3.4.2 still partially works in the known background model. With similar setting in the previous section, the first query contains two keywords as $\{K_1, K_2\}$ while the second query contains three keywords as $\{K_1, K_2, K_3\}$. Given three documents as an example, the first keyword K_1 appears in two documents as F_1 and F_2, and the second keyword K_2 appears in document F_1. Note that there are some difference between this attack and previous one. If the third keyword K_3 appears in each of these three documents as shown in Tab. 3.1, such equivalence relationship as shown in in Eq. 3.3 does no exist among these documents here. Here we only consider the case that the third keyword K_3 does not appear in any of these three documents. The final similarity scores are shown as follows,

$$\begin{cases} y_1 = r(\frac{1+\ln f_{1,1}}{|F_1|} \cdot \ln(1+\frac{m}{f_1}) + \frac{1+\ln f_{1,2}}{|F_1|} \cdot \ln(1+\frac{m}{f_2}) + \varepsilon_1) + t; \\ y_2 = r(\frac{1+\ln f_{2,1}}{|F_2|} \cdot \ln(1+\frac{m}{f_1}) + \varepsilon_2) + t; \\ y_3 = r\varepsilon_3 + t; \\ y'_1 = r'(\frac{1+\ln f_{1,1}}{|F_1|} \cdot \ln(1+\frac{m}{f_1}) + \frac{1+\ln f_{1,2}}{|F_1|} \cdot \ln(1+\frac{m}{f_2}) + \varepsilon_1) + t'; \\ y'_2 = r'(\frac{1+\ln f_{2,1}}{|F_2|} \cdot \ln(1+\frac{m}{f_1}) + \varepsilon_2) + t'; \\ y'_3 = r'\varepsilon_3 + t'. \end{cases} \tag{3.6}$$

Recall that the scale analysis attack presented in section 3.4.2, it is caused by the fixed value of random variable ε_i in each data vector D_i which remains same here. From Eq. 3.6, the cloud server can still deduce the equivalence relationship as presented in Eq. 3.3. As a result, the document frequency could be exposed to cloud server and further used to identify this keyword in the known background model. To this end, we can employ the same solution as presented in MRSE_II to build the new mechanism as MRSE_II_TF where more dummy keywords instead of only one are inserted into data vectors.

3.5.2 Supporting Data Dynamics

After the dataset is outsourced to the cloud server, it may be updated in addition to being retrieved [109]. Along with the updating operation on data documents, supporting the score dynamics in the searchable index is thus of practical importance. While we consider three dynamic data operations as inserting new documents, modifying existing documents and deleting existing documents, corresponding operations on the searchable index includes generating new index, updating existing index and deleting existing index. Since dynamic data operations also affect the document frequency of corresponding keywords, we also need to update the dictionary $\mathcal{W}$.

For the operation of inserting new documents in the dataset, there may be some new keywords in new documents which need to be inserted in the dictionary $\mathcal{W}$. Remember that every subindex in our scheme has fixed dimension as same as the number of keywords in the old dictionary, so the straightforward solution is to retrieve all the subindexes from the cloud server, and then decrypt, rebuild and encrypt them before outsourcing to the cloud server. However, this approach introduces much cost on computation and communication for both sides which is impractical in the "pay-as-you-use" cloud paradigm. To reduce such great cost, we preserve some blank entries in the dictionary and set corresponding entries in each data vector as 0. If the dictionary needs to index new keywords in the case of inserting new documents, we just replace the blank entries in the dictionary by new keywords, and generate subindexes for new documents based on the updated dictionary. The other documents and their subindexes stored on the cloud server are not affected and therefore remain the same as before. The number of preserved entries functions as a tradeoff parameter to balance the storage cost and the system scalability.

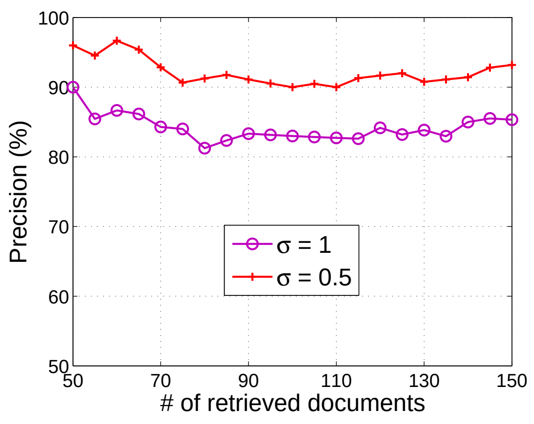
When existing documents are modified, corresponding subindexes are also retrieved from the cloud server and then updated in terms of the term frequency before outsourcing. If new keywords are introduced during the modification operation, we utilize the same method which is proposed in the previous insertion operation. As a special case of modification, the operation of deleting existing documents introduce less computation and communication cost since it only requires to update the document frequency of all the keywords contained by these documents.

3.6 Performance Analysis

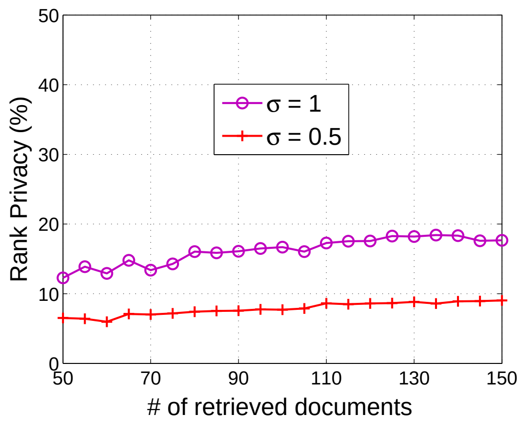
In this section, we demonstrate a thorough experimental evaluation of the proposed technique on a real-world dataset: the Enron Email Dataset [39]. We randomly select different number of emails to build dataset. The whole experiment system is implemented by C language on a Linux Server with Intel Xeon Processor 2.93GHz. The public utility routines by Numerical Recipes are employed to compute the inverse of matrix. The performance of our technique is evaluated regarding the efficiency of four proposed MRSE schemes, as well as the tradeoff between search precision and privacy.

3.6.1 Precision and Privacy

As presented in Section 3.4, dummy keywords are inserted into each data vector and some of them are selected in every query. Therefore, similarity scores of documents will be not exactly accurate. In other words, when the cloud server returns top-k documents based on similarity scores of data vectors to query vector, some of real top-k relevant documents for the query may be excluded. This is because either their original similarity scores are decreased or the similarity scores of some documents out of the real top-k are increased, both of which are due to the impact of dummy keywords inserted into data vectors. To evaluate the purity of the k documents retrieved by user, we define a measure as precision $P_k = k'/k$ where k' is number of real top-k documents that are returned by the cloud server. Fig. 3.3(a) shows that the precision in MRSE scheme is evidently affected by the standard deviation σ of the random variable ε. From the consideration of effectiveness, standard deviation σ is expected to be smaller so as to obtain high precision indicating the good purity of retrieved documents.



(a) Precision



(b) Rank Privacy

Figure 3.3: With different choice of standard deviation σ for the random variable ε, there exists tradeoff between (a) Precision, and (b) Rank Privacy.

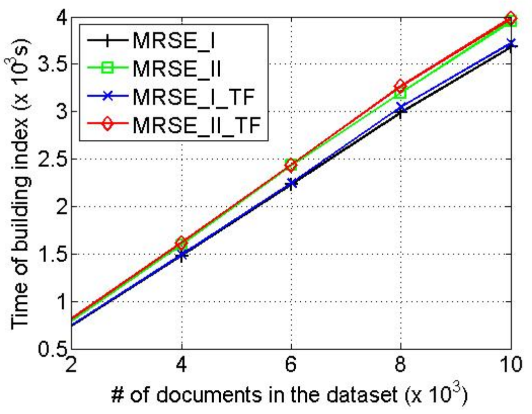
However, user's rank privacy may have been partially leaked to the cloud server as a consequence of small σ. As described in section 3.3.2, the access pattern is defined as the sequence of ranked search results. Although search results cannot be protected (excluding costly PIR technique), we can still hide the rank order of retrieved documents as much as possible. In order to evaluate this privacy guarantee, we first define the rank perturbation as $\widetilde{p_i} = |r_i - r'_i|/k$, where r_i is the rank number of document F_i in the retrieved top-k documents and r'_i is its rank number in the real ranked documents. The overall rank privacy measure at point k is then defined as the average of all the $\widetilde{p}_i$ for every document i in the retrieved top-k documents, denoted as $\widetilde{P}_k = \sum \widetilde{p}_i / k$. Fig. 3.3(b) shows the rank privacy at different points with two standard deviations $\sigma = 1$ and $\sigma = 0.5$ respectively.

From these two figures, we can see that small σ leads to higher precision of search result but lower rank privacy guarantee, while large σ results in higher rank privacy guarantee but lower precision. In other words, our scheme provides a balance parameter for data users to satisfy their different requirements on precision and rank privacy.

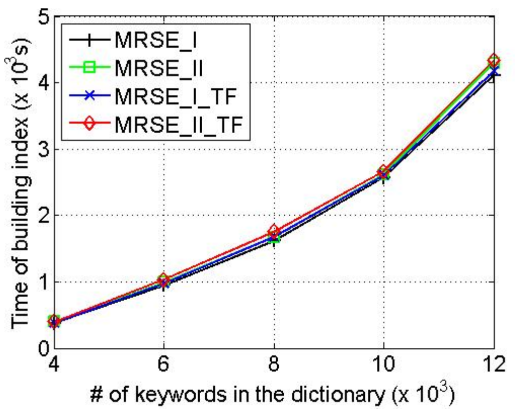
3.6.2 Efficiency

3.6.2.1 Index Construction

To build a searchable subindex I_i for each document F_i in the dataset $\mathcal{F}$, the first step is to map the keyword set extracted from the document F_i to a data vector D_i, followed by encrypting every data vector. The time cost of mapping or encrypting depends directly on the dimensionality of data vector which is determined by the size of the dictionary, i.e., the number of indexed keywords. And the time cost of building the whole index is also related to the number of subindex which is equal to the number of documents in the dataset. Fig. 3.4(a) shows that, given the same dictionary where $|\mathcal{W}| = 4000$, the time cost of building the whole index is nearly linear with the size of dataset since the time cost of building each subindex is fixed. Fig. 3.4(b) shows that the number of keywords indexed in the dictionary determines the time cost of building a subindex. As presented in the section 3.4.1, the major computation to generate a subindex in MRSE_I includes the splitting process and two multiplications of a $(n + 2) \times (n + 2)$ matrix and a $(n + 2)$-dimension vector where $n = |\mathcal{W}|$, both of which have direct relationship with the size of dictionary.



(a) For the different size of dataset with the same dictionary, n = 4000



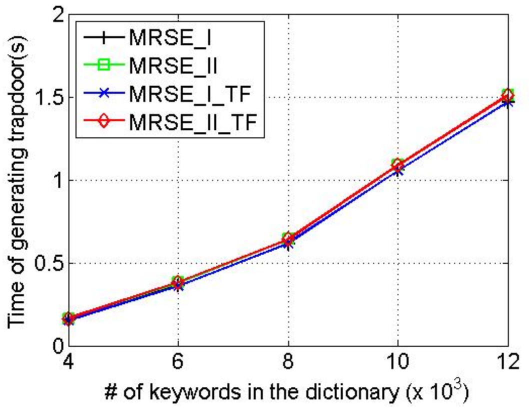
(b) For the same dataset with different size of dictionary, m = 1000

Figure 3.4: Time cost of building index.

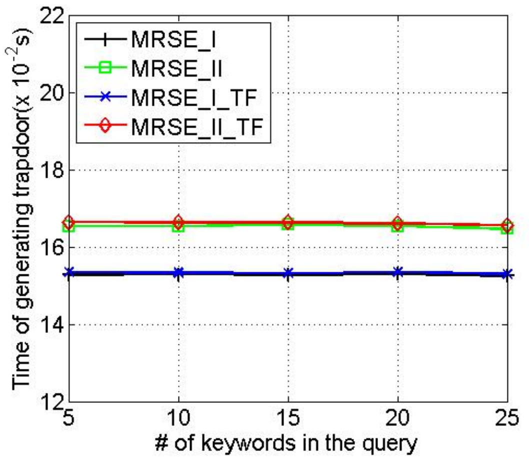
The dimensionality of matrices in MRSE_II is $(n + U + 1) \times (n + U + 1)$ so that its index construction time with complexity $O(m(n + U)^2)$ is bigger than that in MRSE_I with complexity $O(mn^2)$ as shown in both Fig. 3.4(a) and Fig. 3.4(b). As presented in section 3.5.1, both MRSE_I_TF and MRSE_II_TF introduce more computation during the index construction since we need to collect the term frequency information for each keyword in every document and then perform the normalization calculation. But, as shown in both figures, such additional computation in the TF $\times$ IDF weighting rule is insignificant considering much more computation are caused by the splitting process and matrix multiplication. Although the time of building index is not a negligible overhead for the data owner, this is a one-time operation before data outsourcing. Besides, Tab. 3.3 lists the storage overhead of each subindex in two MRSE schemes within different sizes of dictionary. The size of subindex with complexity $O(n)$ is absolutely linear with the dimensionality of data vector which is determined by the number of keywords in the dictionary. The sizes of subindex are very close in the two MRSE schemes because of trivial differences in the dimensionality of data vectors.

3.6.2.2 Trapdoor Generation

Fig. 3.5(a) shows that the time to generate a trapdoor is greatly affected by the number of keywords in the dictionary. Like index construction, every trapdoor generation incurs two multiplications of a matrix and a split query vector, where the dimensionality of matrix or query vector is different in two proposed schemes and becomes larger with the increasing size of dictionary. Fig. 3.5(b) demonstrates the trapdoor generation cost in the MRSE_II scheme with complexity $O((n+U)^2)$ is about 10 percentages larger than that in the MRSE_I scheme with complexity $O(n^2)$. The MRSE_I_TF and MRSE_II_TF have similar difference where the additional logarithm computation accounts for very small proportion of the whole trapdoor generation. Like the subindex generation, the difference of costs to generate trapdoors is majorally caused by the different dimensionality of vector and matrices in the two MRSE schemes. More importantly, it shows that the number of query keywords has little influence on the overhead of trapdoor generation, which is a significant advantage over related works on multi-keyword searchable encryption.



(a) For the same query keywords within different sizes of dictionary, t = 10

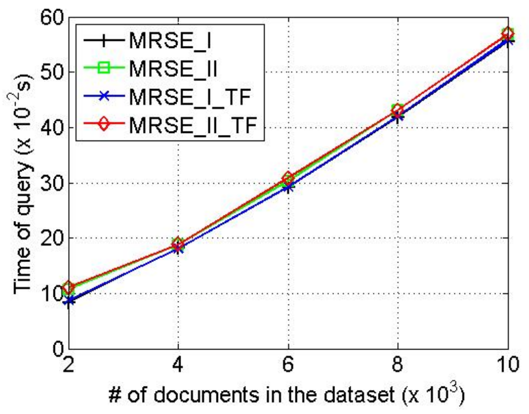


(b) For different numbers of query keywords within the same dictionary, n = 4000

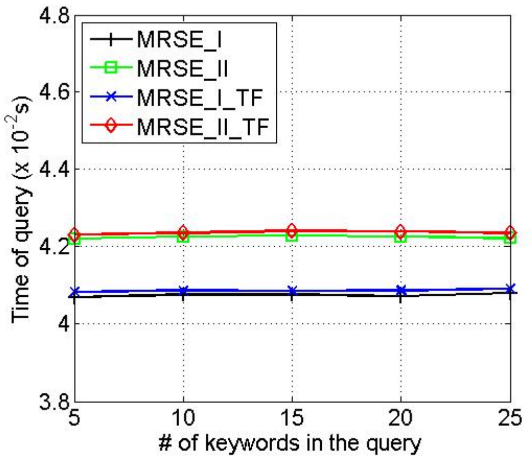
Figure 3.5: Time cost of generating trapdoor.

3.6.2.3 Query

Query execution in the cloud server consists of computing and ranking similarity scores for all documents in the dataset. The computation of similarity scores for the whole data collection is $O(mn)$ in MRSE_I and MRSE_I_TF, and the computation increases to $O(m(n+U))$ in MRSE_II and MRSE_II_TF. Fig. 3.6 shows the query time is dominated by the number of documents in the dataset while the number of keywords in the query has very slight impact on it like the cost of trapdoor generation



(a) For the same query keywords in different sizes of dataset, t = 10



(b) For different numbers of query keywords in the same dataset, m = 1000

Figure 3.6: Time cost of query.

above. The two schemes in the known ciphertext model as MRSE_I and MRSE_I_TF have very similar query speed since they have the same dimensionality which is the major factor deciding the computation cost in the query. The query speed difference between MRSE_I and MRSE_I_TF or between MRSE_II and MRSE_II_TF is also caused by the dimensionality of data vector and query vector. With respect to the communication cost in Query, the size of the trapdoor is the same as that of the subindex listed in the Tab. 3.3, which keeps constant given the same dictionary,

Table 3.3: Size of subindex/trapdoor

Size of dictionary	4000	6000	8000	10000	12000
MRSE_I (KB)	31.3	46.9	62.5	78.1	93.8
MRSE_II (KB)	32.5	48.1	63.8	79.4	95.0

no matter how many keywords are contained in a query. While the computation and communication cost in the query procedure is linear with the number of query keywords in other multiple-keyword search schemes [24, 52], our proposed schemes introduce nearly constant overhead while increasing the number of query keywords.

3.7 Related Work

3.7.1 Single Keyword Searchable Encryption

Traditional single keyword searchable encryption schemes [6,7,15,17,18,22,23,27,35, 40,51,65,98,104,106,115] usually build an encrypted searchable index such that its content is hidden to the server unless it is given appropriate trapdoors generated via secret key(s) [58]. It is first studied by Song et al. [98] in the symmetric key setting, in which each word in the document is encrypted independently under a special two-layered encryption construction. Thus, a searching overhead is linear to the whole file collection length. Goh [51] developed a Bloom Filter based per-file index, reducing the work load for each search request proportional to the number of files in the collection. Chang et al. [35] also developed a similar per-file index scheme. To further enhance search efficiency, Curtmola et al. [40] proposed a per-keyword based approach, where a single encrypted hash table index is built for the entire file collection, with each entry consisting of the trapdoor of a keyword and an encrypted set of related file identifiers. Searchable encryption has also been considered in the public-key setting. Boneh et al. [22] presented the first public-key based searchable encryption scheme, with an analogous scenario to that of [98]. In their construction, anyone with the public key can write to the data stored on the server but only authorized users with the private key can search. Improved definitions are proposed in [7]. Compared to symmetric searchable encryption, public key solutions are usually very computationally expensive. Furthermore, the keyword privacy could not be protected in the public key setting since server could encrypt any keyword with public key and then use the received trapdoor to evaluate this ciphertext. Besides, aiming at tolerance of both minor typos and format inconsistencies in the user search input, fuzzy keyword search over encrypted cloud data has been proposed by Li. et al.in [65] and further extended by Wang. et al. [108]. Note that none of all these schemes support the ranked search problem which we are focusing in this chapter. As an attempt to enrich query experience, our early works [104,106] solve secure ranked keyword search which utilizes keyword frequency to rank results instead of return undifferentiated results. However, it only supports single keyword ranked search.

3.7.2 Boolean Keyword Searchable Encryption

To expand search functionalities, conjunctive keyword search [14,16,24,28,29,52,55, 76,77,80,97] over encrypted data have been proposed. The trapdoor construction in most of these schemes clearly indicates which keyword field will be searched in a query, and therefore exposes information related to keyword privacy and search pattern. These schemes incur large overhead caused by their fundamental primitives, such as computation cost by bilinear map, e.g. [24], or communication cost by secret sharing, e.g. [16]. As a more general search approach, predicate encryption schemes [59,63,93,95] are recently proposed to support both conjunctive and disjunctive keyword search capabilities, and even support inner product. Conjunctive keyword search returns “all-or-nothing”, which means it only returns those documents in which all the keywords specified by the search query appear; disjunctive keyword search returns undifferentiated results, which means it returns every document that contains a subset of the specific keywords, even only one keyword of interest. Note that, inner product queries in predicate encryption only predicates whether two vectors are orthogonal or not, i.e., the inner product value is concealed except when it equals zero. Without providing the capability to compare concealed inner products, predicate encryption is not qualified for performing ranked search. In short, none of existing Boolean keyword searchable encryption schemes support multiple keywords ranked search over encrypted cloud data while preserving privacy as we explore in this chapter. Furthermore, most of these schemes are built upon the expensive evaluation of pairing operations on elliptic curves. Search query is first converted into a polynomial before generating a trapdoor, and computation complexity is polynomial on d^t where t is the number of variables and d is the maximum degree of the resulting polynomial in each variable [59]. Such inefficiency disadvantage also limits their practical performance when deployed in the cloud. Portions of the work studied in this chapter were presented as extended abstract at the 30th IEEE Conference on Computer Communications (INFOCOM’11) [30]. In this chapter we extend and improve more technical details as compared to [30].

3.7.3 Secure Top-K Retrieval from Database Community

In database community, [8, 21, 48, 102, 130] are the most related works to our proposed search schemes. The idea of uniformly distributing posting elements using an order-preserving cryptographic function was first discussed in [102]. However, the order-preserving mapping function proposed in [102] does not support score dynamics, i.e., any insertion and updates of the scores in the index will result in the posting list completely rebuilt. [130] uses a different order-preserving mapping based on pre-sampling and training of the relevance scores to be outsourced, which is not as efficient as our proposed schemes. When scores following different distributions need to be inserted, their score transformation function still needs to be rebuilt. On the contrary, in our scheme the score dynamics can be gracefully handled. We note that supporting score dynamics, which can save quite a lot of computation overhead when file collection changes, is a significant advantage in our scheme. Most important, all the works based on order-preserving mapping techniques do not well support multi-keyword search. Since the order-preserving mapping is only designed for every single keyword, the simple sum of encrypted scores does not preserve the order of the sum of original scores. [48] does not take into consideration the term frequency of keywords during the query which causes the low search quality.

3.7.4 Other Related Techniques

Allowing range queries over encrypted data has been studied in both public key setting [24, 94], where advanced privacy preserving schemes were proposed to allow more sophisticated multi-attribute search over encrypted files. Though these two schemes provide provably strong security, they do not support the ordered result listing on the server side. Thus, they can not be effectively utilized in our settings since the user still does not know which retrieved files would be the most relevant. Related research on range queries in symmetric key setting [9, 10] do not provide provable security guarantee.

3.8 Conclusion

In this chapter, for the first time we define and solve the problem of multi-keyword ranked search over encrypted cloud data, and establish a variety of privacy requirements. Among various multi-keyword semantics, we choose the efficient similarity measure of “coordinate matching”, i.e., as many matches as possible, to effectively capture the relevance of outsourced documents to the query keywords, and use “inner product similarity” to quantitatively evaluate such similarity measure. For meeting the challenge of supporting multi-keyword semantic without privacy breaches, we propose a basic idea of MRSE using secure inner product computation. Then we give two improved MRSE schemes to achieve various stringent privacy requirements in two different threat models. We also investigate some further enhancements of our ranked search mechanism, including supporting more search semantics, i.e., TF $\times$ IDF, and dynamic data operations. Thorough analysis investigating privacy and efficiency guarantees of proposed schemes is given, and experiments on the real-world dataset show our proposed schemes introduce low overhead on both computation and communication.

Chapter 4

Privacy-Preserving Query over Encrypted Graph-Structured Data

4.1 Introduction

In the increasingly prevalent cloud computing, datacenters play a fundamental role as the major cloud infrastructure providers [11], such as Amazon, Google, and Microsoft. Datacenters provide the utility computing service to software providers who further provide the application service to end users through Internet. The later service has long been called "Software as a Service (SaaS)", and the former service has recently been called "Infrastructure as a Service (IaaS)", where the software service provider is also referred to as cloud service provider. To take advantage of computing and storage resources provided by cloud infrastructure providers, data owners outsource more and more data to the datacenters [58] through cloud service providers, e.g., the online storage service provider, which are not fully trusted by data owners. As a general data structure to describe the relation between entities, the graph has been increasingly used to model complicated structures and schemaless data, such as the personal social network (the social graph), the relational database, XML documents and chemical compounds studied by research labs [38, 91, 92, 128, 132, 133]. Images in the personal album can also be modeled as the attributed relational graph (ARG) [20]. For the protection of users' privacy, these sensitive data have to be encrypted before outsourcing to the cloud. Moreover, some data are supposed to be shared among trusted partners. For example, the album owner may share family party photos with only authorized users including family members and friends. For another example, the lab director and members are given the authorization to access the entire lab data. In both cases, authorized users are usually planning to retrieve some portion of data they are interested rather than the entire dataset, mostly because of the "pay-for-use" billing rule in the cloud computing paradigm. Considering the large amount of data centralized in the datacenter, it is a very challenging task to effectively utilize the graph-structured data after encryption.

With the conventional graph data utilization method, we first take the query graph as an input, and then perform the graph containment query: *given a query graph as Q and a collection of data graphs as $\mathcal{G} = (G_1, G_2, \ldots, G_m)$, find all the supergraphs of Q in $\mathcal{G}$, denoted as $\mathcal{G}_Q$.* The straightforward solution is to check whether Q is subgraph isomorphic to every G_i in $\mathcal{G}$ or not. However, checking subgraph isomorphism is NP-complete, and therefore it is infeasible to employ such costly solution. To efficiently solve the graph containment query problem, there have been a lot of proposed techniques [38, 91, 92, 128, 132, 133], most of which follow the principle of "filtering-and-verification". In the filtering phase, a pre-built feature-based index is utilized to prune as many data graphs from the dataset as possible and output the candidate supergraph set. Every feature in the index is a fragment of a data graph, e.g., the subgraph. In the verification phase, each candidate supergraph is verified by checking subgraph isomorphism. Since the candidate supergraph set is much smaller than the entire dataset, such approach involves less subgraph isomorphism checking, and therefore is significantly more efficient than the straightforward solution. However, when data graphs are stored in the encrypted form in the cloud, the encryption excludes the filtering method which is based on the plaintext index. Recently, Chase and Kamara proposed structured encryption [36] to handle private access to parts of a large graph in encrypted form; yet only simple operations such as neighbor queries are supported.

In the most related literature, the searchable encryption [22, 30, 40, 64, 65, 106] is a helpful technique that treats encrypted data as documents and allows a user to securely search over it through specifying single keyword or multiple keywords with Boolean relations. However, the direct application of these approaches to deploy the secure large scale cloud data utilization system would not be necessarily suitable. The keyword-based search provides much less semantics than the graph-based query since the graph could characterize more complicated relations than Boolean relation. More importantly, these searchable encryption schemes are developed as crypto primitives and cannot accommodate such high service-level requirements like system usability, user query experience, and easy information discovery in mind. Therefore, how to design an efficient encrypted query mechanism which supports graph semantics without privacy breaches still remains a challenging open problem.

In this chapter, for the first time, we define and solve the problem of privacy-preserving graph query in cloud computing (PPGQ). To reduce the times of checking subgraph isomorphism, we adopt the efficient principle of "filtering-and-verification" to prune as many negative data graphs as possible before verification. A feature-based index is firstly built to provide feature-related information about every encrypted data graph. Then, we choose the efficient inner product as the pruning tool to carry out the filtering procedure. To achieve this functionality in index construction, each data graph is associated with a binary vector as a subindex where each bit represents whether the corresponding feature is subgraph isomorphic to this data graph or not. The query graph is also described as a binary vector where each bit means whether the corresponding feature is contained in this query graph or not. The inner product of the query vector and the data vector could exactly measure the number of query features contained in the data graph, which is used to filter negative data graphs that do not contain the query graph. However, directly outsourcing the data vector or the query vector will violate the index privacy or the query privacy. To meet the challenge of supporting graph semantics without privacy breaches, we propose a secure inner product computation mechanism, which is adapted from a secure k-nearest neighbor (kNN) technique [120], and then show our improvements on it to achieve various privacy requirements under the known-background threat model. Our contributions are summarized as follows,

1) For the first time, we explore the problem of query over encrypted graph-structured data in cloud computing, and establish a set of strict privacy requirements for such a secure cloud data utilization system to become a reality.

2) Our proposed scheme follows the principle of "filtering-and-verification" for efficiency consideration, and thorough analysis investigating privacy and efficiency guarantees of the proposed scheme is given.

3) The evaluation, which is performed with the widely-used AIDS antiviral screen dataset on the Amazon EC2 cloud infrastructure, further shows our proposed scheme introduces low computation and communication overhead.

The remainder of this chapter is organized as follows. In Section 4.2, we introduce the system model, the threat model and our design goals. Section 4.3 gives preliminaries, and section 4.4 describes the framework and privacy requirements in PPGQ, followed by section 4.5, which gives our proposed scheme. Section 4.6

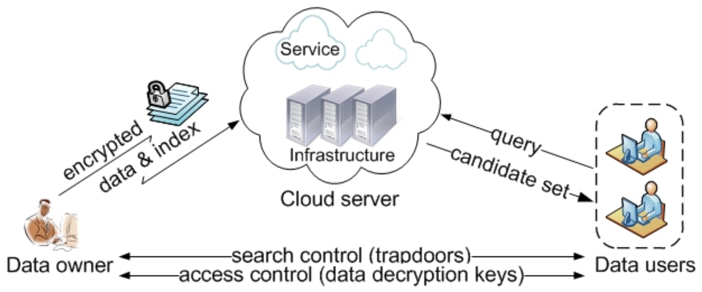


Figure 4.1: Architecture of graph query over encrypted cloud data

presents evaluation results. We discuss related work on both keyword searchable encryption and graph containment query in Section 4.7, and conclude the chapter in Section 4.8.

4.2 Problem Formulation

4.2.1 The System Model

Considering a cloud data storage service, involving four different entities: the data owner, the data user, the storage service provider/cloud service provider, and the datacenter/cloud infrastructure provider. To take advantage of the utility computing services provided by the datacenter, e.g., computing and storage resources, the storage service provider deploys its storage service on top of the utility computing in datacenter and delivers the service to end users (including data owners and data users) through Internet. In our system model, neither cloud service provider nor cloud infrastructure provider is fully trusted by data owners or data users, so they are treated as an integrated entity, named the cloud server, as shown in Fig. 4.1.

The data owner has a graph-structured dataset $\mathcal{G}$ to be outsourced to the cloud server in the encrypted form $\widetilde{\mathcal{G}}$. To enable the query capability over $\widetilde{\mathcal{G}}$ for effective data utilization, the data owner will build an encrypted searchable index $\mathcal{I}$ from $\mathcal{G}$

before data outsourcing, and then both the index $\mathcal{I}$ and the encrypted graph dataset $\widetilde{\mathcal{G}}$ are outsourced to the cloud server. For every query graph Q, an authorized user acquires a corresponding trapdoor T_Q through the search control mechanism, e.g., broadcast encryption [40], and then sends it to the cloud server. Upon receiving T_Q from data users, the cloud server is responsible to perform query over the encrypted index $\mathcal{I}$ and return the encrypted candidate supergraphs. Finally, data users decrypt the candidate supergraphs through the access control mechanism, and verify each candidate by checking subgraph isomorphism.

4.2.2 The Known Background Threat Model

The cloud server is considered as "honest-but-curious" in our model, which is consistent with most related works on searchable encryption [30, 129]. Specifically, the cloud server acts in an "honest" fashion and correctly follows the designated protocol specification. However, it is "curious" to infer and analyze the data and the index in its storage and interactions during the protocol so as to learn additional information. The encrypted data $\widetilde{\mathcal{G}}$ and searchable index $\mathcal{I}$ can be easily obtained by the cloud server, because both of them are outsourced and stored on the cloud server. In addition to these encrypted information, the cloud server is supposed to know some backgrounds on the dataset, such as its subject and related statistical information. As a possible attack similar to that in [130], the cloud server could utilize the feature frequency to identify features contained in the query graph.

4.2.3 Design Goals

To enable the graph query for the effective utilization of outsourced cloud data under the aforementioned model, our design should simultaneously achieve security and performance guarantees.

- **Effectiveness:** To design a graph query scheme that introduces few false positives in the candidate supergraph set.

- **Privacy:** To prevent the cloud server from learning additional information over outsourced data and index in query interactions, and to meet privacy requirements specified in section 4.4.3.

- **Efficiency:** Above goals on effectiveness and privacy should be achieved with low communication and computation overhead.

4.2.4 Notations

- $\mathcal{G}$ – the graph-structured dataset, denoted as a collection of m data graphs $\mathcal{G} = (G_1, G_2, \ldots, G_m)$.

- $\widetilde{\mathcal{G}}$ – the encrypted graph-structured dataset outsourced into the cloud, denoted as $\widetilde{\mathcal{G}} = (\widetilde{G}_1, \widetilde{G}_2, \ldots, \widetilde{G}_m)$.

- $\mathrm{id}(G_i)$ – the identifier of the data graph G_i that can help uniquely locate the graph.

- $\mathcal{F}$ – the feature set mined from the graph dataset, denoted as $\mathcal{F} = (F_1, F_2, \ldots, F_n)$.

- $\mathcal{D}$ – the frequent feature dictionary, denoted as $\mathcal{D} = \{\mathcal{L}_{F_1}, \mathcal{L}_{F_2}, \ldots, \mathcal{L}_{F_n}\}$, where $\mathcal{L}_{F_j}$ is the unique canonical label of F_j;

- $\mathcal{I}$ – the searchable index associated with $\widetilde{\mathcal{G}}$, denoted as $(I_1, I_2, \ldots, I_m)$, where each subindex I_i is built from G_i.

- Q – the query graph from the data user.

- $\mathcal{F}_Q$ – the subset of $\mathcal{F}$, consisting of frequent features contained in Q, denoted as $\mathcal{F}_Q = \{F_j | F_j \subseteq Q, F_j \in \mathcal{F}\}$.

- $\mathcal{G}_{\{Q\}}$ – the subset of $\mathcal{G}$, consisting of exact supergraphs of the graph Q, denoted as $\mathcal{G}_{\{Q\}} = \{\text{id}(G_i)|Q \subseteq G_i, G_i \in \mathcal{G}\}$.

- $\mathcal{G}_{\mathcal{F}_Q}$ – the subset of $\mathcal{G}$, consisting of candidate supergraphs of the graph Q, denoted as $\mathcal{G}_{\mathcal{F}_Q} = \cap \mathcal{G}_{\{F_j\}}$, where $F_j \in \mathcal{F}_Q$.

- T_Q – the trapdoor for the query graph Q.

4.3 Preliminaries

4.3.1 Graph Query

A labeled, undirected, and connected graph is a five-tuple as $\{V, E, \Sigma_V, \Sigma_E, L\}$, where V is the vertex set, $E \subseteq V \times V$ is the edge set, and L is a labeling function: $V \to \Sigma_V$ and $E \to \Sigma_E$. We use the number of vertices $|V(G)|$ to represent the size of the graph G.

Subgraph Isomorphism Given two graphs $G = \{V, E, \Sigma_V, \Sigma_E, L\}$ and $G' = \{V', E', \Sigma_V, \Sigma_E, L'\}$, G is subgraph isomorphic to G' if there is an injection $f : V \to V'$ such that

1. $\forall\ v \in V, L(v) = L'(f(v))$.
2. $\forall\ (u, v) \in E, (f(u), f(v)) \in E'$.
3. $\forall\ (u, v) \in E, L(u, v) = L'(f(u), f(v))$.

Graph Containment Query If G is subgraph isomorphic to G', we call G is a *subgraph* of G' or G' is a *supergraph* of G, denoted as $G \subseteq G'$. Such relation is also referred to as G is *contained by* G' or G' *contains* G. Given a graph dataset $\mathcal{G} = (G_1, G_2, \ldots, G_m)$ and a query graph Q, a graph containment query problem is to find all the supergraphs of Q from the dataset $\mathcal{G}$, denoted as $\mathcal{G}_{\{Q\}} = \{\text{id}(G_i)|Q \subseteq G_i, G_i \in \mathcal{G}\}$ where $\text{id}(G_i)$ is the identifier of the graph G_i. The number of supergraphs of Q, i.e., $|\mathcal{G}_{\{Q\}}|$, is called the support, or the frequency of Q.

Considering the large size of the graph dataset, it is impractical to solve the graph containment query problem by sequentially checking whether Q is subgraph isomorphic to each graph in $\mathcal{G}$ or not, because checking subgraph isomorphism has been proved to be NP-complete [50]. To reduce the times of checking subgraph isomorphism, most graph query works [38, 91, 92, 128, 132, 133] follow the principle of "filtering-and-verification".

Filtering-and-Verification In the filtering phase, a feature-based index for the dataset $\mathcal{G}$ is utilized to prune most negative data graphs that does not contain the query graph Q, and then produce the candidate supergraph set. In the verification phase, the subgraph isomorphism is checked between the query graph and every candidate supergraph to output the exact supergraph set $\mathcal{G}_{\{Q\}}$.

The feature-based index is pre-built from the entire graph dataset, where each feature F_j is a substructure of a data graph in the dataset, such as subpath [92], subtree [91, 132, 133] and subsubgraph [38, 128]. Let $\mathcal{F} = (F_1, F_2, \ldots, F_n)$ represent the feature set. The supergraph set of every feature F_j, denoted as $\mathcal{G}_{\{F_j\}}$, is stored in the index. Let $\mathcal{F}_Q = \{F_j | F_j \subseteq Q, F_j \in \mathcal{F}\}$ denote the query feature set consisting of features contained in the query graph. Then, the candidate supergraphs of the query graph G can be obtained by the intersection operation as $\mathcal{G}_{\mathcal{F}_Q} = \cap \mathcal{G}_{\{F_k\}}$, where $F_k \in \mathcal{F}_Q$. The false positive ratio is then defined as $\frac{|\mathcal{G}_{\mathcal{F}_Q}|}{|\mathcal{G}_{\{Q\}}|}$

Frequent and Discriminative Substructure It is infeasible and unnecessary to index every possible substructure of all the graphs in a large dataset, and therefore only frequent and discriminative substructures are indexed to reduce the index size. A feature F_j is frequent if its support, or frequency is large enough, i.e., $|\mathcal{G}_{\{F_j\}}| \geq \sigma$, where σ is called the minimum support. A feature F_j is discriminative if it can provide more pruning power than its subgraph feature set, i.e., $\frac{|\cap_k \mathcal{G}_{\{F_k\}}|}{|\mathcal{G}_{\{F_j\}}|} \geq \gamma$, where

$F_k \subseteq F_j$ and γ is called the discriminative threshold.

4.3.2 Secure Euclidean Distance Computation

In order to compute the inner product in a privacy-preserving method, we will adapt the secure Euclidean distance computation in the secure k-nearest neighbor (kNN) scheme [120]. In this scheme, the Euclidean distance between a database record p_i and a query vector q is used to select k nearest database records. The secret key is composed of one $(d+1)$-bit vector as S and two $(d+1) \times (d+1)$ invertible matrices as $\{M_1, M_2\}$, where d is the number of fields for each record p_i. First, every data vector p_i and the query vector q are extended to $(d+1)$-dimensional vectors as $\vec{p_i}$ and $\vec{q}$, where the $(d+1)$-th dimension is set to $-0.5||p_i^2||$ and 1, respectively. Besides, the query vector $\vec{q}$ is scaled by a random number $r > 0$ as (rq, r). Then, $\vec{p_i}$ is split into two random vectors as $\{\vec{p_i}', \vec{p_i}''\}$, and $\vec{q}$ is also split into two random vectors as $\{\vec{q}\,', \vec{q}\,''\}$. Note here that vector S functions as a splitting indicator. Namely, if the j-th bit of S is 0, $\vec{p_i}'[j]$ and $\vec{p_i}''[j]$ are set as the same as $\vec{p_i}[j]$, while $\vec{q}\,'[j]$ and $\vec{q}\,''[j]$ are set to two random numbers so that their sum is equal to $\vec{q}[j]$; if the j-th bit of S is 1, the splitting process is similar except that $\vec{p_i}$ and $\vec{q}$ are switched. The split data vector pair $\{\vec{p_i}', \vec{p_i}''\}$ is encrypted as $\{M_1^T\vec{p_i}', M_2^T\vec{p_i}''\}$, and the split query vector pair $\{\vec{q}\,', \vec{q}\,''\}$ is encrypted as $\{M_1^{-1}\vec{q}\,', M_2^{-1}\vec{q}\,''\}$. In the query step, the product of the data vector pair and the query vector pair, i.e., $-0.5r(||p_i||^2 - 2p_i \cdot q)$, is serving as the indicator of the Euclidean distance $(||p_i||^2 - 2p_i \cdot q + ||q||^2)$ to select k nearest neighbors. Without prior knowledge of the secret key, neither the data vector nor the query vector, after such a series of processes, can be recovered by analyzing their corresponding ciphertexts. The security analysis in [120] shows that this computation technique is secure against known-plaintext attack, which is roughly equal in security to a d-bit symmetric key. Therefore, d should be no less than 80 to make the search space sufficiently large.

4.4 PPGQ: The Framework and Privacy

In this section, we define the framework of query over encrypted graph-structured data in cloud computing and establish various strict system-wise privacy requirements for such a secure cloud data utilization system.

4.4.1 The Framework

Our proposed framework focuses on how the query works with the help of index which is outsourced to the cloud server. We do not illustrate how the data itself is encrypted, outsourced or accessed, as this is a complementary and orthogonal issue and has been studied elsewhere [129]. The framework of PPGQ is illustrated as follows.

- $\mathsf{FSCon}(\mathcal{G}, \sigma)$ *Takes the graph dataset* $\mathcal{G}$ *and the minimum support* σ *as inputs, outputs a frequent feature set* $\mathcal{F}$*.*

- $\mathsf{KeyGen}(\xi)$ *Takes a secret* ξ *as input and outputs a symmetric key* $\mathcal{K}$*.*

- $\mathsf{BuildIndex}(\mathcal{G}, \mathcal{K})$ *Takes the graph dataset* $\mathcal{G}$ *and the symmetric key* $\mathcal{K}$ *as inputs, output a searchable index* $\mathcal{I}$*.*

- $\mathsf{TDGen}(Q, \mathcal{K})$ *Takes the query graph* Q *and the symmetric key* $\mathcal{K}$ *as inputs, outputs a corresponding trapdoor* T_Q*.*

- $\mathsf{Query}(T_Q, \mathcal{I})$ *Takes the trapdoor* T_Q *and the searchable index* $\mathcal{I}$ *as inputs, returns* $\mathcal{G}_{\mathcal{F}_Q}$*, i.e., the candidate supergraphs of query graph* Q*.*

The first three algorithms, i.e., FSCon, BuildIndex, and BuildIndex, are run by the data owner as pre-processes. The query algorithm is run on the cloud server as a part of the cloud data storage service. According to various search control mechanisms, the trapdoor generation algorithm TDGen may be run by either the data owner or the data user. Besides, depending on some specific application scenarios, while search requests on confidential documents may be allowed for all users, the access to document contents may be forbidden for those low-priority data users. Note that neither search control or access control are within the scope of this dissertation.

4.4.2 Choosing Frequent Features

To build a feature-based index, there are three choices of features, i.e., subpath, subtree and subgraph, which can be extracted from the graph dataset. According to the feature comparison in [133], with the same minimum support, either subtree-based or subgraph-based feature set is larger than subpath-based one, especially when the feature size is between 5 and 20. To be consistent with the size of graph which is $|V(G)|$, the size of feature is measured by its number of vertices $|V(F_i)|$. As for the cloud server, the larger feature set will demand more index storage, and also incur larger computation cost during the query process. However, the pruning power of the subgraph-based index performs the best among all the three choices, which leads to the lowest false positive ratio and the smallest candidate supergraph set.From the perspective of the data user, the size of the candidate supergraph set has a direct and important impact on the communication and computation cost. Compared with the powerful cloud server, data users may access the cloud server through portable devices, e.g., mobile phones and netbooks, which have limited capability of communication and computation to retrieve the candidate supergraph set and check subgraph isomorphism. To this end, the subgraph-based index is more appropriate than the other two choices for our PPGQ framework that is designed for the efficient graph-structured data utilization in cloud computing. To generate the frequent feature set, there have been a lot of frequent subgraph mining algorithms over the large graph dataset, such as *gSpan* [127], and *Gaston* [72]. For the indexing purpose, every frequent subgraph should be represented as a unique canonical label which can be accomplished by existing graph sequentialization techniques, like CAM [54] and DFS [127]. Besides, the shrinking process on the frequent feature set is not adopted in our framework since it will weaken the pruning power of index. As the subgraph is chosen as the feature to build index in our framework, we do not distinguish between frequent feature and frequent subgraph in the rest of this chapter.

4.4.3 Privacy Requirements

As described in the framework, *data privacy* is to prevent the cloud server from prying into outsourced data, and can be well protected by existing access control mechanism [129]. In related works on privacy-preserving query, like searchable encryption [40], representative privacy requirement is that the server should learn nothing but query results. With this general privacy statement, we explore and establish a set of stringent privacy requirements specifically for the PPGQ framework. While data privacy guarantees are demanded by default in the related literature, various *query privacy* requirements involved in the query procedure are more complex and difficult to tackle as follows.

4.4.3.1 Index Privacy

With respect to the *index privacy*, if the cloud server deduces any association between frequent features and encrypted dataset from outsourced index, it may learn

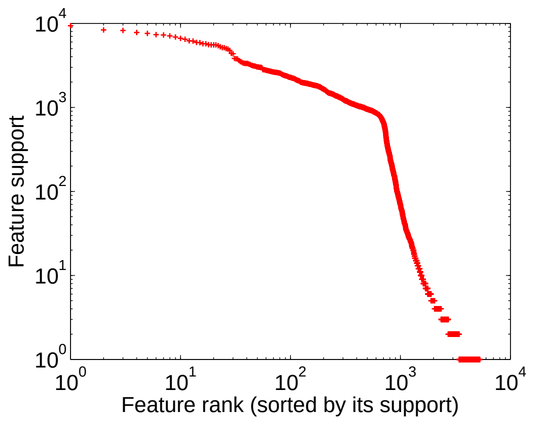


Figure 4.2: Distribution of Feature Support

the major structure of a graph, or even the entire topology of a small graph. Therefore, searchable index should be constructed in such a way that prevents the cloud server from performing such kind of association attack.

4.4.3.2 Feature Privacy

Data users usually prefer to keep their query from being exposed to others like the cloud server, and the most important concern is to hide what they are querying, i.e., the features indicated by the corresponding trapdoor. Although trapdoor can be generated in a cryptographic way to protect the query features, the cloud server may do some statistical analysis over the search results to make an estimate. Especially, the *feature support* (i.e., the number of data graphs containing the feature), a kind of statistical information, is sufficient to identify the feature with high probability. When the cloud server knows some background information of the dataset, this feature-specific information can be utilized to reverse-engineer the feature. As presented in Fig. 4.2, the distribution of feature support in the AIDS antiviral screen dataset [5] provides enough information to identify most frequent features in the dataset. Such problem is similar with the keyword privacy issue in [131], where document frequency (the number of documents containing the keyword) is used as a statistical information to reverse-engineer the keyword.

4.4.3.3 Trapdoor Unlinkability

The trapdoor generation function should be a randomized one instead of being deterministic. In particular, the cloud server should not be able to deduce the relationship of any given trapdoors, e.g., to determine whether the two trapdoors are formed by the same search request or not. Otherwise, the deterministic trapdoor generation would give the cloud server advantage to accumulate frequencies of different search requests regarding different features, which may further violate the aforementioned feature privacy requirement. So the fundamental protection for trapdoor unlinkability is to introduce sufficient nondeterminacy into the trapdoor generation procedure.

4.4.3.4 Access Pattern

Access pattern is the sequence of query results where each query result is $\mathcal{G}_{\mathcal{F}_Q}$, including the id list of candidate supergraphs of the query graph. Then the access pattern is denoted as $(\mathcal{G}_{\mathcal{F}_{Q_1}}, \mathcal{G}_{\mathcal{F}_{Q_1}}, \ldots)$ which are the results of sequential queries. In related literature, although a few schemes (e.g., [23, 29]) have been proposed to utilize private information retrieval (PIR) technique [56] to hide access pattern, our proposed schemes are not designed to protect access pattern for the efficiency concerns. This is because any PIR-based technique must "touch" the whole dataset outsourced on the server which is inefficient in the large scale cloud system. To this end, the query result of any single feature F_j, which is part of access pattern, cannot be hidden from the cloud server. Such query result $\mathcal{G}_{\{F_j\}}$ will directly expose the support of the feature, and break the feature privacy as discussed above. Therefore, we do not consider the single-feature query in our proposed schemes.

4.5 PPGQ: The Proposed Scheme and Analysis

In order to accomplish the filtering purpose in the graph query procedure, the data graph G_i is selected as a candidate supergraph of the query graph Q if and only if G_i contains all the frequent features in Q. Let λ_i represent the number of query features contained in the data graph G_i. For every candidate supergraph G_i, its corresponding λ_i should be equal to the size of the query feature set $\mathcal{F}_Q$, i.e., $\lambda_i = |\mathcal{F}_Q|$. To obtain the candidate supergraph set, we propose to employ the efficient inner product computation for pruning negative data graphs G_j that do not contain the query graph, i.e., $\lambda_j < \mathcal{F}_Q$. Specifically, every data graph G_i is formalized as a bit vector g_i where each bit $g_{i[j]}$ is determined by checking whether G_i contains the frequent feature F_j or not. If $F_j \subseteq G_i$, $g_{i[j]}$ is set as 1; otherwise, it is set as 0. The query graph Q is formalized as a bit vector q where each bit $q_{[j]}$ also represents the existence of the frequent feature F_j in the query feature set $\mathcal{F}_Q$. Then, λ_i can be acquired via computing the inner product of the data vector g_i and the query vector q, i.e., $g_i \cdot q$. To preserve the strict system-wise privacy, the data vector g_i and the query vector q should not be exposed to the cloud server. In this section, we first design a secure inner product computation mechanism, which is adapted from the secure Euclidean distance computation technique, and then show how to improve it to be privacy-preserving under the known-background threat model.

Table 4.1: Analysis on inner products in two correlated queries

G	$\mathcal{F}_Q$	$\mathcal{F}_{Q'} = \mathcal{F}_Q \bigcup \{F_k\}$	y_i'/y_i	$\lambda' - \lambda$
G_i	$y_i = r\lambda_i$	$\lambda_i' = \lambda_i + 1,\ y_i' = r'\lambda_i'$	$\frac{\lambda_i+1}{\lambda_i} \cdot \frac{r'}{r}$	1
G_j	$y_j = r\lambda_j$	$\lambda_j' = \lambda_j,\ y_j' = r'\lambda_j'$	$\frac{r'}{r}$	0

4.5.1 Privacy Concerns on Secure Inner Product Computation

Since the inner product of the data vector and the query vector is preferred to select candidate supergraphs of the query graph, the secure Euclidean distance computation technique in the secure kNN scheme [120] cannot be directly utilized here. As shown in Section 3.4.1, by eliminating the extended dimension which is related to the Euclidean distance, the final inner product result changes to be $r(g_i \cdot q)$. Since the new result $r(g_i \cdot q)$ can serve as an indicator of the original inner product $g_i \cdot q$, it seems that an efficient and secure inner product computation scheme can be appropriately achieved. However, the cloud server may break the feature privacy via analyzing final inner products and figuring out some feature-specific statistical information, e.g., the support of feature. With the background knowledge of the outsourced graph dataset, which can be obtained by the cloud server under the known-background model, such feature-specific information could be further utilized to identify what feature is included in the query at high probability. We first demonstrate how such statistical analysis attack could break feature privacy as follows.

Whenever there exist two query graphs which have inclusion relationship, the cloud server could explore the relationship among final inner products in two queries. Assume that T_Q and $T_{Q'}$ be trapdoors for two query graphs Q and Q', and their corresponding query feature sets have the inclusion relation as $\mathcal{F}_Q \subset \mathcal{F}_{Q'}$. Especially, when the differential feature subset contains only one feature, i.e., $|\mathcal{F}_{Q''}| = 1$ where

$\mathcal{F}_{Q''} = \mathcal{F}_{Q'} \backslash \mathcal{F}_Q$, the cloud server can deduce an estimate of the support of the differential feature and further identify this feature with the background knowledge of the graph dataset. As listed in Tab. 4.1, the second query feature set $\mathcal{F}_{Q'}$ includes one more feature as F_k than the first one $\mathcal{F}_Q$. The cloud server evaluates the expression ${y_i}'/y_i$, which is equal to $({\lambda_i}'/\lambda_i)(r'/r)$ for every graph G_i, and then obtains a large number of different values. However, these values could be distinguished into two categories. If the graph G_i does not contain the feature F_k, i.e., ${\lambda_i}' = \lambda_i$, its corresponding expression evaluation ${y_i}'/y_i$ is equal to r'/r; otherwise, it is larger than r'/r and can be easily detected because of its special ratio as $\frac{\lambda_i+1}{\lambda_i}$. Therefore, the minimum values over the whole dataset indicate that corresponding data graphs do not contain the feature F_k, and other graphs with larger values contain it. In addition, by checking whether the expression y_i is equal to 0 or not, the special case where the data graph G_i contains neither feature in $\mathcal{F}_Q$ can be recognized by the cloud server. In such case, the existence of feature F_k in G_i can be determined by checking whether the expression ${y_i}'$ is equal to 0 or not. To this end, the total number of data graphs containing this feature, i.e., $|\mathcal{G}_{\{F_k\}}|$, is uncovered. Under the known-background threat model, the cloud server could break the feature privacy with both the support of single feature and the distribution of all the supports as illustrated in Fig. 4.2.

4.5.2 The Proposed Privacy-Preserving Graph Query Scheme

The statistical analysis attack shown above works when the final inner product y_i is a multiple of λ_i, i.e., the number of query features contained in the data graph G_i. To this end, we should break such scale relationship to make the previous statistical analysis attack infeasible. Our proposed design is to convert both the data vector and the query vector from the bit structure to more sophisticated structures. Specif-

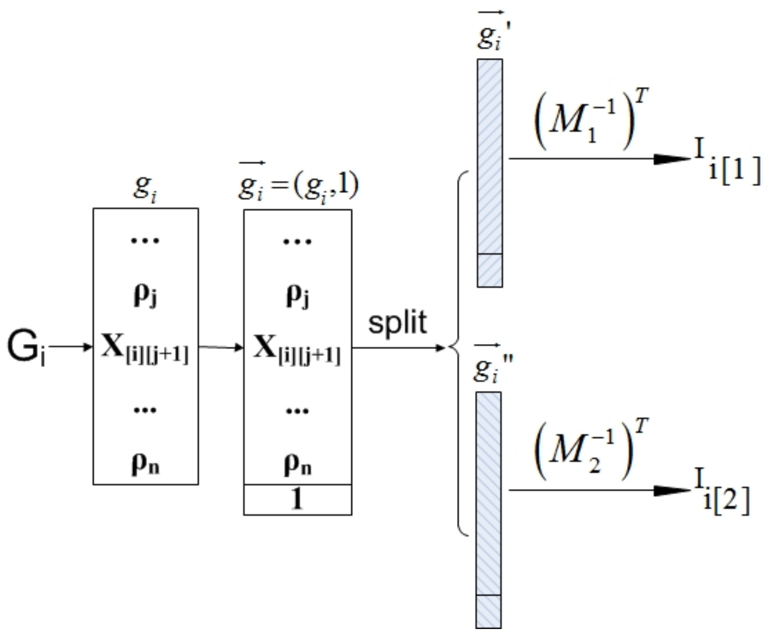


Figure 4.3: Build subindex for each data graph

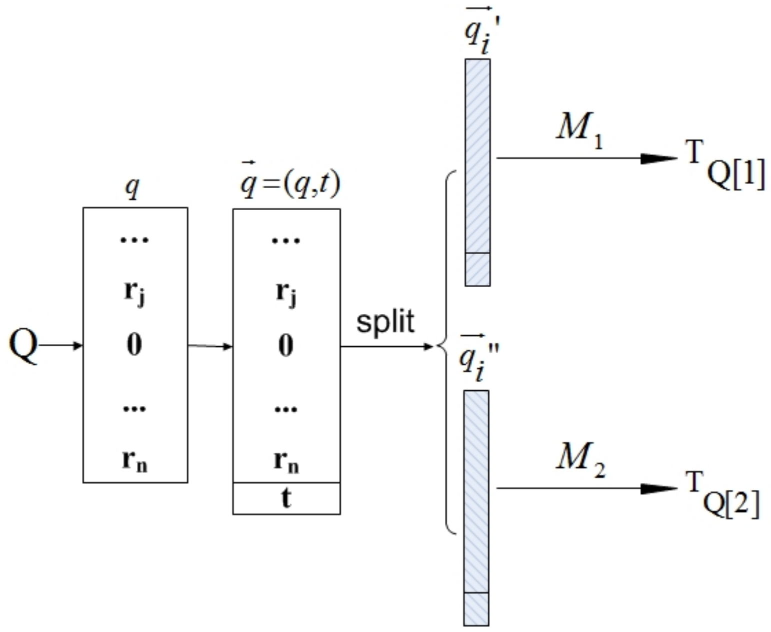


Figure 4.4: Generate trapdoor for query graph

ically, if the frequent feature F_j is contained in the data graph G_i, the corresponding element $g_{i[j]}$ in the data vector g_i is set as $\rho_{[j]}$ instead of 1 where ρ is a n-dimensional vector; otherwise, $g_{i[j]}$ is set as $X_{[i][j]}$ where X is a $n \times n$ matrix and $X_{[i][j]}$ is a random number less than $\rho_{[j]}$. Correspondingly, if F_j is contained in the query graph Q, $q_{[j]}$ is set as a positive random number r_j instead of 1; otherwise, $q_{[j]}$ is set as 0. In addition, to hide the original inner product, we resume the dimension extending operation where the $g_{i[n+1]}$ is set as 1 and the $q_{[n+1]}$ is set as a random number t. As a result of these modifications, the final inner product of the data vector and the query vector, i.e., $g_i \cdot q + t$, should be equal to $\rho \cdot q + t$ for any candidate supergraph. Note that, the vector ρ is constant as a part of the secret key, but t and r_j in q are randomly generated for each query. Our proposed privacy-preserving graph query scheme is designed as follows with details in Fig. 4.10.

- $\mathsf{FSCon}(\mathcal{G}, \sigma)$ The data owner utilizes existing frequent subgraph mining algorithms to generate the frequent subgraph set $\mathcal{F}$, and then creates the frequent feature dictionary $\mathcal{D}$ and the feature-based inverted index I_{inv}.

- $\mathsf{KeyGen}(K_S, n)$ With the master key K_S, the data owner generates the secret key $\mathcal{K}$, consisting of the splitting indicator S, two invertible matrices $\{M_1, M_2\}$, and the vector ρ.

- $\mathsf{BuildIndex}(\mathcal{G}, \mathcal{F}, \mathcal{K})$ For each data graph G_i, this algorithm creates the subindex I_i as shown in Fig. 4.3. The data owner first creates a vector g_i with length n, in which the value of $g_{i[j]}$ is determined by whether graph G_i contains the corresponding feature F_j or not (steps 1 and 2). Subsequently, the data vector g_i is processed by applying the dimension extending where the $(n+1)$-th entry in $\vec{g_i}$ is set to 1 (step 3) and further adopting the splitting and encrypting procedures in the secure Euclidean Distance computation scheme (steps 4 and 5). Finally, a subindex $I_i = \{(M_1^{-1})^T \vec{g_i}', (M_2^{-1})^T \vec{g_i}''\}$ is created for every data graph G_i and associated with the encrypted data graph $\widetilde{G}_i$ for outsourcing to the cloud server .

- $\mathsf{TDGen}(Q)$ With the query graph Q as input from the data user, this algorithm outputs the trapdoor T_Q as shown in Fig. 4.4. The query feature set $\mathcal{F}_Q$ is first generated through checking which features in $\mathcal{F}$ are also contained in Q (steps 1 and 2). An n-dimensional vector q is created by assigning a positive random number r_j to the element $q_{[j]}$ if $F_j \in \mathcal{F}_Q$; otherwise, $q_{[j]} = 0$ (step 3). This initial query vector q is then extended to an $(n+1)$-dimensional vector as $\vec{q} = (q, t)$, where t is a non-zero random number (step 4). After adopting the splitting and encrypting processes in the secure Euclidean distance computation technique (steps 5 and 6), the trapdoor T_Q for the query graph Q is generated as $\{M_1\vec{q}', M_2\vec{q}'', \sum \rho_{[j]}q_{[j]} + t\}$, where the third element is the expected final inner product of the query vector and the data vector for every candidate supergraph.

- $\mathsf{Query}(\mathcal{I}, T_Q)$ With the trapdoor T_Q, the cloud server computes the inner product of $\{T_{Q[1]}, T_{Q[2]}\}$ with every subindex I_i for data graph G_i, and returns graph id list $\mathcal{G}_{\mathcal{F}_Q}$ where each graph has an inner product as exactly same as $T_{Q[3]}$. The data user can further do the graph verification to remove false positives from $\mathcal{G}_{\mathcal{F}_Q}$, and finally get the exact result as $\mathcal{G}_{\{Q\}}$.

4.5.3 The Analysis

Analysis of this proposed scheme follows three aspects of design goals described in Section 4.2.3.

4.5.3.1 Effectiveness

Assume Q consists of ℓ query features, i.e., $\ell = |\mathcal{F}_Q|$. For any supergraph G_i of the query graph Q, it includes all the ℓ features in $\mathcal{F}_Q$ which is extracted from Q.

Therefore, all the ℓ corresponding elements in the data vector are equal to those in the ρ, respectively, i.e., $g_{i[j_k]} = \rho_{[j_k]}$, where $1 \leq k \leq \ell$. Besides, each corresponding element in the query vector as $q_{i[j_k]}$ is set as r_{j_k}, and all other elements is set as 0. The final inner product $g_i \cdot q + t$ for any supergraph G_i is then equal to $\sum \rho_{[j_k]} r_{j_k} + t$, which is also the result of $\rho \cdot q + t$. The later one $\rho \cdot q + t$ has been included in the trapdoor and serves as an indicator to select candidate supergraphs. It means that our scheme does not introduce any false negative into the result $\mathcal{G}_{\mathcal{F}_Q}$, as every exact supergraph in $\mathcal{G}_{\{Q\}}$ will produce the same inner product as $\rho \cdot q + t$ with the query vector. But false positive supergraphs may be introduced into $\mathcal{G}_{\mathcal{F}_Q}$ by those data graphs that do not contain the query graph Q but contain all the features in $\mathcal{F}_Q$.

4.5.3.2 Efficiency

As far as the data user is concerned, the query response is well presented because the final inner product for every data graph can be efficiently computed by the cloud server via two multiplications of $(n+1)$-dimensional vectors. The whole inner product computation during query is $O(mn)$. Although some costly computations are involved in FSCon and BuildIndex, such as graph sequentialization, they are unavoidable for building a graph index. And more importantly, they are executed for only one time during the whole scheme. Apart from these computations, the encryption of the data vector or the query vector only needs two multiplications of a $(n+1) \times (n+1)$ matrix and a $(n+1)$-dimensional vector with complexity $O(n^2)$ in BuildIndex or TDGen, respectively. Besides, to avoid the high computation cost of inverting two high-dimension matrices in TDGen, every query vector is encrypted by the two matrices M_1 and M_2 themselves, instead of their inverses of M_1 and M_2 utilized in the secure Euclidean distance computation. Correspondingly, the costly inverting operation is transferred to the one-time index construction procedure.

4.5.3.3 Privacy

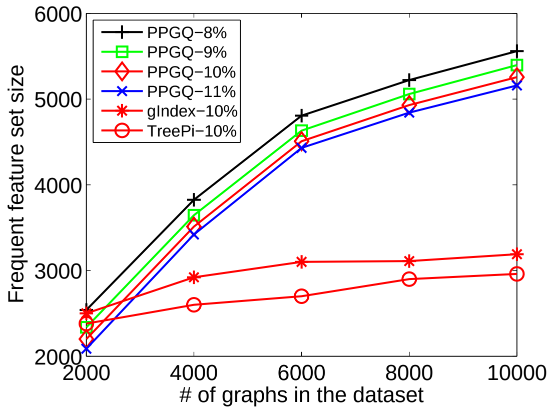
With the randomness introduced by the splitting process and the random numbers r_j and t, our scheme can generate two totally different trapdoors for the same query graph Q. This nondeterministic property of the trapdoor generation can guarantee the *trapdoor unlinkability*.

Recall that the data vector encryption with matrices has been proved to be secure against known-plaintext attack in [120], the *index privacy* is protected unless the secret key $\mathcal{K}$ is disclosed. The number of equations as $2(n+1)m$ in $(M_1^{-1})^T \vec{g_i}' = I_{i[1]}$ and $(M_2^{-1})^T \vec{g_i}'' = I_{i[2]}$ is still less than the number of unknowns as the sum of $2(n+1)m$ unknowns in m data vectors and $2(n+1)^2$ unknowns in $\{M_1, M_2\}$. Therefore, the attacker cannot solve the equations.

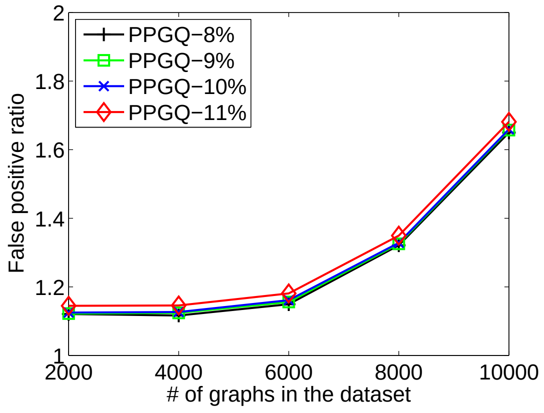
As mentioned above, in the secure inner product computation technique, the primary reason why the statistical analysis attack works is that the final inner product y_i has the scale relationship with λ_i. And this scale relationship exists just because y_i is a multiple of the original inner product $g_i \cdot q$ which is equal to λ_i. Our proposed scheme introduces randomness in both g_i and q to break the equivalence relationship between $g_i \cdot q$ and λ_i. As a consequence, the value of $g_i \cdot q$ does not completely depend on λ_i. In the case where data graph G_i contain fewer query features than data graph G_j, it is still possible that $g_i \cdot q \geq g_j \cdot q$. Moreover, the extended dimension t is utilized to break the direct scale relationship between y_i and $g_i \cdot q$, which further eliminates the indirect scale relationship between y_i and λ_i. So the cloud server cannot deduce the special ratio as $\frac{\lambda_i+1}{\lambda_i}$ which is used to detect the inclusion relationship between two query feature sets as discussed in the previous section 4.5.1. Without disclosing such inclusion relationship, the cloud server cannot compute the support of a single feature. In other words, the statistical analysis cannot break the *feature privacy*, and all the expected privacy requirements in section 4.4.3 are being met by the proposed scheme.

4.6 Experimental Evaluations

In this section, we demonstrate a thorough experimental evaluation of the proposed scheme on the AIDS antiviral screen dataset [5] that is widely used in graph query related works [38, 91, 128, 132, 133]. It contains 42,390 compounds with totally 62 distinct vertex labels. The 5 datasets in our experiment, as same as in [128], are $\mathcal{G}_{2000}$, $\mathcal{G}_{4000}$, $\mathcal{G}_{6000}$, $\mathcal{G}_{8000}$, and $\mathcal{G}_{10000}$, where $\mathcal{G}_N$ contains N graphs randomly chosen from the AIDS dataset. We also adopt the same 6 sets of query graphs Q_4, Q_8, Q_{12}, Q_{16}, Q_{20} and Q_{24}, where Q_i contains i query graphs with i edges. Default dataset and query graphs are set as $\mathcal{G}_{4000}$ and Q_4 in our experiment, respectively. *gSpan* [127] is used as the frequent subgraph mining algorithm in our scheme. The maximum size of frequent subgraph $maxL$ is set to 11, and the minimum support σ for feature F_j is defined as follows, $\sigma = 1$ if $|V(F_j)| < 5$; otherwise, $\sigma = \sqrt{\frac{|V(F_j)|}{maxL}} \cdot minsup \cdot |\mathcal{G}|$, where the default $minsup$ is set to 10% and $|\mathcal{G}|$ is the size of dataset. Graph boosting toolbox [81] is utilized to implement *gSpan* algorithm and check subgraph isomorphism, and the public utility routines by Numerical Recipes are employed to compute the inverse of matrix. The query performance in our scheme is evaluated on the Amazon Elastic Compute Cloud (EC2) in which we deploy the basic 64-bit Linux Amazon Machine Image (AMI) with 4 CPU cores ($2 \times$ 1.2GHz); the performance of other procedures in our scheme, such as index construction and trapdoor generation related to data owners or data users, is evaluated on a 2.8GHz CPU with Redhat Linux. The compared schemes are gIndex [128] and TreePi [132], and their performance data are provided in [132] which are also run on a 2.8GHz CPU with RedHat Linux.



(a) Frequent feature set size

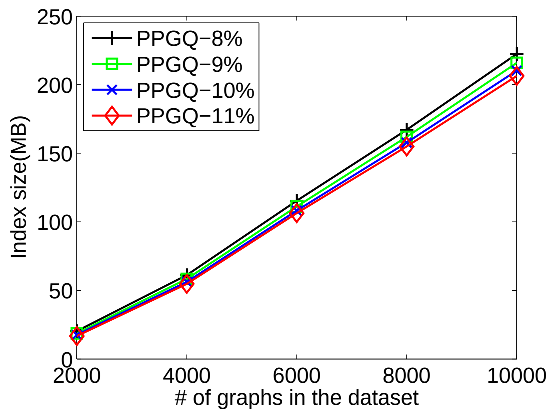


(b) False positive ratio

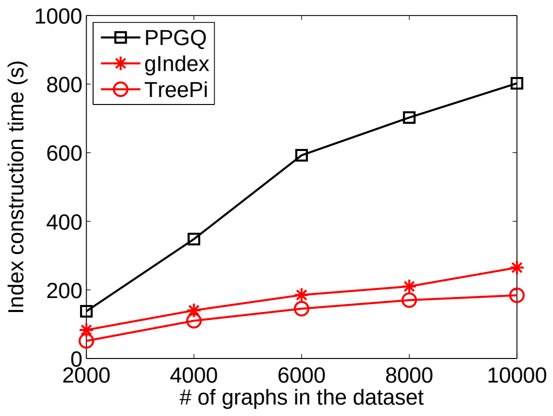
Figure 4.5: Relation between minimum support and other parameters.

4.6.1 False Positive and Index Construction

The minimum support determines the threshold for a subgraph of being an indexed feature. Specifically, the large value of minimum support means that only very frequent subgraphs in the dataset could be treated as valid in the filtering procedure. However, such high requirement will reduce the number of features included in the index whose pruning power would be directly affected. With the decreasing number of indexed features, the query graph can only be represented by less number of query features, and therefore more and more data graphs, which does not contain



(a) Storage cost of index



(b) Time of index construction

Figure 4.6: Index construction cost.

the query graph Q but contain all the graphs in the smaller size query feature set $\mathcal{F}_Q$, are included in the candidate supergraph set $\mathcal{G}_{\mathcal{F}_Q}$. As demonstrated in Fig. 4.5 where four different minimum supports through adjusting *minsup* from 8% to 11% are examined, the false positive ratio defined as $\frac{|\mathcal{G}_{\mathcal{F}_Q}|}{|\mathcal{G}_{\{Q\}}|}$ raises in accordance with the *minsup*. Although the minimum support should be set as small as possible to prune as many data graphs as possible, the larger one will introduce more storage cost of index due to the larger size of frequent feature set as shown in Fig. 4.6(a). Moreover, as shown in Fig. 4.5(a), the size of the frequent feature set increases in a lower speed when the dataset is larger than 600, while the minimum support

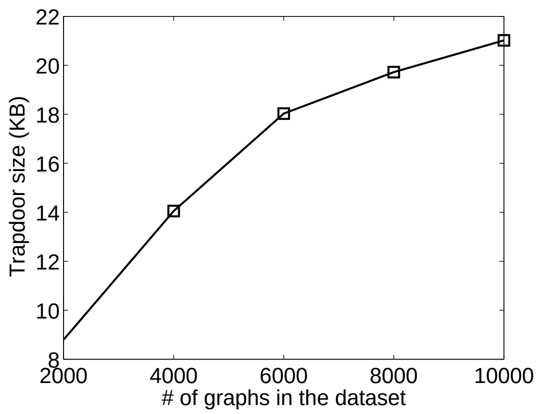


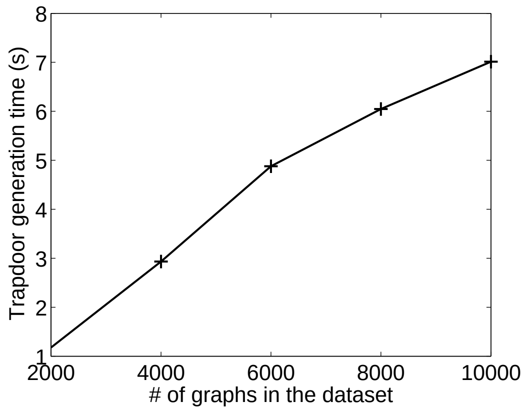
Figure 4.7: Trapdoor size in different dataset

$\sigma = \sqrt{\frac{|V(F_j)|}{maxL}} \cdot minsup \cdot |\mathcal{G}|$ increases linearly with the size of dataset. As a result, there will be increasing false positives in the candidate supergraph set, which is validated in Fig. 4.5(b).

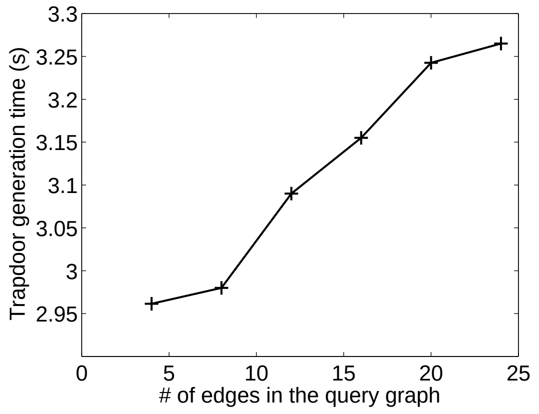
As shown in Fig. 4.5(a), our frequent feature set is larger than that in the other two related works since our scheme does not adopt the shrinking process on the frequent set by choosing discriminative subgraphs. Besides, the false positive ratio in our scheme is almost same as that in gSpan and a little larger than the scheme Tree+$\triangle$ [132], through the performance data provided in [132]. As shown in Fig. 4.6(b), because our index construction involves the encryption process on data vectors, the time cost here is about four times larger than that in other schemes which only deal with plaintext index. Note that this construction is only a one-time procedure in the whole scheme.

4.6.2 Trapdoor Generation and Query

Like index construction, every trapdoor generation incurs two multiplications of a matrix and a split query vector, whose dimensionality becomes larger with the increasing number of documents in dataset. As demonstrated in Fig. 4.8(a), the



(a) Different dataset with same query size as 4

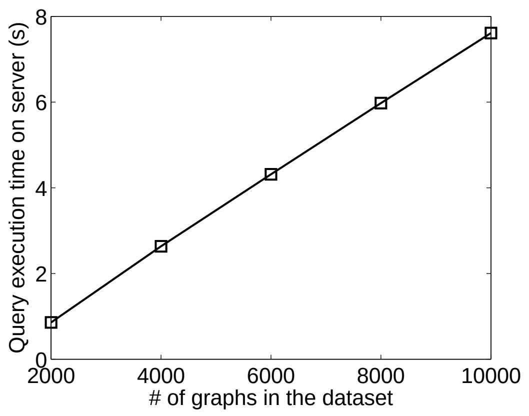


(b) Different query size with same dataset size as 4000

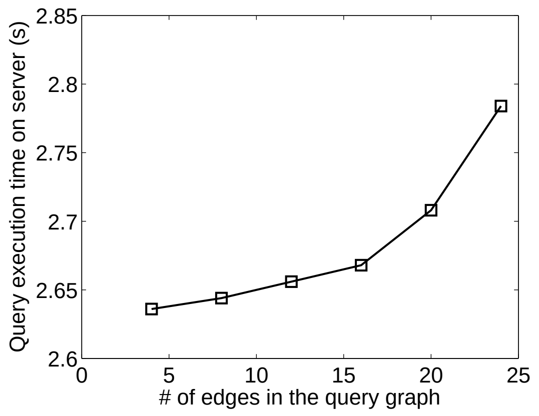
Figure 4.8: Trapdoor generation time.

time to generate a trapdoor is linear with the number of data graphs in the dataset. Fig. 4.8(b) demonstrates the trapdoor generation cost is almost linear with the size of query graph, which is defined as the number of edges in the query graph. Such linearity is caused by the fact that the major costly operation mapping query graph to vector is roughly determined by query size since all the query features should be mapped.

In the query process in our scheme design, the cloud server executes the filtering process by computing the inner product of trapdoor and each encrypted data vector.



(a) Different dataset with same query size as 4



(b) Different query size with same dataset size as 4000

Figure 4.9: Query execution time on server.

Fig. 4.9 shows that the query time is almost linear with the number of data graphs in the dataset. Although the query time in our scheme is much larger than that in gSpan, whose query time is around 100 milliseconds presented in [38], our scheme is performing query on the encrypted index. With respect to the communication cost in the query procedure, the size of trapdoor is the same as that of subindex for single data graph. As shown in Fig. 4.7, the size of trapdoor keeps constant in the same dataset, no matter how many features are contained in a query graph.

4.7 Related Work

4.7.1 Graph Containment Query

To reduce the computation cost caused by checking subgraph isomorphism, most research on plaintext graph containment query problem follows the "filtering-and-verification" framework [38, 91, 92, 128, 132, 133] to decrease the size of candidate supergraph set. Feature-based index has been increasingly explored by choosing different substructures as features. Shasha et al. [92] designed a path-based index approach. However, paths carry few structural information and therefore have limited filtering power. Yan et al. [128] proposed gIndex to build index from frequent and discriminative subgraphs which can carry more structure characteristics. Zhang et al. [132] utilized frequent and discriminative subtrees instead of subgraphs to build the index. Recently, Chase and Kamara proposed structured encryption [36] to handle private access to parts of a large graph in encrypted form; yet only simple operations such as neighbor queries are supported. Portions of the work studied in this chapter were presented as extended abstract at the 31th International Conference on Distributed Computing Systems (ICDCS'11) [31].

4.7.2 Keyword-based Searchable Encryption

Traditional single keyword searchable encryption schemes [22, 40, 65, 106] usually build an encrypted searchable index such that its content is hidden to the server unless it is given appropriate trapdoors generated via secret key(s) [58]. To enrich search semantics, conjunctive keyword search [24] over encrypted data have been proposed. These schemes incur large overhead caused by their fundamental primitives, such as computation cost by bilinear map [24].As a more general search approach, predicate encryption schemes [63] are recently proposed to support both conjunctive and disjunctive search. However, none of existing boolean keyword searchable encryption schemes support graph semantics as we propose to explore in this chapter.

4.8 Conclusion

In this chapter, for the first time, we define and solve the problem of query over encrypted graph-structured cloud data, and establish a variety of privacy requirements. For the efficiency consideration, we adopt the principle of "filtering-and-verification" to prune as many negative data graphs as possible before verification, where a feature-based index is pre-built to provide feature-related information for every encrypted data graph. Then, we choose the inner product as the pruning tool to carry out the filtering procedure efficiently. To meet the challenge of supporting graph semantics, we propose a secure inner product computation technique, and then improve it to achieve various privacy requirements under the known-background threat model. Thorough analysis investigating privacy and efficiency of our scheme is given, and the evaluation further shows our scheme introduces low overhead on computation and communication.

```
FSCon($\mathcal{G}, \sigma$)
  1. Mine frequent feature set $\mathcal{F} = \{F_1, F_2, \ldots, F_n\}$ from graph dataset $\mathcal{G}$ with the minimum support threshold $\sigma$;
     i)  For each frequent feature $F_j$, where $1 \leq j \leq n$, generate its supergraph id set $\mathcal{G}_{\{F_j\}}$;
  2. Create the frequent feature dictionary $\mathcal{D} = \{\mathcal{L}_{F_1}, \mathcal{L}_{F_2}, \ldots, \mathcal{L}_{F_n}\}$, where $\mathcal{L}_{F_j}$ is the unique canonical label of $F_j$;
  3. Build the feature-based inverted index $I_{inv} = \{\mathcal{G}_{\{F_1\}}, \mathcal{G}_{\{F_2\}}, \ldots, \mathcal{G}_{\{F_n\}}\}$.
KeyGen($K_S, n$)
  1. Create an $(n+1)$-bit vector $S$, two $(n+1) \times (n+1)$ invertible matrices $M_1, M_2$, and an $n$-dimensional vector $\rho$;
     i)  $\{S, M_1, M_2, \rho\} \overset{\mathcal{R}}{\leftarrow} K_S$;
  2. Output the secret key $\mathcal{K} = \{S, M_1, M_2, \rho\}$.
BuildIndex($I_{inv}, \mathcal{K}$)
  1. Create a $n \times n$ matrix $X$, where $X_{[i][j]}$ is a random number less than $\mathcal{K}_{[4][j]}$;
  2. For each graph $G_i$, where $1 \leq i \leq m$, create a $n$-dimensional data vector $g_i$;
     i)  If $id(G_i) \in I_{inv[j]}$, set $g_{i[j]} = \mathcal{K}_{[4][j]}$; otherwise, set $g_{i[j]} = X_{[i][j]}$;
  3. Extend every $g_i$ to $(n+1)$-dimensional $\vec{g_i}$;
     i)  For $1 \leq j \leq n$, set $\vec{g}_{i[j]} = g_{i[j]}$; Set $\vec{g}_{i[n+1]} = 1$;
  4. According to the splitting indicator $\mathcal{K}_{[1]}$, split every $\vec{g_i}$ to two vectors $\vec{g_i}'$ and $\vec{g_i}''$;
     i)  For $1 \leq j \leq n+1$, if $\mathcal{K}_{[1][j]} = 0$, set both $\vec{g_i}'_{[j]}$ and $\vec{g_i}''_{[j]}$ as $\vec{g}_{i[j]}$;
           otherwise, set $\vec{g_i}'_{[j]}$ and $\vec{g_i}''_{[j]}$ as two random numbers such that $\vec{g_i}'_{[j]} + \vec{g_i}''_{[j]} = \vec{g}_{i[j]}$;
  5. Encrypt these two vectors by inverses of the two matrices, and combine them as the subindex $I_i$ for $G_i$;
     i)  $I_i = \{((\mathcal{K}_{[2]})^{-1})^T \vec{g_i}', ((\mathcal{K}_{[3]})^{-1})^T \vec{g_i}''\}$;
  6. Output the encrypted index $I = \{I_1, I_2, \cdots, I_m\}$.
TDGen($Q, \mathcal{D}, \mathcal{K}$)
  1. Initialize the query feature set: $\mathcal{F}_Q = \emptyset$;
  2. For each frequent feature $\mathcal{D}_{[j]}$, $1 \leq j \leq n$: if $\mathcal{D}_{[j]} \subseteq Q$, $\mathcal{F}_Q = \mathcal{F}_Q \bigcup \{\mathcal{D}_{[j]}\}$;
  3. Create a $n$-dimensional query vector $q$ for the input query graph $Q$;
     i)  Generate $n$ positive random numbers as $r_1, r_2, \ldots, r_n$;
     ii) For $1 \leq j \leq n$, if $\mathcal{D}_{[j]} \in \mathcal{F}_Q$, set $q_{[j]} = r_j$; otherwise, set $q_{[j]} = 0$;
  4. Extend $q$ to $(n+1)$-dimensional $\vec{q}$, and generate a random number $t$;
     i)  For $1 \leq j \leq n$, set $\vec{q}_{[j]} = q_{[j]}$; Set $\vec{q}_{[n+1]} = t$;
  5. According to the splitting indicator $\mathcal{K}_{[1]}$, split $\vec{q}$ to two vectors as $\vec{q}'$ and $\vec{q}''$;
     i)  For $1 \leq j \leq n+1$, if $\mathcal{K}_{[1][j]} = 1$, set both $\vec{q}'_{[j]}$ and $\vec{q}''_{[j]}$ as $\vec{q}_{[j]}$;
           otherwise, set $\vec{q}'_{[j]}$ and $\vec{q}''_{[j]}$ as two random numbers such that $\vec{q}'_{[j]} + \vec{q}''_{[j]} = \vec{q}_{[j]}$;
  6. Encrypt these two vectors by the two invertible matrices as $\{(\mathcal{K}_{[2]})^T \vec{q}', (\mathcal{K}_{[3]})^T \vec{q}''\}$;
  7. Output the trapdoor $T_Q$ for query graph $Q$;
     i)  $T_Q = \{(\mathcal{K}_{[2]})^T \vec{q}', (\mathcal{K}_{[3]})^T \vec{q}'', \sum \mathcal{K}_{[4][j]} q_{[j]} + t\}$;
Query($\mathcal{I}, T_Q$)
  1. Initialize the query result: $\mathcal{G}_{\mathcal{F}_Q} = \emptyset$;
  2. For each subindex $I_{[i]}$, $1 \leq i \leq m$:
     i)  Compute inner product as $I_{i[1]} \cdot T_{Q[1]} + I_{i[2]} \cdot T_{Q[2]}$;
     ii) If the inner product is equal to $T_{Q[3]}$, set $\mathcal{G}_{\mathcal{F}_Q} = \mathcal{G}_{\mathcal{F}_Q} \bigcup \{\text{id}(G_i)\}$;
  3. Output the query result $\mathcal{G}_{\mathcal{F}_Q}$.
```

Figure 4.10: Privacy-Preserving Graph Query Scheme

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this dissertation, we investigated and addressed the fundamental problem of secure and reliable data outsourcing in Cloud Computing. We summarize our results as follows.

In Chapter 2, we address the problem of secure and reliable cloud storage with efficiency consideration of both data repair and data retrieval. By utilizing a near-optimal erasure codes, specifically LT codes, our designed storage service has faster decoding during data retrieval than existing solutions. To minimize the data repair complexity, we employ the exact repair method to efficiently recover the exact form of any corrupted data. Such a design also reduces the data owner's cost during data repair since no verification tag needs to be generated (old verification tags can be recovered as data recovery). Our proposed cloud storage service provides a better overall efficiency of data retrieval and repair than existing counterparts. It also greatly reduces cost and completely releases the data owner from the burden of being online by enabling public integrity check and exact repair.

In Chapter 3, we address the problem of privacy-preserving multi-keyword ranked search over encrypted data in cloud computing. We establish a set of strict privacy requirements for such a secure cloud data utilization system. Among various multi-keyword semantics, we choose the efficient similarity measure of "coordinate matching", i.e., as many matches as possible, to capture the relevance of data documents to the search query. We further use "inner product similarity" to quantitatively evaluate such similarity measure. We first propose a basic idea for ranked keyword search based on secure inner product computation, and then give two significantly improved schemes to achieve various stringent privacy requirements in two different threat models. We also investigate some further enhancements of our ranked search mechanism, including supporting more search semantics, i.e., TF $\times$ IDF, and dynamic data operations.

In Chapter 4, we address the problem of privacy-preserving query over encrypted graph-structured data in cloud computing. Our work utilizes the principle of "filtering-and-verification". We pre-build a feature-based index to provide feature-related information about each encrypted data graph, and then choose the efficient inner product as the pruning tool to carry out the filtering procedure. To meet the challenge of supporting graph query without privacy breaches, we propose a secure inner product computation technique, and then improve it to achieve various privacy requirements under the known-background threat model.

5.2 Future Work

We identify a few challenging issues for future work on secure and reliable data outsourcing in cloud computing as follows.

As presented in our proposed cloud data storage service, the availability in data retrieval is guaranteed by the decodability detection before data outsourcing and the exact repair of corrupted data. Once the encoding configuration successfully passes the decodability detection, it can be reused for all the storage services that specifies the same reliability level in terms of n and k. However, such detection computation still takes nonnegligible cost when every cloud user may have its own expected reliability requirement. We plan to investigate more efficient decodability detection algorithm which will make such cloud storage solution more practical. Besides, in cloud computing, the outsourced data might not only be accessed but also updated by the data owners, e.g., through block modification, deletion and insertion, etc. Hence, we also plan to investigate supporting dynamic data operations which can be of vital importance to the practical application of data outsourcing services.

In our proposed data utilization solutions as presented in chapter 3 and chapter 4, the query computation cost in the server side is linear with the number of documents in the dataset. Currently data owners outsource more and more data into cloud servers, so it is of practical use to make the query faster. To address this problem, we plan to explore more efficient search algorithm based on tree structures [73, 83]. We further plan to investigate new security and privacy problems in the untrusted cloud server model. In practice, cloud servers may sometimes behave beyond the known background model. This can happen either because cloud server intentionally wants to do so for saving cost when handling large number of search requests, or there may be software bugs, or internal/external attacks. Thus, enabling a search result authentication mechanism that can detect such unexpected behaviors of cloud server is also of practical interest and worth further investigation. Our early work has been aware of this problem, and provided a solution to authenticating ranked search result [104], but only for single keyword search.

Bibliography

[1] `http://www.dropbox.com/`.

[2] Blog service hosted by google crashes review. `http://hostwisely.com/blog/blog-service-hosted-by-google-crashes/`.

[3] Microsoft cloud data breach heralds things to come. `http://www.techworld.com.au/article/372111/`.

[4] Summary of the amazon ec2 and amazon rds service disruption in the us east region. `http://aws.amazon.com/message/65648/`.

[5] Aids antiviral screen dataset, 1999. `http://dtp.nci.nih.gov/docs/aids/aids_data.html`.

[6] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. In *CRYPTO 2005*, pages 205–222. Springer-Verlag, 2005.

[7] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. *J. Cryptol.*, 21(3):350–391, 2008.

[8] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order preserving encryption for numeric data. In *Proceedings of SIGMOD '04*, pages 563–574, 2004.

[9] Georgios Amanatidis, Alexandra Boldyreva, and Adam O'Neill. New security models and provably-secure schemes for basic query support in outsourced databases.

[10] Georgios Amanatidis, Alexandra Boldyreva, and Adam O'Neill. Provably-secure schemes for basic query support in outsourced databases. In *DBSec'07: Proceedings of the 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security*. Springer-Verlag, 2007.

[11] Michael Armbrust, Armando Fox, and et al. Above the clouds: A berkeley view of cloud computing. Technical Report UCB-EECS-2009-28, University of California, Berkeley.

[12] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proceedings of CCS*, New York, NY, USA, 2007. ACM.

[13] Giuseppe Ateniese, Roberto Di Pietro, Luigi V. Mancini, and Gene Tsudik. Scalable and efficient provable data possession. In *Proceedings of the 4th international conference on Security and privacy in communication netowrks*, SecureComm '08, pages 9:1–9:10, New York, NY, USA, 2008. ACM.

[14] Joonsang Baek, Reihaneh Safiavi-naini, and Willy Susilo. Public key encryption with keyword search revisited. In *Cryptology ePrint Archive, Report 2005/151*, 2005.

[15] Joonsang Baek, Reihaneh Safiavi-naini, and Willy Susilo. Public key encryption with keyword search revisited. In *Computational Science and Its ApplicationsCICCSA 2008*, 2008.

[16] L. Ballard, S. Kamara, and F. Monrose. Achieving efficient conjunctive keyword searches over encrypted data. In *Proc. of ICICS*, 2005.

[17] Feng Bao, Robert Deng, Xuhua Ding, and Yanjiang Yang. Private query on encrypted data in multi-user settings. In *Proc. of ISPEC*, 2008.

[18] Mihir Bellare, Alexandra Boldyreva, and Adam ONeill. Deterministic and efficiently searchable encryption. In *Proc. of CRYPTO*, 2007.

[19] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography: The case of hashing and signing. In *Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '94, pages 216–233, London, UK, UK, 1994. Springer-Verlag.

[20] S. Berreti, A.D. Bimbo, and E. Vicario. Efficient matching and indexing of graph models in content-based retrieval. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 2001.

[21] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-preserving symmetric encryption. In *Proceedings of Eurocrypt'09, volume 5479 of LNCS*. Springer, 2009.

[22] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *Proc. of EUROCRYPT*, 2004.

[23] Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E. Skeith III. Public key encryption that allows pir queries. In *Proc. of CRYPTO*, 2007.

[24] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *Proc. of TCC*, pages 535–554, 2007.

[25] Kevin D. Bowers, Ari Juels, and Alina Oprea. Hail: a high-availability and integrity layer for cloud storage. In *Proc. of CCS*, 2009.

[26] Kevin D. Bowers, Ari Juels, and Alina Oprea. Proofs of retrievability: theory and implementation. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, CCSW '09, pages 43–54, New York, NY, USA, 2009. ACM.

[27] Xavier Boyen and Brent Waters. Anonymous hierarchical identity-based encryption (without random oracles). In *Proc. of CRYPTO*. LNCS, vol. 4117. Springer,Heidelberg, 2006.

[28] R. Brinkman, J. Doumen, and W. Jonker. Using secret sharing for searching in encrypted data. In *Secure Data Management*, 2004.

[29] Richard Brinkman. Searching in encrypted data. In *University of Twente, PhD thesis*, 2007.

[30] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou. Privacy-preserving multi-keyword ranked search over encrypted cloud data. In *Proc. of INFOCOM*, 2011.

[31] Ning Cao, Zhenyu Yang, Cong Wang, Kui Ren, and Wenjing Lou. Privacy-preserving query over encrypted graph-structured data in cloud computing. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 393 –402, june 2011.

[32] Ning Cao, Shucheng Yu, Zhenyu Yang, Wenjing Lou, and Y.T. Hou. Lt codes-based secure and reliable cloud storage service. In *INFOCOM, 2012 Proceedings IEEE*, pages 693 –701, march 2012.

[33] Mary-Luc Champel, Kevin Huguenin, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. Lt network codes. *Proc. of ICDCS*, pages 536–546, 2010.

[34] Ee-Chien Chang and Jia Xu. Remote integrity check with dishonest storage server. In Sushil Jajodia and Javier Lopez, editors, *Computer Security - ESORICS 2008*, volume 5283 of *Lecture Notes in Computer Science*, pages 223–237. Springer Berlin / Heidelberg, 2008.

[35] Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Proc. of ACNS*, 2005.

[36] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *Proc. of ASIACRYPT*, 2010.

[37] Bo Chen, Reza Curtmola, Giuseppe Ateniese, and Randal Burns. Remote data checking for network coding-based distributed storage systems. In *Proc. of CCSW '10*, 2010.

[38] J. Cheng, Y. Ke, W. Ng, and A. Lu. Fg-index: towards verification-free query processing on graph databases. In *Proc. of SIGMOD*, 2007.

[39] W. W. Cohen. Enron email dataset. `http://www.cs.cmu.edu/~enron/`.

[40] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proc. of ACM CCS*, 2006.

[41] Reza Curtmola, Osama Khan, Randal Burns, and Giuseppe Ateniese. Mr-pdp: Multiple-replica provable data possession. In *Proc. of ICDCS*, 2008.

[42] A. G. Dimakis D. Cullina and T. Ho. Searching for minimum storage regenerating codes. In *IN ALLERTON CONFERENCE ON CONTROL, COMPUTING, AND COMMUNICATION*, 2009.

[43] A.G. Dimakis, P.B. Godfrey, Yunnan Wu, M.J. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. *ITIT*, 2010.

[44] A.G. Dimakis, K. Ramchandran, Yunnan Wu, and Changho Suh. A survey on network codes for distributed storage. *Proceedings of the IEEE*, 99(3):476 –489, march 2011.

[45] Yevgeniy Dodis, Salil Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification. In *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, TCC '09, pages 109–127, Berlin, Heidelberg, 2009. Springer-Verlag.

[46] A. Duminuco and E. Biersack. A practical study of regenerating codes for peer-to-peer backup systems. In *Proc. of ICDCS*, June 2009.

[47] Chris Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 213–222, New York, NY, USA, 2009. ACM.

[48] Daniel Fabbri, Arnab Nandi, Kristen Lefevre, and H. V. Jagadish. Privatepond: Outsourced management of web corpuses, 2009.

[49] Dcio Luiz Gazzoni Filho and Paulo Srgio Licciardi Messeder Barreto. Demonstrating data possession and uncheatable data transfer. Technical report, 2006.

[50] M. R. Garey and D. S. Johnson. Computers and intractability: A guide to the theory of np-completeness. Freeman, New York, NY, USA, 1990.

[51] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, 2003. `http://eprint.iacr.org/2003/216`.

[52] P. Golle, J. Staddon, and B. Waters. Secure conjunctive keyword search over encrypted data. In *Proc. of ACNS*, pages 31–45, 2004.

[53] T. Ho, M. Medard, R. Koetter, D.R. Karger, M. Effros, Jun Shi, and B. Leong. A random linear network coding approach to multicast. *ITIT*, 2006.

[54] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Proc. of ICDM*, 2003.

[55] Y.H. Hwang and P.J. Lee. Public key encryption with conjunctive keyword search and its extension to a multi-user system. In *Pairing*, 2007.

[56] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography from anonymity. In *Proc. of FOCS*, pages 239–248, 2006.

[57] Ari Juels and Burton S. Kaliski, Jr. Pors: proofs of retrievability for large files. In *Pro. of CCS*, pages 584–597, New York, NY, USA, 2007. ACM.

[58] S. Kamara and K. Lauter. Cryptographic cloud storage. In *RLCPS*, 2010.

[59] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Proc. of EUROCRYPT*, 2008.

[60] Ranjita Bhagwan Kiran, Kiran Tati, Yu chung Cheng, Stefan Savage, and Geoffrey M. Voelker. Total recall: System support for automated availability management. In *Proc. of NSDI*, 2004.

[61] John Kubiatowicz, David Bindel, Yan Chen, and et al. OceanStore: an architecture for global-scale persistent storage. In *ASPLOS*, New York, NY, USA, 2000. ACM.

[62] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, page 364, Washington, DC, USA, 1997. IEEE Computer Society.

[63] Allison Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *Proc. of EUROCRYPT*, 2010.

[64] Jin Li, Qian Wang, Cong Wang, Ning Cao, Kui Ren, and Wenjing Lou. Enabling efficient fuzzy keyword search over encrypted data in cloud computing. Cryptology ePrint Archive, 2009. `http://eprint.iacr.org/2009/593`.

[65] Jin Li, Qian Wang, Cong Wang, Ning Cao, Kui Ren, and Wenjing Lou. Fuzzy keyword search over encrypted data in cloud computing. In *Proc. of IEEE INFOCOM'10 Mini-Conference*, San Diego, CA, USA, March 2010.

[66] Jun Li, Shuang Yang, Xin Wang, Xiangyang Xue, and Baochun Li. Tree-structured data regeneration in distributed storage systems with regenerating codes. In *Proc. of IWQoS*, July 2009.

[67] Ming Li, Shucheng Yu, Ning Cao, and Wenjing Lou. Authorized private keyword search over encrypted data in cloud computing. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 383 –392, june 2011.

[68] Mark Lillibridge, Sameh Elnikety, Andrew Birrell, Mike Burrows, and Michael Isard. A cooperative internet backup scheme. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '03, pages 3–3, Berkeley, CA, USA, 2003. USENIX Association.

[69] M. Luby. Lt codes. In *Proc. of FoCS*, pages 271–280, 2002.

[70] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman. Efficient erasure correcting codes. *ITIT*, (2):569–584, 2001.

[71] M. Naor and G.N. Rothblum. The complexity of online memory checking. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 573 – 582, oct. 2005.

[72] S. Nijssen and J. N. Kok. A quickstart in frequent structure mining can make a difference. In *Proc. of SIGKDD*, 2004.

[73] M. Ondreicka and J. Pokorny. Extending fagins algorithm for more users based on multidimensional b-tree. In Paolo Atzeni, Albertas Caplinskas, and Hannu Jaakkola, editors, *Advances in Databases and Information Systems*, volume 5207 of *Lecture Notes in Computer Science*, pages 199–214. 2008.

[74] Alina Oprea, Michael K. Reiter, and Ke Yang. Space-efficient block storage integrity. In *In Proc. of NDSS 05*, 2005.

[75] Rafail Ostrovsky and William E. Skeith. Private searching on streaming data. *Journal of Cryptology*, 20(4):397–430, October 2007.

[76] D. Park, J. Cha, and P. Lee. Searchable keyword-based encryption. In *Cryptology ePrint Archive, Report 2005/367*, 2005.

[77] D.J. Park, K. Kim, and P.J. Lee. Public key encryption with conjunctive field keyword search. In *WISA*, 2004.

[78] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran. Exact regenerating codes for distributed storage. In *Proc. Allerton Conf. Control Comput. Commun.*, pages 337–350, 2009.

[79] I. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the SIAM*, 1960.

[80] Eun-Kyung Ryu and Tsuyoshi Takagi. Efficient conjunctive keyword-searchable encryption. *Advanced Information Networking and Applications Workshops, International Conference on*, 2007.

[81] H. Saigo, S. Nowozin, T. Kadowaki, T. Kudo, and K. Tsuda. Gboost: A mathematical programming approach to graph classification and regression. In *Machine Learning*, 2008.

[82] Peter Sanders, Sebastian Egner, and Ludo Tolhuizen. Polynomial time algorithms for network information flow. In *Proc. of SPAA*, pages 286–294, 2003.

[83] Peter Scheuermann and Mohamed Ouksel. Multidimensional b-trees for associative searching in database systems. *Information Systems*, 7(2):123 – 137, 1982.

[84] Mathew J. Schwartz. 6 worst data breaches of 2011, 2011. `http://www.informationweek.com/news/security/attacks/232301079`.

[85] T.S.J. Schwarz and E.L. Miller. Store, forget, and check: Using algebraic signatures to check remotely administered storage. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, page 12, 2006.

[86] Saeed Sedghi, Jeroen Doumen, Pieter Hartel, and Willem Jonker. Towards an information theoretic analysis of searchable encryption. In *Proceedings of the 10th International Conference on Information and Communications Security*, ICICS '08, pages 345–360, Berlin, Heidelberg, 2008. Springer-Verlag.

[87] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *Proceedings of Asiacrypt*, 2008.

[88] Mehul A. Shah, Mary Baker, Jeffrey C. Mogul, and Ram Swaminathan. Auditing to keep online storage services honest. In *Proceedings of the 11th USENIX workshop on Hot topics in operating systems*, HOTOS'07, pages 11:1–11:6, Berkeley, CA, USA, 2007. USENIX Association.

[89] Mehul A. Shah, Ram Swaminathan, and Mary Baker. Privacy-preserving audit and extraction of digital contents, cryptology eprint archive, report 2008/186, 2008.

[90] N.B. Shah, K.V. Rashmi, P.V. Kumar, and K. Ramchandran. Explicit codes minimizing repair bandwidth for distributed storage. In *Information Theory Workshop (ITW), 2010 IEEE*, pages 1 –5, jan. 2010.

[91] H. Shang, Y. Zhang, X. Lin, and J. X. Yu. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. In *Proc. of VLDB*, 2008.

[92] D. Shasha, J.T-L Wang, and R. Giugno. Algorithmics and applications of tree and graph searching. In *Proc. of PODS*, 2002.

[93] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In *Proc. of TCC*, 2009.

[94] E. Shi, J. Bethencourt, T.-H.H. Chan, Dawn Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *Security and Privacy, 2007. SP '07. IEEE Symposium on*, pages 350 –364, may 2007.

[95] Elaine Shi. Evaluating predicates over encrypted data. In *CMU-CS-08-166, PhD thesis*, 2008.

[96] A. Singhal. Modern information retrieval: A brief overview. *IEEE Data Engineering Bulletin*, 24(4):35–43, 2001.

[97] Radu Sion and Bogdan Carbunar. Conjunctive keyword search on encrypted data with completeness and computational privacy. In *Cryptology ePrint Archive, Report 2005/172*, 2005.

[98] Dawn Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Proc. of S&P*, 2000.

[99] Aaron Souppouris. Linkedin investigating reports that 6.46 million hashed passwords have leaked online, 2012. `http://www.theverge.com/2012/6/6/3067523/linkedin-password-leak-online`.

[100] Darlene Storm. Epsilon breach: hack of the century?, 2011. `http://blogs.computerworld.com/18079/epsilon_breach_hack_of_the_century`.

[101] C. Suh and K. Ramchandran. Exact regeneration codes for distributed storage repair using interference alignment. In *Proc. IEEE Int. Symp. Inf. Theory*, 2010.

[102] Ashwin Swaminathan, Yinian Mao, Guan-Ming Su, Hongmei Gou, Avinash L. Varna, Shan He, Min Wu, and Douglas W. Oard. Confidentiality-preserving rank-ordered search. In *Proceedings of the 2007 ACM workshop on Storage security and survivability*, StorageSS '07, pages 7–12, New York, NY, USA, 2007. ACM.

[103] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, 2009.

[104] C. Wang, N. Cao, K. Ren, and W. Lou. Enabling secure and efficient ranked keyword search over outsourced cloud data. *Parallel and Distributed Systems, IEEE Transactions on*, PP(99):1, 2012.

[105] C. Wang, S. Chow, Q. Wang, K. Ren, and W. Lou. Privacy-preserving public auditing for secure cloud storage. *Computers, IEEE Transactions on*, PP(99):1, 2011.

[106] Cong Wang, Ning Cao, Jin Li, Kui Ren, and Wenjing Lou. Secure ranked keyword search over encrypted cloud data. In *Proc. of ICDCS*, 2010.

[107] Cong Wang, Kui Ren, Wenjing Lou, and Jin Li. Toward publicly auditable secure cloud data storage services. *Network, IEEE*, 24(4):19 –24, july-august 2010.

[108] Cong Wang, Kui Ren, Shucheng Yu, and K.M.R. Urs. Achieving usable and privacy-assured similarity search over outsourced cloud data. In *INFOCOM, 2012 Proceedings IEEE*, pages 451 –459, march 2012.

[109] Cong Wang, Qian Wang, Kui Ren, Ning Cao, and Wenjing Lou. Toward secure and dependable storage services in cloud computing. *Services Computing, IEEE Transactions on*, 5(2):220 –232, april-june 2012.

[110] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou. Ensuring data storage security in cloud computing. In *Proc. of IWQoS*, 2009.

[111] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for data storage security in cloud computing. In *INFOCOM, 2010 Proceedings IEEE*, pages 1 –9, march 2010.

[112] Qian Wang, Cong Wang, Jin Li, Kui Ren, and Wenjing Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In Michael Backes and Peng Ning, editors, *Computer Security – ESORICS 2009*, volume 5789 of *Lecture Notes in Computer Science*, pages 355–370. Springer Berlin / Heidelberg, 2009.

[113] Qian Wang, Cong Wang, Kui Ren, Wenjing Lou, and Jin Li. Enabling public auditability and data dynamics for storage security in cloud computing. *Parallel and Distributed Systems, IEEE Transactions on*, 22(5):847 –859, may 2011.

[114] Shuhong Wang, Xuhua Ding, Robert H. Deng, and Feng Bao. Private information retrieval using trusted hardware. In *In ESORICS 2006, September 2006. LNCS*, page 4189.

[115] Brent Waters, D Balfanz, G Durfee, and D.K. Smetters. Building an encrypted and searchable audit log. In *Proc. of NDSS*, 2004.

[116] Hakim Weatherspoon and John D. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *IPTPS*, 2002.

[117] Zack Whittaker. Amazon web services suffers partial outage. `http://www.zdnet.com/blog/btl/amazon-web-services-suffers-partial-outage/79981`.

[118] Peter Williams, Radu Sion, and Bogdan Carbunar. Building castles out of mud: practical access pattern privacy and correctness on untrusted storage. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 139–148, New York, NY, USA, 2008. ACM.

[119] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. Managing gigabytes: Compressing and indexing documents and images. Morgan Kaufmann Publishing, San Francisco, May 1999.

[120] W. K. Wong, David W. Cheung, Ben Kao, and Nikos Mamoulis. Secure knn computation on encrypted databases. In *Proc. of SIGMOD*, 2009.

[121] Y. Wu. A construction of systematic mds codes with minimum repair bandwidth. In *IEEE Trans. Inf. Theory*, 2009.

[122] Yunnan Wu. Existence and construction of capacity-achieving network codes for distributed storage. In *Information Theory, 2009. ISIT 2009. IEEE International Symposium on*, pages 1150–1154, 28 2009-july 3 2009.

[123] Yunnan Wu. Existence and construction of capacity-achieving network codes for distributed storage. *JSAC*, 28(2):277 –288, 2010.

[124] Yunnan Wu and A.G. Dimakis. Reducing repair traffic for erasure coding-based storage via interference alignment. In *Information Theory, 2009. ISIT 2009. IEEE International Symposium on*, pages 2276 –2280, 28 2009-july 3 2009.

[125] Yunnan Wu, Ros Dimakis, and Kannan Ramchandran. Deterministic regenerating codes for distributed storage. In *IN ALLERTON CONFERENCE ON CONTROL, COMPUTING, AND COMMUNICATION*, 2007.

[126] Huaxia Xia and Andrew A. Chien. Robustore: a distributed storage architecture with robust and high performance. In *Proc. of Supercomputing*, 2007.

[127] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *Proc. of ICDM*, 2002.

[128] X. Yan, P. S. Yu, and J. Han. Graph indexing: a frequent structurebased approach. In *Proc. of SIGMOD*, 2004.

[129] S. Yu, C. Wang, K. Ren, and W. Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *Proc. of INFOCOM*, 2010.

[130] S. Zerr, D. Olmedilla, W. Nejdl, and W. Siberski. Zerber+r: Top-k retrieval from a confidential index. In *Proc. of EDBT*, 2009.

[131] Sergej Zerr, Elena Demidova, Daniel Olmedilla, Wolfgang Nejdl, Marianne Winslett, and Soumyadeb Mitra. Zerber: r-confidential indexing for distributed documents. In *Proc. of EDBT*, pages 287–298, 2008.

[132] S. Zhang, M. Hu, and J. Yang. Treepi: A novel graph indexing method. In *Proc. of ICDE*, 2007.

[133] P. Zhao, J. X. Yu, and P. S. Yu. Graph indexing: tree + delta >= graph. In *Proc. of VLDB*, 2007.

[134] Justin Zobel and Alistair Moffat. Exploring the similarity space. *SIGIR FORUM*, 32:18–34, 1998.