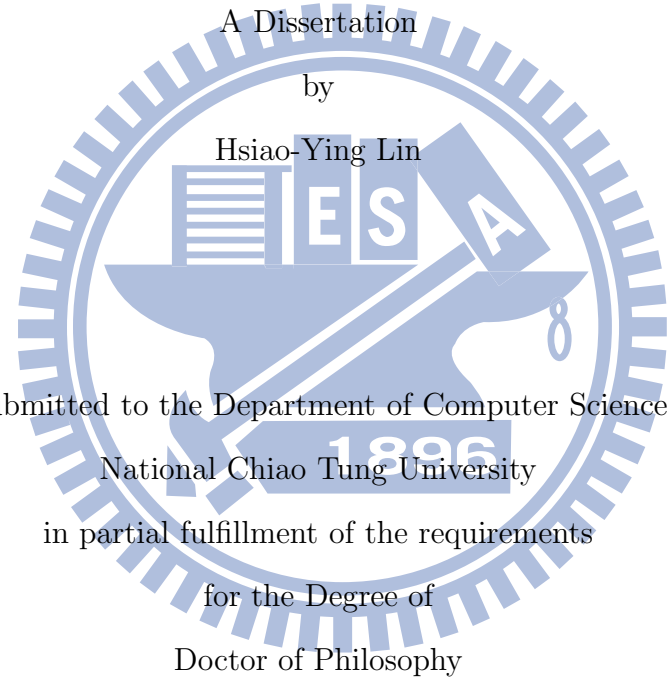
Data Confidentiality and Robustness in Decentralized Cloud Storage Systems

A Dissertation

by

Hsiao-Ying Lin



Submitted to the Department of Computer Science

National Chiao Tung University

in partial fulfillment of the requirements

for the Degree of

Doctor of Philosophy

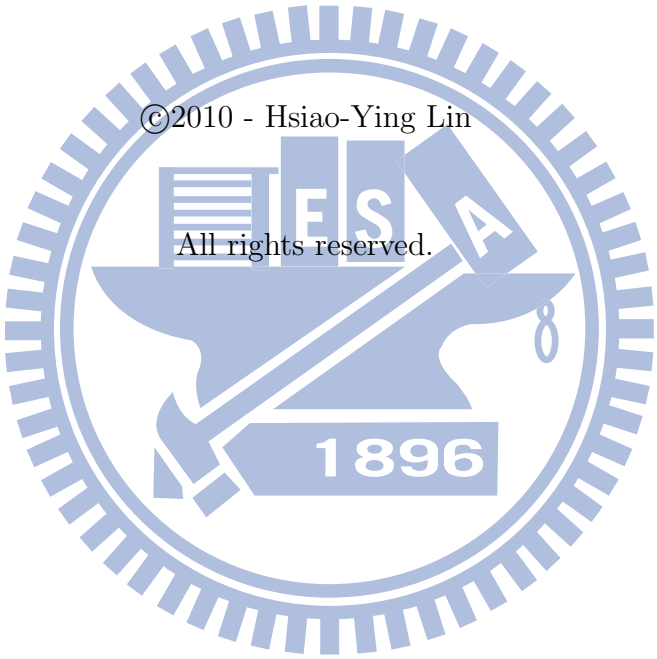
in

Computer Science

Dissertation Director: Wen-Guey Tzeng

May 2010

Hsinchu, Taiwan, Republic of China



Abstract

A cloud storage system consisting of a collection of storage servers provides storage services over the Internet for long-term storage. A user can store data into the system and access data from anywhere at any time via the Internet access. However, storing data in a third party's cloud system brings a serious concern on the data confidentiality. We consider a cloud storage system model that has no central authority. A tight integration of public key encryption schemes and random erasure codes is developed. By using this integration, we present a secure cloud storage system, which guarantees the data confidentiality and robustness and supports the secure data forwarding functionality. Hence, in our storage system, a user can not only securely store data but also forward data to other user in a confidential way.

Keywords: Randomized erasure codes, homomorphic encryption schemes, networked storage systems, cloud storage systems, network coding.

摘要

隨著高速網路與行動通訊的普及，雲端儲存服務已融入常日生活中，例如網路信箱，網路相簿等。使用者可以隨時隨地遠端透過行動裝置存取資料。除了可信賴的儲存機制之外，雲端儲存系統中的資料隱私性問題已日益被重視。將資料儲存在雲端系統中意味著將資料放置在第三者的環境中。如何同時保障使用者資料隱私性與儲存系統功能性是我們研究的主題。我們考慮一個沒有中央控制單位的雲端儲存系統，結合了公開金鑰加密系統與容錯編碼技術來設計一個同時具有高度隱私性與容錯能力的雲端儲存系統，除了基本的存取功能之外，我們的系統更提供了一個安全的資料轉移機制，使用者可以將自己的資料授權給其他使用者使用。我們的儲存系統保障了使用者資料的隱私性，即使是所有的儲存伺服器都被攻擊者控制，也無法破壞。系統同時具有容錯能力，當儲存系統中的儲存伺服器無預警離線或關閉，系統服務仍能正常運作。爲了非集中式的系統架構，我們的公開金鑰加密系統經過特殊設計，使得編碼的程序與解密的程序可以平行地在各伺服器中運作，無須中央控制單位的協助。整體儲存系統除了基本的容錯能例外，使用者可以享有高度的資料隱私安全。

Dedicated to my parents

獻給我的父母

Acknowledgements

I still find it hard to believe that this is the end of my graduate school days.

There are several people without whom this work would not have been possible, first and foremost is my advisor, Prof. Wen-Guey Tzeng. With his endless encouragement and patience, he has been a terrific advisor. Many years of collaboration with him have come to fruition in this dissertation. My appreciation also goes to all the other members of my committees for their valuable feedback and time. Prof. John Kubiatowicz and Prof. Doug Tygar gave me large amount of their time and served as my advisors for my third year when I were at UC Berkeley.

Despite being neither my advisor nor a member of my thesis committees, Dr. Cheng-Kang Chu, one of my senior officemates, has taught me many things, right from the tricks of security proofs to administrative services of our laboratory. Most importantly, he always had the time for me whether I was asking him even if he has already graduated and been a researcher in Singapore. These acknowledgements would be incomplete without thanking all of my officemates for all the wonderful time we had together. Especially, Shiuan-Tzuo and Yi-Ruei always helped me out from the huge amount of administrative things.

I thank the department administrative staff not only for that they took care of the university formalities for me but also that they encouraged and supported me whenever I needed. I am going to miss the cookies or cakes they gave me when I occasionally walked in the department office.

I cannot thank my parents and my husband enough in words for their unconditional love and support at all time. I would also like to thank my sisters for their constant support during my years at NCTU.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A cloud providing on-demand services on the Internet has a collection of servers. Servers in a cloud connect to each other via networks. The contributed resources, such as storage and computing power, from servers are implicitly merged as one huge resource and shared among users. A cloud storage system is an accumulation of storage servers that provide a location independent storage service as long as a user has network access. A user can store data in and retrieve data from a cloud storage system from anywhere at any time. Companies can also outsource their data storage and management to a cloud storage system. Cloud storage systems not only provide the storage service as a purely virtual file system but also contribute to many service-based applications. For instance, web-mail systems [1] allow people to read and to write emails through the web browser without storing those emails in local machines [2].

The advantage of using cloud storage systems instead of the local storage devices is substantial. A user can ubiquitously enjoy the cloud storage service

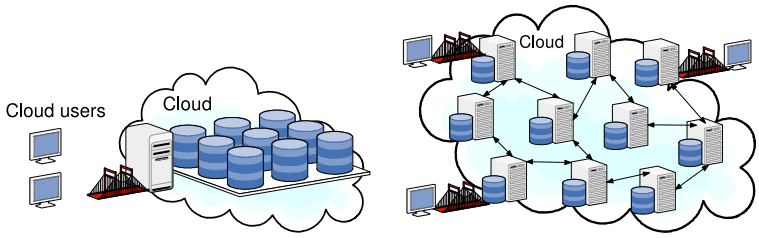


Figure 1.1: The centralized architecture and the decentralized architecture. *The centralized architecture has a central authority that manages how data are stored and retrieved. In the decentralized architecture, each storage server independently manages the data storage.*

without the heavy burden of hardware or software maintenance. In additional to low maintenance cost and ubiquitous use, cloud storage systems guarantee the data availability for a long period of time. Not only users but also the service providers gain advantage from the cloud storage systems. The service providers can deliver quality of service to users non-interactively in real-time in the cloud storage systems.

Cloud storage systems are not new technologies. A networked storage system is a simple cloud storage system and this technology can be traced back to the networked attached storage (NAS) [3] and the Network File System (NFS) [4]. The extra storage devices are deployed in the network and a user can access the devices via network connection. Afterward, the extra storage devices are replaced by the storage servers because the cost of hardware becomes much economical. Moreover, the decentralized architecture of the storage servers are proposed for better scalability because any storage server can join or leave without the control of a central authority. Figure 1.1 shows the centralized architecture and the decentralized architecture of the storage servers.

Contributed by the rapid and ubiquitous network access technology, cloud storage systems have become a reality. For example, Amazon Simple Storage Service (S3) provides the storage service in which users store and retrieve data via a web interface, and IBM also offers the storage cloud for enterprises. However, storing data in the cloud brings up many security issues. According to a research report from International Data Corporation (IDC) Enterprise Panel [5], the security issue is the top challenge for the cloud services. The second and third issues are performance and availability. Figure 1.2 shows the IDC analysis about issues of the cloud. Without the security guarantee, few people will use the system and users face potential risks of privacy leaking and confidentiality breaking.

In the history of networked storage systems, which are the ancestors of cloud storage systems, some of them use encryption schemes to provide the data confidentiality. Some of storage systems [6, 7, 8, 9] using symmetric encryption schemes to encrypt data on disk. In those storage systems, the storage servers own the keys. Thus, the data confidentiality is not against the storage servers. Some of the storage systems, such as Farsite [10], uses asymmetric encryption schemes to encrypt the stored data. However, those systems use the replication mechanism for the data robustness. The replication mechanism causes a high storage overhead.

Motivated by the strong need for security mechanisms for cloud storage systems, this dissertation focuses on the data confidentiality and robustness of cloud storage systems with the decentralized architecture. My dissertation statement can be summarized as follows.

A cryptographic secure cloud storage system built on a decentralized ar-

Rate the challenge/issues ascribed to the 'cloud/on-demand model'
1=not significant, 5=very significant

Ranking	Issue	% responding 4 or 5
1	Security	74.6
2	Performance	63.1
3	Availability	63.1
4	Hard to integrate with in-house IT	61.1
5	Not enough ability to customize	55.8
6	Worried on-demand will cost more	50.4
7	Bringing bake in-house may be difficult	50.0
8	Regulatory requirements prohibit clouds	49.2
9	Not enough major suppliers yet	44.3

Figure 1.2: IDC analysis.

IDC conducted a survey of 244 *IT executives/CIOs and their colleagues about their companies views about IT Cloud Service. The main results are summarized. The top challenge is security. IDC concluded that cloud services still need to be more secure.*

chitecture provides strong data confidentiality against collusion of all storage servers and is robust with low storage overhead. Furthermore, the system is adaptable to allow data forwarding inside the cloud in a confidential way.

To validate my dissertation statement, this dissertation presents a cloud storage system that is decentralized and provides the data confidentiality even if all storage servers are compromised. Moreover, it also provides an improved cloud storage system, which additionally supports the data forwarding function in a confidential way. An investigation is done on the relationship between the communication-efficiency parameter and the probability of a successful data retrieval event. A general setting of the parameters is provided according to the scale of the storage system.

Through the process of validating my dissertation statement, this dissertation describes two cloud storage systems and provides both a theoretic

analysis of the performance and the probability of a successful data retrieval event. The data confidentiality and robustness issues are addressed by using public key encryption schemes and random erasure codes. The core technique we developed is a tight integration between the public key encryption schemes and random erasure codes. The following novel contributions are made in this dissertation:

- A novel cloud storage system model that provides both storage service and key-management service.
- Cloud storage systems that allow each storage server independently encoding data when the data are in encrypted form.
- A cloud storage system that allows data being forwarded in an encrypted form after the data are encoded and distributed.
- A general setting for the communication-efficiency parameters is suggested according to the scale of the storage system.

Roadmap. Chapter 2 presents some background on networked storage systems and a discussion of the security needs of networked storage systems to illustrate our motivation for the secure cloud storage systems. Chapter 3 presents some algebraic setting and notation, our first cloud storage system model, the advanced cloud storage system model, and the threat model for the data confidentiality. Chapter 4 presents the construction of the first secure cloud storage system. It also contains the full analysis of correctness, performance, security, and the probability of a successful data retrieval event. Chapter 5 presents the advanced cloud storage system, which supports

the data forwarding function. Chapter 6 explores a variety of mechanism for security issues about data integrity checking in cloud storage systems and discusses how our systems can potentially integrate with some of them. Chapter 7 concludes and presents some future directions for further research.

Chapter 2

Review of Networked Storage Systems

We introduce the problem domain of this dissertation in this chapter. There has been much research on networked storage systems in the last decade of years. Networked storage systems are classified as centralized and decentralized ones depending on the existence of a central authority. Because the purpose of networked storage systems is to store data reliably over long periods of time, this chapter further classifies them by what technique they employed for data robustness against machine failures. This chapter describes some well-known networked storage systems in literature, although this brief survey is by no means complete or exhaustive. In particular, we do not cover the papers related to altered data detection and correction in networked storage systems. After introducing the basic research area, we discuss the data confidentiality problem in the context of networked storage systems.

Robust storage technologies. Ensuring data available against machine failures requires the introduction of redundancy. Replication is the simplest approach for achieving resilience to arbitrary corruption of storage. Unfortunately, this method has a high overhead in storing full copy of the file at each storage server. Erasure codes are applied in many networked storage systems for tolerating failure of storage servers with lower storage overhead. Networked storage systems apply each of the technologies or a mixed of them to provide a robust data storage in either centralized or decentralized architecture.

2.1 Centralized Data Management

A centralized networked storage system has a central authority which may be a single server or a small cluster of central servers. The central authority has a global view of the whole storage system, such as the network topology and the global routing table, and controls how data are stored among many storage servers.

2.1.1 Mirrors and Replicas

At the early years, the network-attached storage (NAS) [3] deploys extra storage devices in the network and a user can access the devices via network connection. The Network File System (NFS) [4] is then proposed for the application of a file system. A user can backup his data in the NAS devices. This mechanism is analog to the one in the traditional file system with multiple directly-attached storage disks, such as the redundant array of

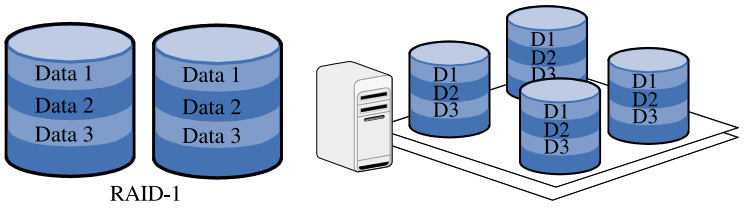


Figure 2.1: Replication on storage devices.
*The left figure is RAID-*1 *and the right figure is a small SAN with* 4 *storage devices. In RAID-*1*, each drive stores one replica of the data. Similarly, each storage device in the SAN stores a copy of the data. However, the storage devices in SAN are consolidated together via a high-speed network.*

independent disks(RAID)-1. The exception is that replicas are transmitted via a network connection instead of a physical connection. In contrast to NAS which is a file-wise storage system, Storage Attached Network (SAN) is a block-wise storage system. A file system can be built upon SAN. Mirroring is the simplest robust technique offered by SAN. Figure 2.1 illustrates the RAID-1 and a small SAN.

Many file systems built upon the networked storage systems use replication and store replicas on multiple storage servers. For example, AFS [11], Coda [12], SFS [13], and Plutus [14] are all distributed file systems that store replicas and have varied replication mechanisms. AFS maintains replicas in a read-only manner except one of them is writable to achieve the consistency among all replicas, while Coda allows all storage servers to receive updates and uses an expensive repair mechanism to handle conflicts.

When the scale of a networked storage system becomes bulky, the replica placement and consistency problems arise. The replica placement problem is to decide which set of storage servers to store the replicas such that the life time of data is extended. The consistency problem is to keep all repli-

cas consistent with the original. Fortunately, the central authority can well address these two problems by using the global system information.

2.1.2 Erasure Codes

The RAID-5 uses erasure codes to survive one drive failure and the RAID-6 can survive the failure of two drives. Figure 2.2 and Figure 2.3 show how data are stored in RAID-5 and RAID-6. The parity operation is performed in RAID-5 systems. One parity result is additionally stored to tolerate one erasure error. In RAID-6 systems, a parity computation and a Reed-Solomon encoding are performed over the stored data. As a result, up to two erasure errors a RAID-6 system can tolerate.

Erasure codes are codes that encode the input of k symbols as a codeword of n symbols such that as long as k out of n symbols of the codeword are available, the original k symbols can be decoded back. This code can tolerate $(n - k)$ erasure errors.

Erasure codes are applied in many networked storage systems for data robustness with lower storage overhead. The erasure codes are mainly used by the following framework for the robustness of networked storage systems against the failure of machines when a machine failure is modeled as an erasure error. Assume that there are n storage servers in a networked storage system. A user represents a file as an input of the encode algorithm, and encodes the file as a codeword. Later, each storage server stores a symbol of the codeword. As a result, as long as k out of n storage servers are available, the file can be decoded and retrieved by the user. The scalable distributed

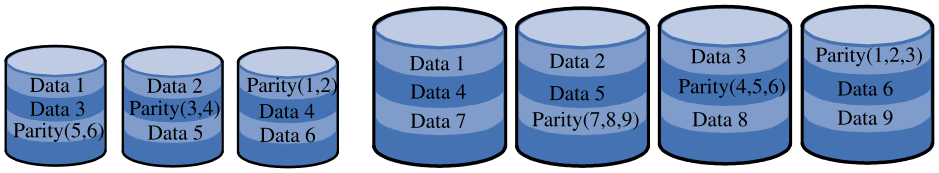


Figure 2.2: RAID-5.

The left figure is RAID-5 with 3 drives and the right figure is RAID-5 with 4 drives. The parity function is a bitwise XOR over the input. In a RAID-5 system, data are available as long as at most one drive crashes.

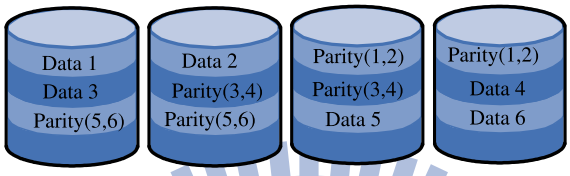


Figure 2.3: RAID-6.

The figure shows a RAID-6 with 4 drives. In a RAID-6 system, data are available as long as at most two drives crash.

storage [15] uses the Lincoln Erasure Codes, a class of erasure codes, to provide the data availability.

The central authority of a networked storage system can choose special coding method to obtain a better ability on tolerating errors or a better coding performance. We categorize the storage systems that use erasure codes by what operations the erasure codes employ.

Algebraic Operation based Erasure Codes

One of the most well-known erasure codes is the Reed-Solomon codes and the storage system in [16] uses Reed-Solomon codes to tolerate both erasure and faulty errors. A simple Reed-Solomon code is described as follows and illustrated in Figure 2.4. A message m is represented as k elements in a finite field, i.e. $m = (m_1, m_2, \ldots, m_k)$. Consider a polynomial function f with

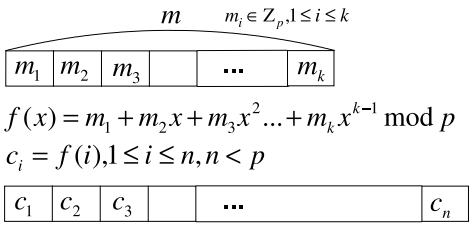


Figure 2.4: An encoding example of a Reed Solomon code.
The message defines a polynomial function and a codeword is defined by the values of the polynomial on n points.

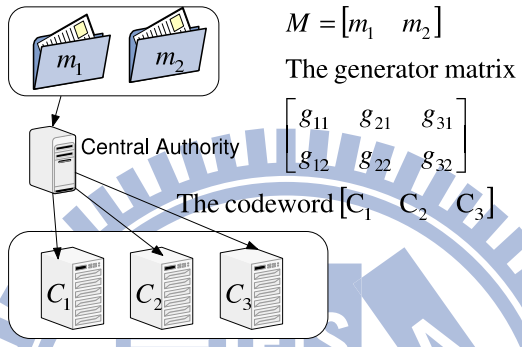


Figure 2.5: An example of a storage system that uses a linear code.
The central authority encodes the messages into a codeword (C_1, C_2, C_3) and sends a distinct symbol to a storage server for storage.

degree $k-1$ where $f(x) = m_1 + m_2 x + m_3 x^2 + \cdots + m_k x^{k-1}$. A codeword c with length n is $(f(1), f(2), \ldots, f(n))$, where the polynomial function is computed over certain finite field. As long as k elements in the codeword are available, the polynomial function f can be recovered as well as the message m.

In a storage system using a linear code to provide the data robustness, the central authority can encode messages and recover them back. Figure 2.5 shows the example where there are 2 messages and 3 storage servers and the coding is operated in a finite field. After the central authority gets the two messages, he generates a generator matrix and encodes the messages via the

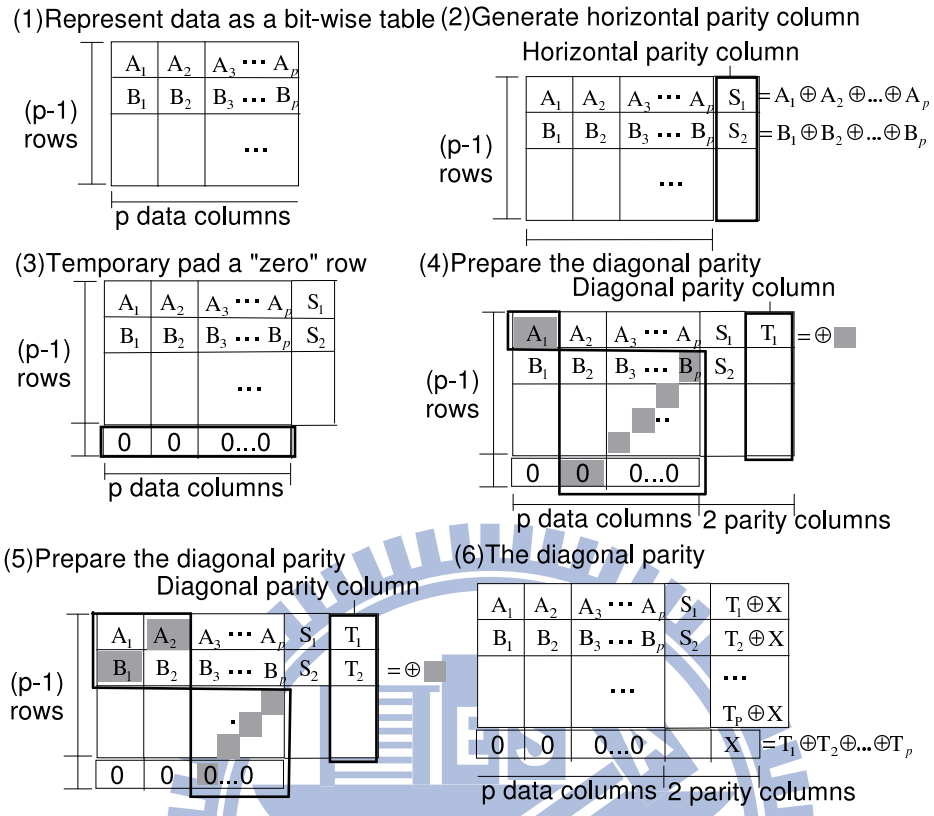


Figure 2.6: The EVENODD encoding.

The data are represented as a $(p-1)\times p$ table. An entry is a bit of the data. After the encoding, the codeword is represented as a $(p-1)\times(p+2)$ table. Each storage server stores one column of the resulting table.

generator matrix. The codeword contains 3 symbols and each storage server stores one of them. After the central authority retrieves 2 out of 3 codeword symbols, he performs the decoding process and sends the messages back to the user.

XOR Operation based Erasure Codes

Some erasure codes are proposed for their excellent performance. They only use exclusive-or operations. As a result, the storage systems also have good

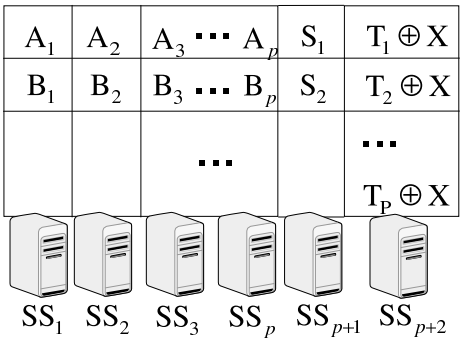


Figure 2.7: A system using the EVENODD encoding.
Each storage server stores a column of the encoded table. This storage system tolerates the failure of two servers.

performance on the data storage and retrieval processes. The EVENODD code [17, 18] and the STAR code [19] are proposed for tolerating 2 and 3 erasure errors, respectively. The encoding of the EVENODD codes and the STAR codes are efficient, but the codes can only tolerate constant number of erasure errors. Figure 2.6 illustrates the EVENODD encoding and Figure 2.7 shows how a system stores the encoded data. There are $(p+2)$ storage servers SS_1,SS_2,...,SS_{p+2} in the system and each of them stores a column data, which is p bits. The STAR codes use an additional parity column for tolerating one more erasure error. Figure 2.8 shows the STAR encoding and how a system stores the encoded data.

Many low-density parity check (LDPC) codes have also more efficient encoding and decoding algorithms than other erasure codes that use linear algebraic operations do. A networked storage system that uses a systematic LDPC code is described as follows. Figure 2.9 shows an overview of the storage system. Assume that there are n messages. The storage servers are divided into two groups R and L. The R group consists of n storage servers

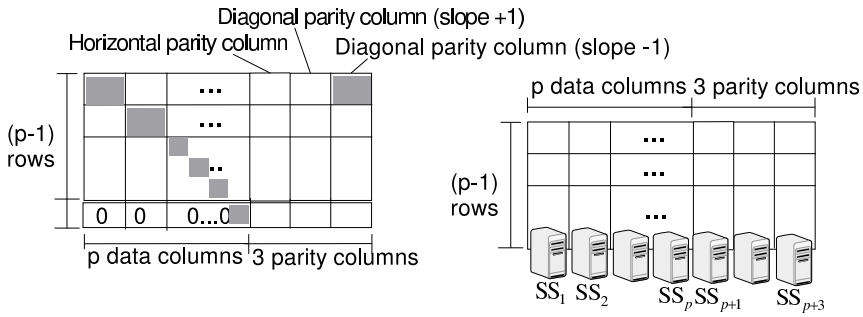


Figure 2.8: The STAR encoding and a system storing the encoded data. *STAR codes tolerate the failure of* 3 *servers. There are* 3 *columns of parity bits.*

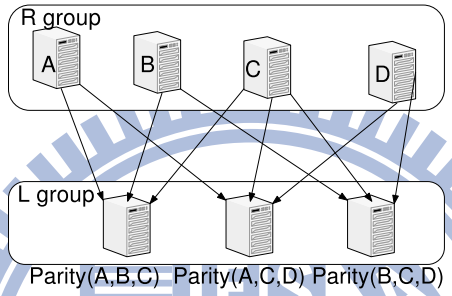


Figure 2.9: A networked storage system that uses a LDPC code. *The system contains* 7 *storage servers and* 4 *out of them store the original data. The other* 3 *storage server store the parity results of the data.*

and each of them keeps one message. The L group has the rest of the storage servers and each of them stores a parity over a subset of n messages.

Tornado codes [20, 21] are also a class of LDPC erasure codes that have fast encoding and decoding algorithms. Tornado codes illustrated in Figure 2.10 use irregular bipartite graphs as the encoding structures. The archival storage [22] uses Tornado codes as the fault-tolerance technique.

Different LDPC codes define different policies on the L group, i.e. how many and which messages are used to produce the parity. The experimental results in [23] show that how differently LDPC codes perform on the robustness ability. When the number of message symbols are below 100, the

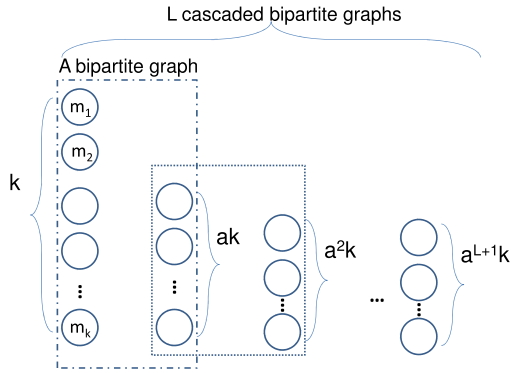


Figure 2.10: The encoding structure of Tornado codes.
Tornado codes use cascaded irregular bipartite graphs. The k input message bits are listed as k left vertices of the first bipartite graph. The right vertices of the first bipartite graph are ak parity bits. A similar encoding process is performed on the rest $L-1$ cascaded bipartite graphs. The decoding algorithm is simple. For each right node whose all but one neighbors are known, the missing neighbor can be recovered. The decoding algorithm successfully terminates when all k input message bits are recovered.

systematic LDPC codes perform better (fewer codeword symbols are required for a successful decoding). However, when the number of message symbols equals or greater than 100, irregular repeat-accumulate LDPC codes perform better.

2.2 Decentralized Data Management

Because no central authority can arrange how data are distributed among and retrieved from the whole storage system, a storage server independently manages the stored data without global information of the networked storage system. Although the data management without a central authority is much more complicated, the decentralized architecture has many advantages in practice. A decentralized architecture for storage systems offers a good

scalability since any storage server can join or leave without the control of a central authority. The architecture can bear a global-size system scale and the resulting massive storage space. Peer-to-peer networks are major concrete examples for the decentralized architecture.

In this section, We describes several decentralized networked storage servers that use different robust storage technologies.

2.2.1 Mirrors and Replicas

When there is no central authority, a datum can be flooded into the storage system and each (available) storage server stores a copy of it. PAST [24] leaves the choice of a replication factor, i.e. the number of replicas, to the data owner. Farsite [10] is a reliable file system that uses random replication to support long-term data persistence. Glacier [25] is a distributed storage system that uses massive replication to provide data robustness across large-scale correlated failures. Many strategies aim to arrange where a replica is stored such that this replica can be found when needed. Carbonite [26] is a replication algorithm for keeping data durable at a low cost. It contributes to distributed storage prototypes such as Antiquity [27], OverCite [28], and UsenetDHT [29].

Consider a peer-to-peer network as a special case of a networked storage system. The replica placement and the quality of availability are a huge research area in peer-to-peer networks. For example, a repair mechanism handles how replicas should be re-distributed when a peer leaves the network.

In addition to the static control over the number of available replicas,

proactive replication [30, 31] provides a better guarantee for data durability. The replication algorithm in [30] produces replicas periodically. The replication algorithm in [31] generates replicas by predicting the machine availability.

2.2.2 Erasure Codes

Similar as the case in the centralized architecture, applying an erasure code provides a lower storage overhead while the resulting system retains the data robustness. Technologies such as fountain codes, decentralized erasure codes, or random linear codes are proposed for the decentralized storage systems [32, 33, 34, 35, 36, 37, 38].

XOR Operation based Erasure Codes

Fountain codes enable a robust storage technique. Luby transform (LT) codes and Raptor codes [39, 40] are two classes of fountain codes. In a fountain code, the message symbols and codeword symbols can be modeled as vertices in a random bipartite graph. The degree of each vertex is random over some probabilistic distribution. Different distributions define different codes. Figure 2.11 illustrates an overview of the fountain codes. In the LT codes, each codeword symbol is the result of exclusive-or over d message symbols, where the value d is random according to an Ideal Soliton distribution and a robust Soliton distribution. The Raptor code is a class of the fountain code with linearly encoding and decoding complexity. It uses a modification of the Ideal Soliton distribution for the degree d. The distributed storage systems

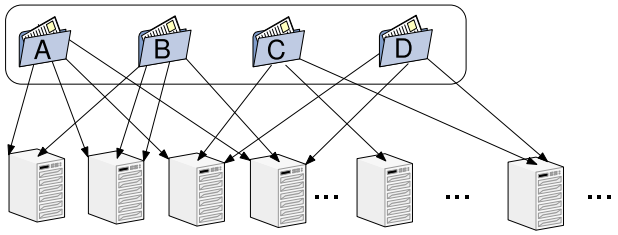


Figure 2.11: An abstract overview of a fountain code.
The data A, B, C and D are randomly distributed according to some probabilistic distribution. Different codes define different distributions.

using LT codes [35] and the system using Raptor codes [37, 38] are proposed in wireless sensor networks.

Algebraic Operation based Erasure Codes

A random linear code is a linear code with a random generator matrix. Each storage server can randomly determine a column of the generator matrix. The result in [34] shows that a sub-square matrix of the generator matrix is invertible with an overwhelming probability. That is, the probability that a user can successfully retrieve his data with an overwhelming probability. Figure 2.12 shows an example of a small storage system that uses a random linear code. There are 6 storage servers SS_1, SS_2, ..., and SS_6 in the system. After receiving some message symbols, a storage server randomly picks coefficients and linearly combines the received message symbols. The storage server SS_1 stores the codeword symbol C_1 and the coefficients x_{1a}, x_{1b}, x_{1c} and x_{1d}, where $x_{1d} = 0$.

The number of codeword symbols that a datum is contributed could influence the probability of a successful retrieval. Therefore, a decentralized erasure code [34] is proposed for certain settings. The system is summarized

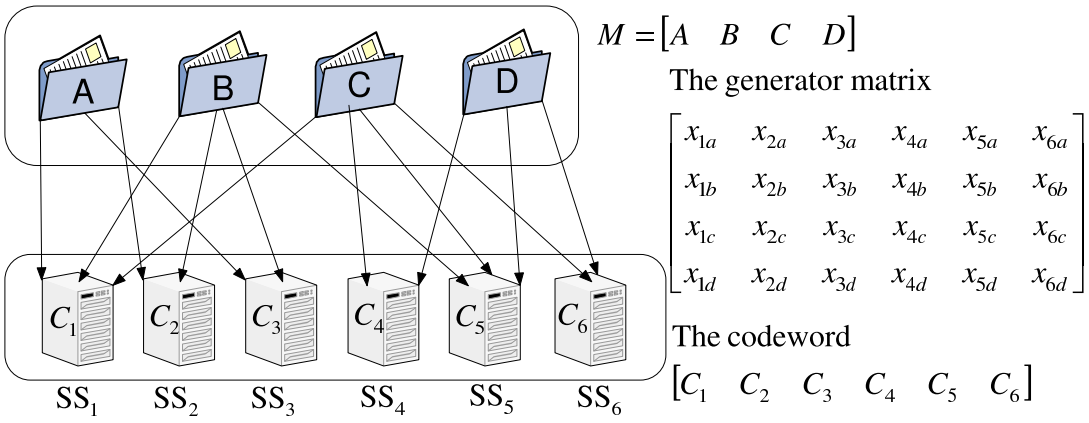


Figure 2.12: A networked storage system using a random linear code. *Each storage server selects a random coefficient for each received message. The linear combination is performed by the storage server. Each column of the generator matrix and each codeword symbol is determined and computed by a storage server. No central authority is required in the encoding process.*

in Figure 2.13. In this approach, to store k data, a user makes v copies for each datum and distributes them to randomly selected storage servers in the storage system. After receiving those data, each storage server encodes those data and stores the result. To retrieve k data, the user randomly queries k storage servers for the encoding results and tries to decode those k data. It has been shown that for n storage servers and k messages, if $n = ak$ and $v = bk \ln k$ where $b > 5a$, the probability of successful retrieval is at least $1 - k/p - o(1)$, where p is the size of the used group [34].

2.2.3 Hybrid Strategy

The hybrid strategy is a mixed method of the replication and an erasure code. It trades off the advantages and disadvantages between those two approaches. Many well-known and newly decentralized networked storage systems, such as OceanStore [41] and Total Recall [42], take the hybrid strategy. The

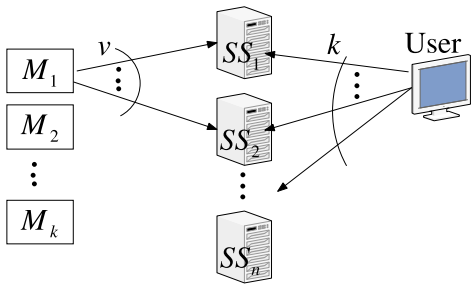


Figure 2.13: A networked storage system using a decentralized erasure code. *The settings for the parameters n, k, and v and the random process of data distribution determine the probability of a successful data retrieval.*

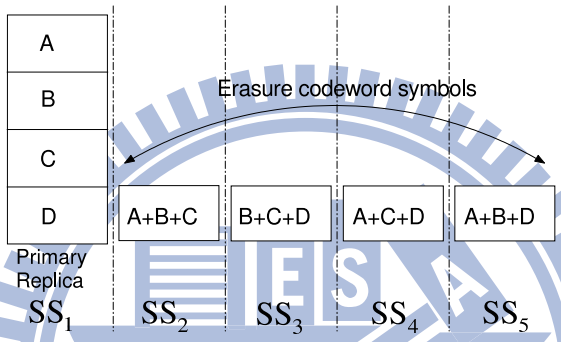


Figure 2.14: A networked storage system that uses a hybrid strategy. *The storage system simultaneously stores the data and the resulting symbols of the erasure coding. The number of the replicas of the data and the number of symbols of the rasure code influence the data availability of the storage system.*

storage servers are divided into primary storage servers that keep the replicas and the secondary storage servers that store the encoding results.

The ratio of two kinds of storage servers may influence the data availability. Some research [43, 44, 45] are conducted to find an approximated-optimal ratio for providing a practically long-term data availability. Figure 2.14 gives a layout for an erasure coding with primary copy. The experimental results [45] are produced by running experiments on the real traces of large-scale distributed storage systems (e.g. Farsite [10] and Overnet [46]).

2.3 Challenge of Data Confidentiality

After the storage servers are located on the network, the control of the storage servers is no longer at the data owner's hand. Therefore, there is an increasing attention on the data confidentiality.

2.3.1 Cleartext Storage

Early networked storage systems are proposed for robust storage with simple access control mechanisms. The data are stored in cleartext. When the storage system uses an erasure code for data robustness and each storage server stores a codeword symbol, the system has certain data confidentiality because less than k storage servers cannot recover the original message, where the message has k symbols.

2.3.2 Symmetric Encryption

After the hardware and software of the storage servers are improved, the storage servers can handle more computation. Many new networked storage systems provide stronger data confidentiality by storing data in encrypted form. Once a storage system receives data from the owner, the data are encrypted by symmetric encryption scheme such as DES or AES before stored into the physical drives. Blaze's CFS [6], TCFS [8], StegFS [7] and NCryptfs [9] are file systems that encrypt data before writing them to storage drives. Those file systems only protect the stored data at rest and assume that the storage servers are fully trusted.

Other large-scale networked storage systems, such as OceanStore [41],

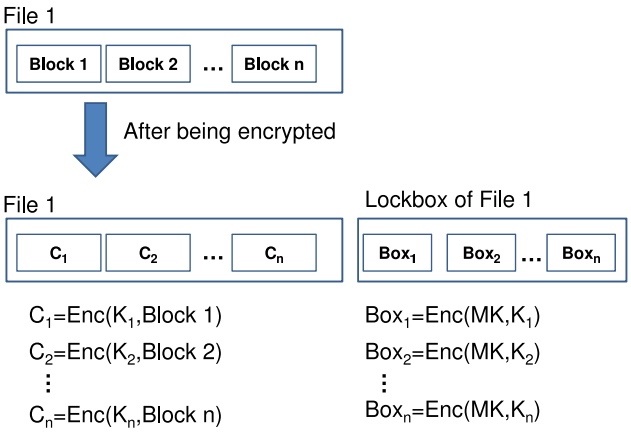


Figure 2.15: File 1 before and after being encrypted in a Plutus system. *The file is divided into blocks and each file block is encrypted by using a distinct symmetric key. All symmetric keys are encrypted by using another symmetric key MK. The encrypted symmetric keys are stored with the encrypted file blocks. A user may use a different MK for each file.*

Plutus [14] and Tahoe [36], use encryption schemes to protect the data confidentiality against both internal and external attackers. In OceanStore, all information that enters the system must be encrypted while the owner manages the access control. For example, when the owner wants to share a datum with others, he needs to distribute the symmetric key to the authorized readers. Similarly, in both Plutus and Tahoe, a user needs to encrypt files with distinct symmetric keys and manage all of the keys by himself. A file before and after being encrypted in Plutus is illustrated in Figure 2.15. A newly encryption service [47] is provided for any cloud storage user who uses Amazon Simple Storage Service. The encryption service encrypts the user's data by using AES and stores the ciphertext into the cloud storage system for the user. Again, this application assumes that the servers who encrypt the data are fully trusted.

However, the data confidentiality of above systems is guaranteed either

when the storage servers are honest and secure or when users take the responsibility of key management over the huge amount of symmetric keys. For most cases, those storage servers are assumed that they will follow the user-defined access policy on the stored data and keep the stored data in the encrypted form all the time. The trust on all storage servers sometimes is unrealistic especially when the storage system is decentralized and distributed over a large geographic area. Any one of the storage servers could be vulnerable from internal or external attacks. On the other hand, the burden of key management for users should be decreased or moved to the servers. Hence, stronger data confidentiality with low overhead on users is required. The data should be kept secret even if all storage servers are compromised, and users store as few as possible keys and put as less as possible effort on the key management.

2.3.3 Public Key Encryption

Applying a public key encryption scheme in a centralized networked storage system gives a straightforward solution to the data confidentiality issue. A user encrypts the data and then stores into the system. The central authority simply treats the ciphertext as a RAW data just like in the non-encrypted case. Similarly, the strong data confidentiality is also achievable in a decentralized system with replication technology. For instance, Farsite [10] uses the hybrid encryption to protect the data confidentiality and provide an access control mechanism. In Farsite, a datum is first encrypted by using a symmetric encryption and the symmetric key is encrypted by using the

owner's public key. The user only needs to store his secret key for all of his data. When he wants to share some data with some user, he encrypts the corresponding symmetric keys by using the authorized user's public key and stores the ciphertext in the storage system. The overhead of the key management is mainly moved to the servers because most of the keys are stored in storage servers (in an encrypted form) except for the user's secret key.

2.3.4 Motivation

To my best knowledge, few research addresses the data confidentiality against the collusion of all storage servers in a decentralized networked storage system that uses erasure codes. Here is the place where my results fill in. We provide a secure cloud storage system that provides a strong data confidentiality in a decentralized environment and a good data availability by using erasure codes. Our key technique is combining a public key encryption scheme and a variant of random linear code. As a result, the data are stored in an encrypted and encoded form in each storage server and no storage server has the decryption key. The access right management is totally controlled by the data owner. The data confidentiality is fully guaranteed even if all storage servers are corrupted at the same time.

Chapter 3

Erasure Codes and System Models

We introduce our basic algebraic notations and the erasure codes we used in this chapter. The special erasure code is one of our key techniques to achieve both robustness and parallelism in our cloud storage system. We consider that there is no central authority in the collection of storage servers and introduce our first system model and an advanced one. We also describe the threat model to measure the security degree of the cloud storage systems.

3.1 Bilinear Map and Assumptions

Bilinear map. Let $\mathbb{G}_1, \mathbb{G}_2$ be cyclic multiplicative groups[1] with prime order p and $g \in \mathbb{G}_1$ be a generator. A polynomial-time computable map $\tilde{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ is a bilinear map if it has the bilinearity and non-

[1]It can also be described as additive groups over points on an elliptic curve.

degeneracy: for any $x, y \in \mathbb{Z}_p, \tilde{e}(g^x, g^y) = \tilde{e}(g, g)^{xy}$ and $\tilde{e}(g, g)$ is not the identity element in $\mathbb{G}_2$. In fact, $\tilde{e}(g, g)$ is a generator of $\mathbb{G}_2$. Let $\mathsf{Gen}(1^\lambda)$ be an algorithm generating $(p, \mathbb{G}_1, \mathbb{G}_2, \tilde{e}, g)$, where λ is the length of p.

Let $x \in_R X$ denote that x is randomly chosen from the set X.

Bilinear Diffie-Hellman assumption. Following the above parameters, given g, g^x, g^y, g^z, where x, y, and z are randomly chosen from $\mathbb{Z}_p$, the bilinear Diffie-Hellman problem is to find $\tilde{e}(g, g)^{xyz}$. The assumption is that it is hard to solve the problem with a significant probability in polynomial time. Formally, for any probabilistic polynomial time algorithm $\mathcal{A}$, the following probability is negligible (in λ):

$$\Pr[\mathcal{A}(g, g^x, g^y, g^z) = \tilde{e}(g, g)^{xyz} : x, y, z \in_R \mathbb{Z}_p]$$

Decisional Bilinear Diffie-Hellman assumption. This assumption is that given g, g^x, g^y, g^z, it is hard to distinguish $\tilde{e}(g, g)^{xyz}$ from a random element from $\mathbb{G}_2$. Formally, for any any probabilistic polynomial time algorithm $\mathcal{A}$, the following is negligible (in λ):

$$|\Pr[\mathcal{A}(g, g^x, g^y, g^z, \mathbb{Q}_b) = b : x, y, z, r \in_R \mathbb{Z}_p;$$
$$\mathbb{Q}_0 = \tilde{e}(g, g)^{xyz}; \mathbb{Q}_1 = \tilde{e}(g, g)^r; b \in_R \{0, 1\}] - 1/2|$$

3.2 Erasure Codes over Exponents

The erasure codes we used can be seen as a variant of the traditional random linear codes. We briefly review the random linear codes and present the

erasure codes over exponents in this section.

3.2.1 Random Linear Codes

Let the message be $\vec{I} = (m_1, m_2, \ldots, m_k)$, the generator matrix $G = [g_{i,j}]_{1 \le i \le k, 1 \le j \le n}$ and the codeword be $\vec{O} = (w_1, w_2, \ldots, w_n)$. The elements of $\vec{I}$ and $\vec{O}$ and entries of G are all over a finite field $\mathbb{F}$ of size p. The generator matrix of a random linear code has random entries from the finite field. As a result, each element of $\vec{O}$ is a linear combination of $\vec{I}$ where the coefficients are randomly chosen from $\mathbb{F}$.

A decentralized erasure code [34] is a random linear code with a sparse generator matrix. The generator matrix G of a decentralized erasure code constructed by an encoder is as follows. First, for each row, the encoder randomly marks an entry as 1 and repeats this process for $an \ln k/k$ times with replacement (an entry can be marked multiple times), where a is a constant. Second, the encoder randomly sets a value from $\mathbb{F}$ for each marked entry. The encoding process is expressed as $\vec{I} \cdot G = \vec{O}$. As for the decoding, a decoder receives k columns $j_1, j_2, \ldots, j_k$ of G and the corresponding codeword elements $w_{j_1}, w_{j_2}, \ldots, w_{j_k}$. The decoding process is computed as follows:

$$[m_1, m_2, \ldots, m_k] = [w_{j_1}, w_{j_2}, \ldots, w_{j_k}] \begin{bmatrix} g_{1,j_1} & g_{1,j_2} & \cdots & g_{1,j_k} \\ g_{2,j_1} & g_{2,j_2} & \cdots & g_{2,j_k} \\ \cdots & \cdots & \cdots & \cdots \\ g_{k,j_1} & g_{k,j_2} & \cdots & g_{k,j_k} \end{bmatrix}^{-1}$$

A decoding is successful if and only if the $k \times k$ submatrix formed by the

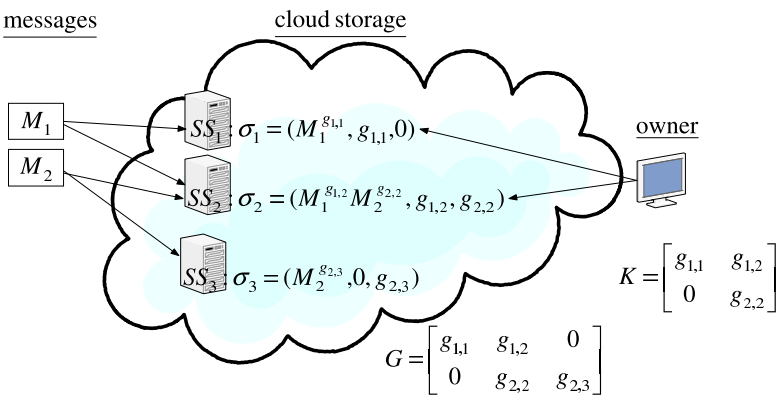


Figure 3.1: A storage system using the random erasure code.
Messages M_1 and M_2 are randomly distributed to storage servers SS_1, SS_2, and SS_3. The storage server SS_1 randomly selects a coefficient $g_{1,1}$ for the received message. Similarly, the storage servers SS_2 and SS_3 individually select coefficients.

k chosen columns is invertible. Thus, the probability of a successful decoding is the probability of the chosen submatrix being invertible. It has been shown in [34] that the probability is at least $1 - k/p - o(1)$, where the randomness is introduced by the random choices for marked entries, the random values for marked entries, and the random choices for k columns.

3.2.2 Random Erasure Codes over Exponents

We fix a cyclic multiplicative group $\mathbb{G}$ with prime order p. The message domain is $\mathbb{G}$. The generation of the generator matrix G is the same as the above decentralized erasure code except that the entries of G are over $\mathbb{Z}_p$. The encoding process is to generate $w_1, w_2, \ldots, w_n \in \mathbb{G}$, where $w_i = m_1^{g_{1,i}} m_2^{g_{2,i}} \cdots m_k^{g_{k,i}}$. An example is shown in Figure 3.1. There are 2 messages stored into 3 storage servers. The first step of the decoding process is to compute the inverse of a $k \times k$ submatrix K of G. Let $K^{-1} = [d_{i,j}]_{1 \le i,j \le k}$. The

second step of the decoding process is to compute $m_i = w_{j_1}^{d_{1,i}} w_{j_2}^{d_{2,i}} \cdots w_{j_k}^{d_{k,i}}$, where $j_1, j_2, \ldots, j_k$ are the indices of columns of K in G. Therefore, a sufficient condition for a success decoding of the variant decentralized erasure code is that the $k \times k$ submatrix K is invertible. Similar to the decentralized erasure code, the probability of a success decoding is at least $1 - k/p - o(1)$.

Since the decoder only requires k columns of G and their corresponding codeword elements to decode, this code is resilient to $(n - k)$ erasure errors. Moreover, the code is decentralized because each codeword element w_i can be independently generated. A distributed networked storage system having n servers uses a random erasure code as follows. The owner wants to store k messages M_i, $1 \leq i \leq k$. For each M_i, the owner randomly selects v servers with replacement and sends a copy of M_i to each of them. Each server randomly selects a coefficient for each received message and performs a linear combination of all received messages. Those coefficients chosen by a server form a column of the matrix and the result of the linear combination is a codeword element. Because there are n servers, a $k \times n$ generator matrix and a codeword are implicitly formed. Each server can perform the computation independently. This makes the code decentralized.

3.3 System Models

We first consider a basic storage system model which is capable for the fundamental functions, i.e. storing and retrieval. Later, we extend the system by adding the data forwarding function. The advanced system model supports the data forwarding function such that the both the owner forwards and the

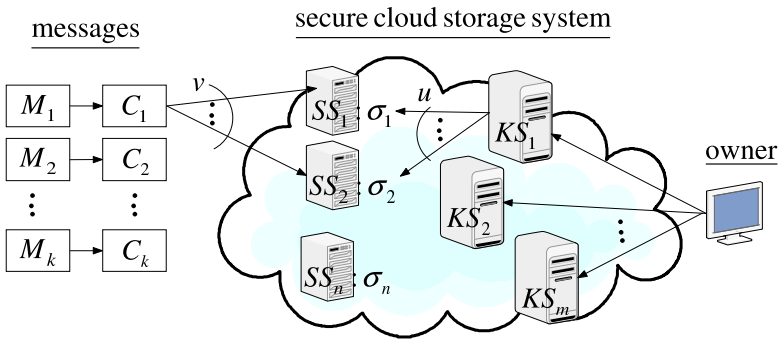


Figure 3.2: Our first system model of our secure cloud storage system. *Messages are encrypted and then randomly distributed amon the storage servers. Each storage server performs the combination on received ciphertexts and only stores the result and chosen coefficients.*

granted user retrieves data in a confidential way.

3.3.1 The First System Model

Figure 3.2 provides an overview of our first system model. There are n storage servers SS_1, SS_2, ..., SS_n and m key servers KS_1, KS_2, ..., KS_m. The storage servers provide storage services and the key servers provide key management services. The system consists of 3 phases: system setup, data storage, and data retrieval. They are described as follows.

In the *system setup* phase, the system chooses and computes public parameters. A user A has his own storage space, his public key PK_A and secret key SK_A. The user A publishes his public key and shares his secret key to a set of key servers by his own choice with a threshold value t. As a result, each chosen key server KS_i, $1 \leq i \leq m$, holds a key share $\mathsf{SK}_{\mathsf{A},i}$ of the user's secret key SK_A.

In the *data storage* phase, a user A wants to store k messages M_i, $1 \leq$

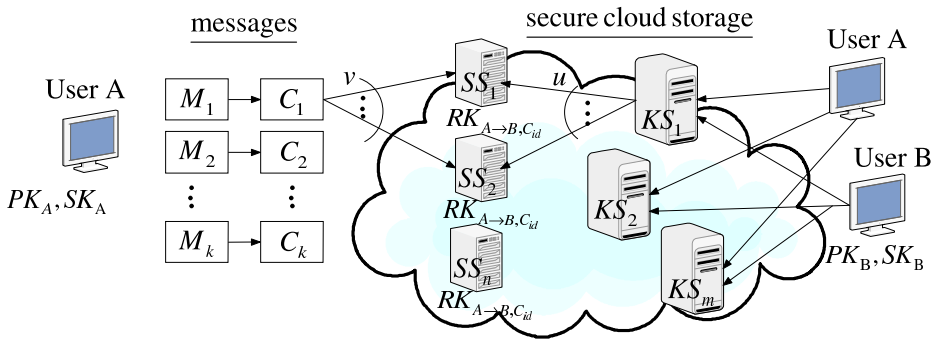


Figure 3.3: The advanced system model of our secure cloud storage system. *User* A *generates a re-encryption key* $\mathsf{RK}_{\mathsf{A}\to\mathsf{B},\mathsf{F_{id}}}$ *and distributes it to all storage servers such that the storage servers re-encrypt the ciphertexts into ones under user* B*'s key.*

$i \leq k$, into n storage servers SS_i, $1 \leq i \leq n$. We could think that these messages are the segments of a file. For those k messages, A assigns a message identifier. Each message M_i is encrypted under the public key $\mathsf{PK_A}$ as $C_i = E(\mathsf{PK_A}, M_i)$. Then, each ciphertext is sent to v storage servers, where the storage servers are randomly chosen. Each storage server SS_i combines the received ciphertexts by using the erasure code to form the stored data σ_i.

In the *data retrieval* phase, to retrieve the k messages, A instructs the m key servers such that each key server retrieves stored data from u storage servers and does partial decryption for the retrieved data. Then, A collects the partial decryption results, called decryption shares, from the key servers and combines them to recover the k messages.

3.3.2 Advanced System Model

Our advanced system model is illustrated in Figure 3.3. Again, the system model consists of users, n storage servers SS_1, SS_2, ..., SS_n and m key servers KS_1, KS_2, ..., KS_m. In addition to the basic functions, the advanced system

model supports one more important function – data forwarding. Hence, the storage system consists of 4 phases: system setup, data storage, data forwarding, and data retrieval. The only different phases from the first system model are the data forwarding and the data retrieval. Thus the two phases are described as follows.

In the *data forwarding* phase, user A can forward the data D to another user B. User A computes a re-encryption key from A to B respect to the data D and sends it to all storage servers. After getting the re-encryption key, each storage server re-encrypts the data D of user A. The re-encryption operation transfers the ciphertext of D to a ciphertext for B. As a result, the re-encrypted data can be decrypted by using B's secret key. We say that the originally encrypted data as level-0 ciphertexts and the re-encrypted data as level-1 ciphertexts.

In the *data retrieval* phase, user A retrieves the data from the system. The data either belong to the user A or are forwarded to him. First, the user sends a retrieval request to all key servers. Upon receiving the user's request of retrieval, a key server queries a set of u storage servers. The queried storage servers will send the requested messages (in encrypted and encoded form) and the coefficients back to the key server. After receiving messages from the queried storage servers, the key server performs the partial decryption by using the key share and forwards the results to user A. As long as at least t key servers reply to user A's request, user A can retrieve messages with an overwhelming probability.

A small example of a commercial company is illustrated in Figure 3.4. A manager A stores his data in the storage system and classifies the data by

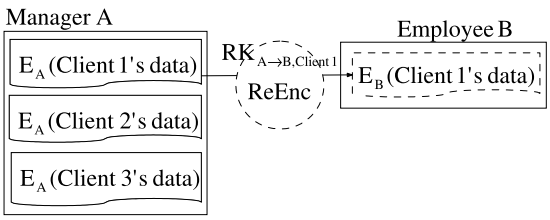


Figure 3.4: A storage system using the secure decentralized erasure code.

different clients to whom data are associated. One day, the manager wants to assign the case of client 1 to his employee B. The manager securely forwards the data associated with client 1 to B. Afterward, only B can access those forwarded data.

3.4 Threat Model

In this system model, we consider that an attacker wants to corrupt the data confidentiality of a target user and he colludes with all storage servers and up to $(t-1)$ key servers. We assume that the attacker will not tamper the stored data but he will try to get the data content from the stored data. We model this attack by the standard chosen plaintext attack of the underlying encryption scheme in a threshold version.

3.4.1 Model without Forwarding

For our first system model, we extend the standard chosen plaintext attack (CPA) security game for the threshold public key encryption scheme. The threshold CPA security game consists of a challenger $\mathcal{C}$ and an attacker $\mathcal{A}$.

- *Setup:* $\mathcal{C}$ does the following:

- Run $\mathsf{Setup}(\lambda)$ to get $\mu = (p, \mathbb{G}_1, \mathbb{G}_2, \tilde{e}, g)$.
- Run $\mathsf{KeyGen}(\mu)$ to get a key pair $(\mathsf{PK}, \mathsf{SK})$ and run $\mathsf{ShareKeyGen}$ on (SK, t, n) to get $\mathsf{SK}_i, 1 \leq i \leq n$, where t and n are randomly chosen.
- Send (μ, PK, t, n) to $\mathcal{A}$.

- *Key share query:* $\mathcal{A}$ queries $(t-1)$ secret key shares from $\mathcal{C}$ and gets $\mathsf{SK}_{q_1}, \mathsf{SK}_{q_2}, \ldots, \mathsf{SK}_{q_{t-1}}$, where $q_1, q_2, \ldots, q_{t-1} \in [1, n]$.
- *Challenge:* $\mathcal{A}$ chooses two messages M_0 and M_1, where $M_0 \neq M_1$, and sends them to $\mathcal{C}$. $\mathcal{C}$ encrypts M_b as C, where b is randomly selected from $\{0, 1\}$, and sends C to $\mathcal{A}$.
- *Output:* $\mathcal{A}$ outputs a bit b' for guessing b.

The advantage of $\mathcal{A}$ is defined as $Adv_{\mathcal{A}} = |\Pr[b' = b] - 1/2|$. A threshold public key encryption scheme is CPA secure if and only if for any probabilistic polynomial time algorithm $\mathcal{A}$, $Adv_{\mathcal{A}}$ is a negligible function in λ. A cloud storage system in the basic system model is secure if the used threshold public key encryption scheme is secure.

3.4.2 Model with Forwarding

For the advanced system model, the security game is a little bit different from the previous one for the first system model. We consider that an attacker wants to corrupt the data confidentiality of a target user with respect to an identifier and he colludes with all storage servers and up to $(t-1)$ key servers.

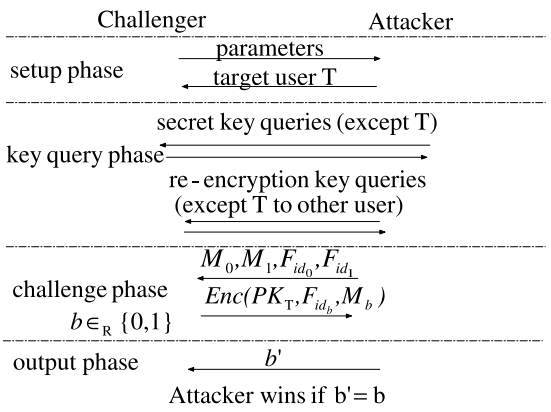


Figure 3.5: The security game for the chosen plaintext attack.
Not only secret key shares but also re-encryption keys are queried by the attacker in the security game.

We model this attack by the standard chosen plaintext attack of the proxy re-encryption scheme in a threshold version.

This game is the same with the previous one except the steps in the key share query. In the key share query, in addition to $(t-1)$ secret key shared of T, the attacker can also query all re-encryption keys except those re-encryption keys from T to other users. In the challenge phase, the attacker can choose the message identifiers for each of the chosen messages.

Figure 3.5 shows the full security game for the chosen plaintext attack in a threshold version. The challenger provides the system parameters. After the attacker chooses a target user T, the challenger gives $(t-1)$ key shares of the secret key SK_T of the target user T to the attacker. This step models that the attacker colludes with $(t-1)$ key servers. Then the attacker can query all re-encryption keys except those re-encryption keys from T to other users. In the challenge phase, the attacker chooses two messages M_0 and M_1 and indicates the identifiers $\mathsf{F}_{\mathsf{id}_0}, \mathsf{F}_{\mathsf{id}_1}$ for each of them. The challenger throws a

random coin b and encrypts the message M_b with T's public key PK_{T} and the identifier $\mathsf{F}_{\mathsf{id}_b}$. After getting the ciphertext from the challenger, the attacker outputs a bit b' for guessing b. In this game, the attacker wins if and only if $b' = b$. The advantage of the attacker is defined as $|1/2 - \Pr[b' = b]|$.

A cloud storage system in the advanced system model is secure if no probabilistic polynomial time attacker wins the game with a non-negligible advantage.

Chapter 4

Secure Cloud Storage System

At the starting point, we consider a secure cloud storage system, which supports the basic functions – data storing and retrieval. We design a threshold public key encryption scheme to protect the confidentiality of the stored data. However, it is not easy to maintain the decentralized structure of the whole network. One of the differences of our threshold encryption scheme from other ones is that the partial decryption is independently done by each key server. In this section, we present the threshold public key encryption scheme and the secure storage system that employs the encryption scheme. We also analyze the performance of the storage system and show that our storage system is secure.

4.1 Threshold Public Key Encryption

A threshold public key encryption consists of 6 algorithms: SetUp, KeyGen, ShareKeyGen, Enc, ShareDec, and Combine. SetUp generates the public pa-

$$PK = g^x, SK = x$$
$$Enc(PK, M) = (\alpha, \beta, \gamma) = (g^r, h, M\tilde{e}(g^x, h^r))$$
$$Dec(\alpha, \beta, \gamma) : M = \gamma / \tilde{e}(\alpha, \beta)^x$$

Figure 4.1: The primitive encryption scheme.
This is the primitive encryption scheme. We modify it into a threshold version.

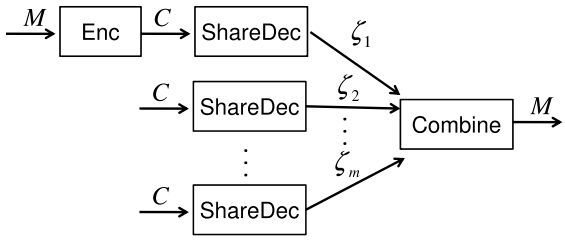


Figure 4.2: The flowchart of the threshold encryption scheme.
The flow of encryption and partial decryption of the threshold encryption scheme. Message M is encrypted as C. The decryption is performed by m partial decryption processes and a final combine process.

rameters of the whole system, and KeyGen generates a key pair, consisting of a public key PK and a secret key SK, for each user. Each user uses ShareKeyGen to share his secret key into m secret key shares such that any t of them can recover the secret key. Enc encrypts a given message by a public key PK, and outputs a ciphertext. ShareDec partially decrypts a given ciphertext by a secret key share and outputs a decryption share. Combine takes a set of decryption shares as input and outputs the message if and only if there are at least t decryption shares.

Figure 4.1 gives the non-threshold primitive encryption scheme. We modify this primitive and propose a threshold public key encryption scheme Π using bilinear maps as follows. Figure 4.2 shows a flow diagram of the encryption and partial decryption of the encryption schemes.

- $\mathsf{SetUp}(1^\lambda)$. To generate μ, run $\mathsf{Gen}(1^\lambda)$ and set $\mu = (p, \mathbb{G}_1, \mathbb{G}_2, \tilde{e}, g)$.

- $\mathsf{KeyGen}(\mu)$. To generate a key pair for a user, select $x \in_R \mathbb{Z}_p$ and set $\mathsf{PK} = g^x$, $\mathsf{SK} = x$.

- $\mathsf{ShareKeyGen}(\mathsf{SK}, t, m)$. The secret key shares $\mathsf{SK}_i = f(i)$ are derived by the polynomial $f(\mathrm{z})$, where

$$f(\mathrm{z}) = \mathsf{SK} + a_1\mathrm{z} + a_2\mathrm{z}^2 + \cdots + a_{t-1}\mathrm{z}^{t-1} \pmod{p},$$

  and $a_1, a_2, \ldots, a_{t-1} \in_R \mathbb{Z}_p$.

- $\mathsf{Enc}(\mathsf{PK}, M)$. To generate a ciphertext C of the message $M \in \mathbb{G}_2$, compute

$$C = (\alpha, \beta, \gamma) = (g^r, h, M\tilde{e}(g^x, h^r)),$$

  where $r \in_R \mathbb{Z}_p$, and $h \in_R \mathbb{G}_1$.

- $\mathsf{ShareDec}(\mathsf{SK}_i, C)$. Let $C = (\alpha, \beta, \gamma)$. By using the secret key share SK_i, a decryption share ζ_i of C is generated as follows.

$$\zeta_i = (\alpha_i, \beta_i, \beta_i', \gamma_i) = (\alpha, \beta, \beta^{\mathsf{SK}_i}, \gamma)$$

- $\mathsf{Combine}(\zeta_{i_1}, \zeta_{i_2}, \ldots, \zeta_{i_t})$. It combines the t values $(\beta_{i_1}', \beta_{i_2}', \ldots, \beta_{i_t}')$ to obtain $\beta^{\mathsf{SK}} = \beta^{f(0)}$ via Lagrange interpolation over exponents:

$$\beta^{\mathsf{SK}} = \prod_{i \in \mathsf{S}} \left((\beta_i')^{\Pi_{r \in \mathsf{S}, r \neq i} \frac{-i}{r-i}} \right)$$

  where $\mathsf{S} = \{i_1, i_2, \ldots, i_t\}$ and $\zeta_{i_j} = (\alpha_{i_j}, \beta, (\beta)_{i_j}', \gamma_{i_j})$ for all $1 \leq j \leq t$.

The output message is $M = \gamma / \tilde{e}(\alpha, \beta^{f(0)})$.

When a fixed h is used for a set of ciphertexts, the set of those ciphertexts are multiplicative homomorphic. The multiplicative homomorphic property is that given a ciphertext for M_1 and a ciphertext for M_2, a ciphertext for $M_1 \times M_2$ can be generated without knowing the secret key x, M_1, and M_2. Let $C_1 = \mathsf{Enc}(\mathsf{PK}, M_1)$ and $C_2 = \mathsf{Enc}(\mathsf{PK}, M_2)$, where

$$C_1 = (g^{r_1}, h, M_1 \tilde{e}(g^x, h^{r_1})) \text{ and}$$
$$C_2 = (g^{r_2}, h, M_2 \tilde{e}(g^x, h^{r_2})).$$

A new ciphertext C which is an encryption of $M_1 \times M_2$ under the public key PK is computed as follows:

$$C = (g^{r_1} g^{r_2}, h, M_1 \tilde{e}(g^x, h^{r_1}) M_1 \tilde{e}(g^x, h^{r_2}))$$
$$= (g^{r_1 + r_2}, h, M_1 M_2 \tilde{e}(g^x, h^{r_1 + r_2}))$$

4.2 System Construction

We assume that there are n storage servers which store data and m key servers which own secret key shares and perform partial decryption. We consider that the owner has the public key $\mathsf{PK} = g^x$ and shares the secret key x to m key servers with a threshold t, where $m \geq t \geq k$. Let the k messages be $M_1, M_2, \ldots, M_k$. We use $h_{\mathsf{ID}} = H(M_1 || M_2 || \cdots || M_k)$ as the identifier for this set of messages, where $H : \{0,1\}^* \rightarrow \mathbb{G}_1$ is a secure hash function.

The storage process, illustrated in Figure 4.3, and the retrieval process

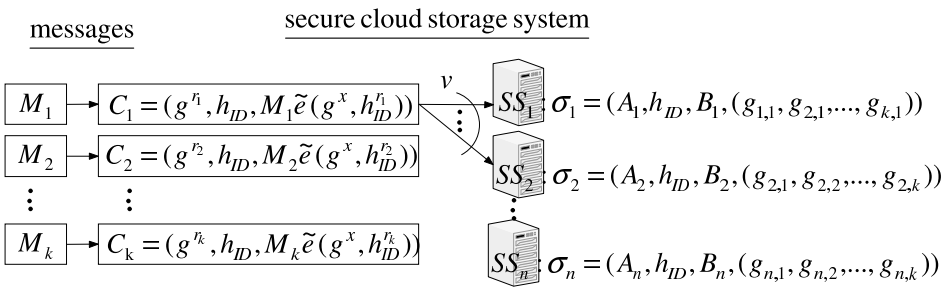


Figure 4.3: The storage process of our first secure cloud storage system. *The messages are encrypted and distributed among the storage servers. Each storage server combines all received ciphertexts and stores the result and the chosen coefficients.*

are described in the following.

- *Storage process.* To store k messages, the storage process is as follows:

  1. *Message encryption.* The owner encrypts all k messages via the threshold public key encryption Π with the same h_{ID}, where $h_{\mathsf{ID}} = H(M_1||M_2||\cdots||M_k)$ is the identifier for the set of messages $M_1, M_2, \ldots, M_k$. Let the ciphertext of M_i be

  $$C_i = (\alpha_i, \beta, \gamma_i) = (g^{r_i}, h_{\mathsf{ID}}, M_i\tilde{e}(g^x, h_{\mathsf{ID}}^{r_i})),$$

  where $r_i \in_R \mathbb{Z}_p$, $1 \leq i \leq k$.

  2. *Ciphertext distribution.* For each C_i, the owner randomly chooses v storage servers (with replacement) and sends each of them a copy of C_i.

  3. *Decentralized encoding.* For all received ciphertexts with the same message identifier h_{ID}, the storage server SS_j groups them as N_j. The storage server SS_j selects a random coefficient $g_{i,j}$ from $\mathbb{Z}_p$

for each $C_i \in N_j$ and sets $g_{i,j} = 0$ for $C_i \notin N_j$. This step forms a generator matrix $G = [g_{i,j}]_{1 \le i \le k, 1 \le j \le n}$ of the decentralized erasure code.

Each storage server SS_j computes the following (A_j, B_j),

$$A_j = \prod_{C_i \in N_j} \alpha_i^{g_{i,j}} \text{ and } B_j = \prod_{C_i \in N_j} \gamma_i^{g_{i,j}}$$

and stores

$$\sigma_j = (A_j, h_{\mathsf{ID}}, B_j, (g_{1,j}, g_{2,j}, \ldots, g_{k,j})).$$

In fact, $(A_j, h_{\mathsf{ID}}, B_j)$ is a ciphertext for $\prod_{1 \le i \le k} M_i^{g_{i,j}}$ since

$$\begin{aligned}
&(A_j, h_{\mathsf{ID}}, B_j) \\
&= (\prod_{C_i \in N_j} (g^{r_i})^{g_{i,j}}, h_{\mathsf{ID}}, \prod_{C_i \in N_j} (M_i \tilde{e}(g^x, h_{\mathsf{ID}}^{r_i}))^{g_{i,j}}) \\
&= (g^{\prod_{C_i \in N_j} r_i g_{i,j}}, h_{\mathsf{ID}}, (\prod_{C_i \in N_j} M_i^{g_{i,j}}) \tilde{e}(g^x, h_{\mathsf{ID}}^{\prod_{C_i \in N_j} r_i g_{i,j}})) \\
&= (g^{\tilde{r}}, h_{\mathsf{ID}}, (\prod_{C_i \in N_j} M_i^{g_{i,j}}) \tilde{e}\left(g^x, h_{\mathsf{ID}}^{\tilde{r}}\right)),
\end{aligned}$$

where $\tilde{r} = \prod_{C_i \in N_j} r_i g_{i,j}$.

- *Retrieval process.* To retrieve k messages, the retrieval process is as follows:

  1. *Retrieval command.* The owner sends a command to the m key servers with the message identifier h_{ID}.

2. *Partial decryption.* Each key server KS_i randomly queries u storage servers with the message identifier h_{ID} and obtains at most u stored data σ_j from the storage servers. Then the key server KS_i performs $\mathsf{ShareDec}$ on each received ciphertext by its secret key share SK_i to obtain a decryption share of the ciphertext. Assume that KS_i receives σ_j. KS_i decrypts the ciphertext $(A_j, h_{\mathsf{ID}}, B_j)$ as a decryption share $\zeta_{i,j} = (A_j, h_{\mathsf{ID}}, h_{\mathsf{ID}}^{\mathsf{SK}_i}, B_j)$, and sends the following to the owner:

$$\tilde{\zeta}_{i,j} = (A_j, h_{\mathsf{ID}}, h_{\mathsf{ID}}^{\mathsf{SK}_i}, B_j, (g_{1,j}, g_{2,j}, \ldots, g_{k,j}))$$

3. *Combining and decoding.* The owner chooses $\tilde{\zeta}_{i_1,j_1}, \tilde{\zeta}_{i_2,j_2}, \ldots, \tilde{\zeta}_{i_t,j_t}$ from all received data $\tilde{\zeta}_{i,j}$ and computes $h_{\mathsf{ID}}^{\mathsf{SK}} = h_{\mathsf{ID}}^{f(0)} = h_{\mathsf{ID}}^{x}$ by the Lagrange interpolation over exponents, where $i_1 \neq i_2 \neq \cdots \neq i_t$ and $\mathsf{S} = \{i_1, i_2, \ldots, i_t\}$:

$$h_{\mathsf{ID}}^{x} = \prod_{i \in \mathsf{S}} (h_{\mathsf{ID}}^{\mathsf{SK}_i})^{\prod_{r \in \mathsf{S}, r \neq i} \frac{-i}{r-i}}$$

If the number of the received $\tilde{\zeta}_{i,j}$ is more than t, the owner randomly selects t out of them. If the number is less than t, the retrieval process fails. After having h_{ID}^{x}, the owner reconsiders all received data and chooses $\tilde{\zeta}_{i_1,j_1}, \tilde{\zeta}_{i_2,j_2}, \ldots, \tilde{\zeta}_{i_k,j_k}$ with $j_1 \neq j_2 \neq \cdots \neq j_k$. By using h_{ID}^{x}, the owner decrypts $\zeta_{i,j}$ as w_j for all

$(i,j) \in \{(i_1,j_1),(i_2,j_2),\ldots,(i_k,j_k)\}$:

$$w_j = \frac{B_j}{\tilde{e}(A_j, h^x_{\mathsf{ID}})} = \prod_{C_l \in N_j} M_l^{g_{l,j}} \tag{4.1}$$

The owner then computes

$$K^{-1} = [d_{i,j}]_{1 \le i,j \le k},$$

where $K = [g_{i,j}]_{1 \le i \le k, j \in \{j_1,j_2,\ldots,j_k\}}$. If K is not invertible, the retrieval process fails. Otherwise, the owner successfully obtains M_i, $1 \le i \le k$, by the following computation:

$$\begin{aligned} & w_{j_1}^{d_{1,i}} w_{j_2}^{d_{2,i}} \cdots w_{j_k}^{d_{k,i}} \\ &= M_1^{\sum_{l=1}^{k} g_{1,j_l} d_{l,i}} M_2^{\sum_{l=1}^{k} g_{2,j_l} d_{l,i}} \cdots M_k^{\sum_{l=1}^{k} g_{k,j_l} d_{l,i}} \\ &= M_1^{\tau_1} M_2^{\tau_2} \cdots M_k^{\tau_k} \\ &= M_i, \end{aligned}$$

where $\tau_r = \sum_{l=1}^{k} g_{r,j_l} d_{l,i} = 1$ if $r = i$ and $\tau_r = 0$ otherwise.

An example is given in Figure 4.4. In the ciphertext distribution step, the ciphertext C_1 is distributed to SS_1, SS_2, and SS_3. The ciphertext C_2 is distributed to SS_2 and SS_3 only. After receiving $\tilde{\zeta}_{1,1}$, $\tilde{\zeta}_{1,2}$, $\tilde{\zeta}_{2,2}$, and $\tilde{\zeta}_{2,3}$, the owner computes h^x_{ID} from $\tilde{\zeta}_{1,1}$ and $\tilde{\zeta}_{2,2}$. By using h^x_{ID}, the owner computes the encoded messages, $M_1^{g_{1,2}} M_2^{g_{2,2}}$ and $M_1^{g_{1,3}} M_2^{g_{2,3}}$, and decodes them to get messages M_1 and M_2.

Our design uses two techniques. Firstly, for retrieving messages, the

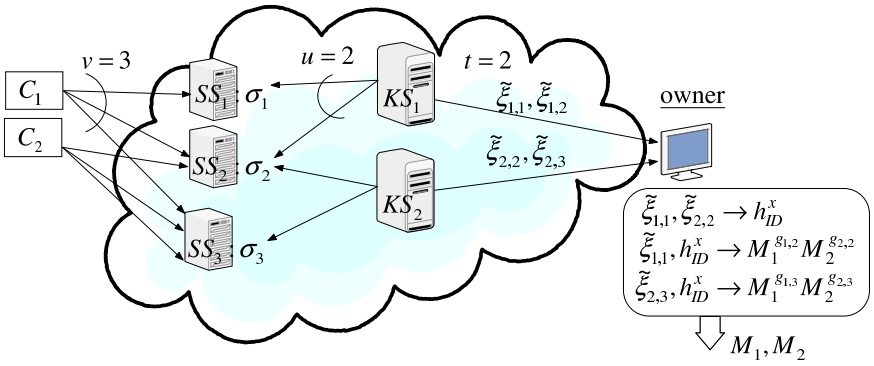


Figure 4.4: A storage system using the secure decentralized erasure code.

decryption process can be performed *before* the decoding process. Secondly, the decryption process can be performed by the key servers independently. The first technique comes from the multiplicative homomorphic property of our encryption scheme. For those k messages, a fixed message identifier h_{ID} is used. As a result, the set of ciphertexts is multiplicative homomorphic. An encoding result of ciphertexts $C_1, C_2, \ldots, C_k$ is also an encryption of an encoding result of messages $M_1, M_2, \ldots, M_k$. As for the second key technique, the design of the encryption scheme embeds the decryption power at the value h_{ID}^x, while h_{ID} is the message identifier. With h_{ID}^x, the owner can decrypt all ciphertexts marked with the message identifier h_{ID}. A key server KS_i can compute a share $h_{\mathsf{ID}}^{\mathsf{SK}_i}$ of h_{ID}^x. With at least t key servers, h_{ID}^x can be computed.

4.2.1 Correctness

The correctness is that the owner A correctly retrieves the messages with an overwhelming probability. The correctness of the encryption and the decryption for user A is that any ciphertext $C' = (g^r, h_{\mathsf{ID}}, w\tilde{e}(g^x, h_{\mathsf{ID}}^r))$ can be correctly decrypted to w, when there are more than t active key servers

who have shares of $\mathsf{SK_A}$. This correctness can be seen from Equation (4.1). The user combines t decryption shares and then correctly gets the encoded message.

4.3 Analysis

We analysis the performance, the probability of a successful retrieval, and the security of the secure cloud storage system.

4.3.1 Performance Analysis

We analyze the computation cost and the storage cost. Let the bit-length of the element in the group $\mathbb{G}_1$ be l_1 and $\mathbb{G}_2$ be l_2.

Computation cost. We measure the computation cost in the number of pairing operations, modular exponentiations in $\mathbb{G}_1$ and $\mathbb{G}_2$, modular multiplications in $\mathbb{G}_1$ and $\mathbb{G}_2$, and arithmetic operations over $GF(p)$. Those operations are denoted as $\mathsf{Pairing}$, Exp_1, Exp_2, Mult_1, Mult_2, and F_p, respectively. We consider the cost for k messages together since the storage process and retrieval process are designed for a set of k messages. The cost is listed in Table 4.1. In fact, F_p has much lower cost than Mult_1 and Mult_2. One Exp_1 is about $1.5\lceil \log_2 p \rceil$ Mult_1 on average (by using the fast square and multiply algorithm). That is, when p is about 1000 bits, one Exp_1 is about 1500 Mult_1 on average. Similarly, Exp_2 is about $1.5\lceil \log_2 p \rceil$ Mult_2 on average.

Since in practice the coefficients can be chosen from a smaller set, the measure of the computation cost of the Exp_1 and Exp_1 is an over-estimation. $\mathsf{Pairing}$ is considered as a more expensive operation than Exp. However,

Operations	Computation cost
Message encryption (for k messages)	k $\mathsf{Pairing}$ + $2k$ Exp_1 + k Mult_2
Decentralized encoding (for each SS)	k Exp_1 + k Exp_2 + $(k-1)$ Mult_1 + $(k-1)$ Mult_2
Partial decryption (for t KS)	t Exp_1
Combining	k $\mathsf{Pairing}$ + k Mult_2 + $\mathrm{O}(t^2)$ F_p
Decoding	k^2 Exp_2 + $(k-1)k$ Mult_2 + $\mathrm{O}(k^3)$ F_p

- $\mathsf{Pairing}$: a pairing computation of $\tilde{e}$.
- Exp_1 and Exp_2: a modular exponentiation computation in $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively.
- Mult_1 and Mult_2: a modular multiplication computation in $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively.
- F_p: an arithmetic operation in $GF(p)$.

Table 4.1: Computation cost of each step in our first secure storage system.

some improved algorithms [48, 49] are proposed for accelerating the pairing computation.

In the storage process, for each message encryption, generating α_i requires one Exp_1, and generating γ_i requires one Exp_1, one $\mathsf{Pairing}$, and one Mult_2. Hence, in the message encryption step for k messages, the cost is (k $\mathsf{Pairing}$ + $2k$ Exp_1 + k Mult_2). In the ciphertext distribution step, no computation occurs. In the encoding step, each SS_i encodes all received messages. Here we use a worse cast estimation that each SS_i receives k messages. To compute A_i, SS_i requires k Exp_1 and $(k-1)$ Mult_1 while to compute B_i, the cost is k Exp_2 and $(k-1)$ Mult_2.

For the partial decryption step, each KS_i performs one Exp_1 to get $h_{\mathsf{ID}}^{\mathsf{SK}_i}$. For a successful retrieval, t key servers would be sufficient; hence, for this

step, we consider the total cost of t key servers. That is t Exp_1. For the combining and decoding step, we split it into two sub-steps: the combining sub-step and the decoding sub-step. The combining sub-step includes the computation of h_{ID}^x and the computation of codeword elements w_j's from the decryption shares $\tilde{\zeta}_{i,j}$'s. The computation of h_{ID}^x is a Lagrange interpolation over exponents in $\mathbb{G}_1$, which requires $O(t^2)$ F_p, t Exp_1 , and $(t-1)$ Mult_1. Computing w_j from A_j, B_j, and h_{ID}^x requires one $\mathsf{Pairing}$ and one modular division, which takes 2 Mult_2. The decoding sub-step includes the matrix inversion and the computation of messages M_i's from codeword elements w_j's. The matrix inversion takes $O(k^3)$ arithmetic operations over $GF(p)$, and the decoding for each message takes k Exp_2 and $(k-1)$ Mult_2.

Storage cost. The storage cost in a key server for a user is $\lceil \log_2 p \rceil$ because the key server only requires to store the secret key share. The main storage cost lies on the storage servers.

We measure the storage cost in bits as the average cost in a storage server for a message bit. To store k messages, each storage server SS_j stores $(A_j, h_{\mathsf{ID}}, B_j)$ and the coefficient vector $(g_{1,j}, g_{2,j}, \ldots, g_{k,j})$. The total cost in a storage server is $(2l_1 + l_2 + k\lceil \log_2 p \rceil)$ bits, where $A_j, h_{\mathsf{ID}} \in \mathbb{G}_1$, and $B_j \in \mathbb{G}_2$; hence, the average cost for a message bit is $(2l_1 + l_2 + k\lceil \log_2 p \rceil)/kl_2$ bits, which is dominated by $\lceil \log_2 p \rceil / l_2$ for a sufficient large k. In practicality, $g_{i,j}$'s are chosen from a much smaller set than $\mathbb{Z}_p$. Then we can use fewer bits to represent $g_{i,j}$'s. This reduces the storage cost in each storage server.

4.3.2 Successful Retrieval Probability

When n and k are fixed, u and v affect the probability of a successful retrieval. We investigate the relations of these parameters for the success probability. The results are given in Theorem 1 and Theorem 2.

To retrieve all k messages, the key servers have to get k stored data σ_{j_1}, σ_{j_2}, $\cdots$, σ_{j_k} from k different storage servers $\mathsf{SS}_{j_1}, \mathsf{SS}_{j_2}, \ldots, \mathsf{SS}_{j_k}$ and apply $\mathsf{ShareDec}$ to acquire $\tilde{\zeta}_{i_1,j_1}$, $\tilde{\zeta}_{i_2,j_2}$, ..., $\tilde{\zeta}_{i_k,j_k}$. Furthermore, a $k \times k$ matrix K formed by the coefficient vectors in $\tilde{\zeta}_{i_1,j_1}$, $\tilde{\zeta}_{i_2,j_2}$, ..., $\tilde{\zeta}_{i_k,j_k}$ needs to be invertible in order to solve the k messages. The random process is on the selection of distinct SS_{j_1}, SS_{j_2}, ..., SS_{j_k} by the key servers and the coefficient vectors in σ_{j_1}, σ_{j_2}, ..., and σ_{j_k}. Let E_1 be the event that less than k distinct storage servers are queried by the key servers. For the generator matrix G implicitly generated by the owner and the storage servers, let E_2 be the event that the submatrix K of k columns $j_1, j_2, \ldots, j_k$ of G is non-invertible. The Figure 4.5 shows the probability space of the successful retrieval event. The outer circle presents the sample space. The solid circle presents the event E_1 and the inner circle shows the event $\overline{\mathrm{E}}_2$. The event of a successful retrieval is showed as the shadow area. Thus, the probability of a successful retrieval by the owner is

$$1 - \Pr[\mathrm{E}_1] - \Pr[\mathrm{E}_2|\overline{\mathrm{E}}_1]\Pr[\overline{\mathrm{E}}_1] \tag{4.2}$$

We analyze suitable settings of m, v, and u, where $n = ak^{3/2}$ and $n = ak$, respectively and the results are listed in the following:

1. $n = ak^{3/2}$, $a > \sqrt{2}$, $m \geq t \geq k > 1$, $v = bk^{1/2} \ln k$, $u = 2$ with $b > 5a$

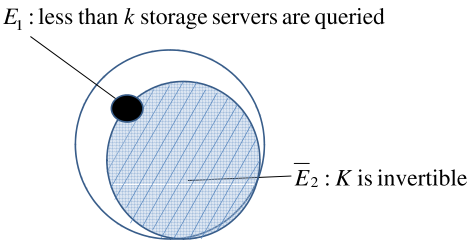


Figure 4.5: The event of a successful retrieval is showed as the shadow area.

2. $n = ak$, $a > 1$, $m = t = k > 1$, $v = b_1 \ln k$, $u = b_2 \ln k$ with $b_1 > 5a$ and $b_2 > 4 + 3/\ln a$

We image a networked storage system that consists of a large number of storage servers. The number k of stored messages each time is much less than n. Thus, the first setting of $n = ak^{3/2}$ is better than the second setting of $n = ak$. Although, in the regular coding theory, the constant information rate for the second setting may be preferred, the first setting is more suitable for the application to practical networked storage systems.

Theorem 1. *Assume that there are k messages, n storage servers, and m key servers where $n = ak^{3/2}$, $m \geq t \geq k > 1$ and a is a constant with $a > \sqrt{2}$. For $v = bk^{1/2} \ln k$ and $u = 2$ with $b > 5a$, the probability of a successful retrieval is at least $1 - k/p - o(1)$.*

Proof. To analyze $\Pr[\mathrm{E}_1]$, we consider that each storage server is a bin and each key server has u balls, where $u = 2$. When a key server queries a storage server, we consider that the key server throws a ball into the bin. Because the key servers make queries randomly, those balls are randomly thrown into

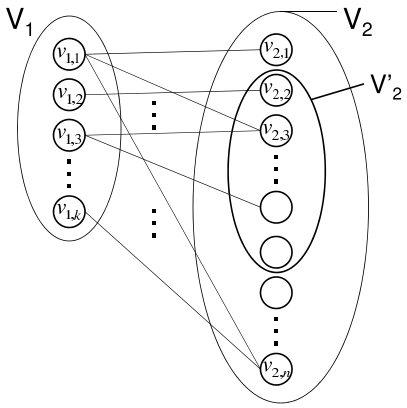


Figure 4.6: The random bipartite graph H.

The random bipartite graph H has two sets of vertices and random edges. The random subgraph H' is defined by a random set of k vertices in V_2 and the set of k vertices in V_1.

n bins. The probability that less than k bins contain balls is:

$$\begin{aligned}
\Pr[\mathrm{E}_1] &\le C_{k-1}^{n}\left(\frac{k-1}{n}\right)^{2m} \\
&= \frac{n(n-k+2)}{(k-1)1}\frac{(n-1)(n-k+3)}{(k-2)2} \\
&\quad \cdots \frac{(n-\lfloor\frac{k-2}{2}\rfloor)(n-\lceil\frac{k-2}{2}\rceil)}{\lceil\frac{k-1}{2}\rceil\lfloor\frac{k-1}{2}\rfloor}\left(\frac{k-1}{n}\right)^{2m} \\
&\le \frac{2n(n-k+2)}{k}^{\frac{k-1}{2}}\left(\frac{k-1}{n}\right)^{2m} \\
&\le \left(2a^2k^2-2ak^{3/2}+4ak^{1/2}\right)^{\frac{k-1}{2}}\left(\frac{k-1}{n}\right)^{2k} \quad (\because n = ak^{3/2}) \\
&\le \left(\frac{2a^2k^2-2ak^{3/2}+4ak^{1/2}}{a^4k^2}\right)^{\frac{k}{2}}\left(\frac{k-1}{k}\right)^{2k} \\
&= o(1) \qquad\qquad (\because a > \sqrt{2}) \qquad (4.3)
\end{aligned}$$

The event E_2 under the condition $\overline{\mathrm{E}}_1$ can be modeled by forming a perfect matching in the random bipartite graph H with respect to G. The random

bipartite graph H is illustrated in Figure 4.7 and constructed as follows. Let each ciphertext C_i be a vertex $v_{1,i}$ and V_1 be the set of all vertices for C_i's. Let each storage server SS_j be a vertex $v_{2,j}$ and V_2 be the set of all vertices for SS_j's. When a ciphertext C_i is distributed to the storage server SS_j, there is an edge $(v_{1,i},\ v_{2,j})$. The matrix K induces a subgraph H' of the bipartite graph H. The subgraph H' consists of all vertices in V_1, a subset $V_2' \subset V_2$ that V_2' is a subset of queried storage servers and $|V_2'| = k$, and edges $(v_{1,i}, v_{2,j})$ for all $v_{1,i} \in V_1$ and $v_{2,j} \in V_2'$. If H' has no perfect matching, K is not invertible. If H' has a perfect matching, K is non-invertible if and only if $det(K) = 0$. The value of $det(K)$ depends on the random coefficients chosen by the storage servers. Let E_3 be the event that H' has no perfect matching, and E_4 be the event that $det(K) = 0$. We have,

$$\begin{aligned}\Pr[\mathrm{E}_2|\overline{\mathrm{E}}_1] &= \Pr[\mathrm{E}_3|\overline{\mathrm{E}}_1] + \Pr[\mathrm{E}_4|\overline{\mathrm{E}}_3 \wedge \overline{\mathrm{E}}_1]\Pr[\overline{\mathrm{E}}_3|\overline{\mathrm{E}}_1] \\ &\leq \Pr[\mathrm{E}_3|\overline{\mathrm{E}}_1] + \Pr[\mathrm{E}_4|\overline{\mathrm{E}}_3 \wedge \overline{\mathrm{E}}_1] \qquad (4.4)\end{aligned}$$

We analyze the probability of E_3 conditioned on $\overline{\mathrm{E}}_1$ by using the Hall's Lemma in the following form [34].

Lemma 1. *(Hall's Lemma.)*

Let H' be a bipartite graph with vertex sets V_1 and V_2', where $|V_1| = |V_2'| = k$. If H' has no isolated vertex and no perfect matching, then there exists a set $\mathrm{A} \subset V_1$ or $\mathrm{A} \subset V_2$ such that:

- $2 \leq |A| \leq \frac{k+1}{2}$
- *The number of neighbors of* A *is* $|\mathrm{A}| - 1$.

- *The subgraph induced by* A *and its neighbors is connected.*

Hence, there are two cases that H' has no perfect matching. First, H' has at least one isolated vertex. Second, H' has no isolated vertex and a set A satisfies the above conditions. Let $\mathrm{E_I}$ be the event that H' has at least one isolated vertex and $\mathrm{E_A}$ be the event that some set A satisfies the conditions. We obtain

$$\Pr[\mathrm{E}_3|\overline{\mathrm{E}}_1] \leq \Pr[\mathrm{E_I}|\overline{\mathrm{E}}_1] + \Pr[\mathrm{E_A}|\overline{\mathrm{E}}_1] \tag{4.5}$$

Starting from $\mathrm{E_I}$, we consider each vertex in V_2 as a bin and each edge from V_1 to V_2 as a ball. When an edge connects to a vertex in V_2, a ball is thrown into the bin. Consider the subset B of the bins corresponding to the subset V_2' of V_2. Thus, B contains k bins. $\mathrm{E_I}$ means that there is one or more empty bins in B. For a fixed bin in B, the probability of the bin being empty is $(1-1/n)^{bk^{3/2}\ln k}$ since there are $bk^{3/2}\ln k$ balls. By using the union bound on k bins, we have the probability of $\mathrm{E_I}$ conditioned on $\overline{\mathrm{E}}_1$ as:

$$\begin{aligned}
\Pr[\mathrm{E_I}|\overline{\mathrm{E}}_1] &\leq k(1-1/n)^{bk^{3/2}\ln k} \\
&= k\left(1-\frac{1}{ak^{3/2}}\right)^{ak^{3/2}\cdot\frac{b}{a}\ln k} && (\because n = ak^{3/2}) \\
&\leq k(e^{-\frac{b}{a}\ln k}) && (\because 1-x \leq e^{-x}) \\
&= \left(\frac{1}{k}\right)^{\frac{b}{a}-1} \\
&= o(1) && (\because b > 5a)
\end{aligned} \tag{4.6}$$

As for $\Pr[\mathrm{E_A}|\overline{\mathrm{E}}_1]$, we separate the event into two sub-events by $\mathrm{A} \subset V_1$

or $\mathrm{A} \subset V_2'$. Thus,

$$\begin{aligned}
\Pr[\mathrm{E_A}|\overline{\mathrm{E}}_1] &= \Pr[\mathrm{E_A \text{ and } A} \subset V_1|\overline{\mathrm{E}}_1]\Pr[\mathrm{A} \subset V_1] \\
&\quad + \Pr[\mathrm{E_A \text{ and } A} \subset V_2'|\overline{\mathrm{E}}_1]\Pr[\mathrm{A} \subset V_2'] \\
&\leq \Pr[\mathrm{E_A \text{ and } A} \subset V_1|\overline{\mathrm{E}}_1] \\
&\quad + \Pr[\mathrm{E_A \text{ and } A} \subset V_2'|\overline{\mathrm{E}}_1]
\end{aligned}$$

For $\Pr[\mathrm{E_A \text{ and } A} \subset V_2'|\overline{\mathrm{E}}_1]$, we further divide the event into sub-events according to the size of A and use the union bound again. Consider a set $A \subset V_2'$ with $|A| = i$. The event $\mathrm{E_A}$ conditioned on $\overline{\mathrm{E}}_1$ can be overestimated by the event that $\Gamma(A) \subset V_1$ and $|\Gamma(A)| = i - 1$. In other words, there is a set $A' \subset V_1$ with $|A'| = i - 1$ such that all vertices in $V_1 \backslash A'$ only connect to vertices in $V_2 \backslash A$. Thus, we have

$$
\begin{aligned}
&\Pr[\mathrm{E_A} \text{ and } \mathrm{A} \subset V_2' | \overline{\mathrm{E}}_1] \\
&\leq \sum_{i=2}^{(k+1)/2} \Pr[\mathrm{E_A}, \mathrm{A} \subset V_2' \text{ and } |A| = i | \overline{\mathrm{E}}_1] \\
&= \sum_{i=2}^{(k+1)/(2)} C_i^k C_{i-1}^k (\frac{n-i}{n})^{(k-i+1)bk^{1/2}\ln k} \\
&\leq \sum_{i=2}^{(k+1)/(2)} (\frac{ek}{i})^{2i} (\frac{n-i}{n})^{(k-i+1)bk^{1/2}\ln k} \qquad (\because C_i^k \leq (\frac{ek}{i})^i) \\
&\leq k \max_i \{(\frac{ek}{i})^{2i} (\frac{n-i}{n})^{(k-i+1)bk^{1/2}\ln k}\} \\
&= \max_i \{\exp(2i(1-\ln i) \\
&+ \ln k[b(k-i+1)k^{1/2} \ln\left(\frac{n-i}{n}\right) + 2i + 1])\}
\end{aligned} \tag{4.7}
$$

To achieve $\Pr[\mathrm{E_A} \text{ and } \mathrm{A} \subset V_2' | \overline{\mathrm{E}}_1] = o(1)$ as $k \to \infty$, it is sufficient to have

$$
b(k-i+1)k^{1/2} \ln\left(\frac{n-i}{n}\right) + 2i + 1 < 0 \tag{4.8}
$$

Since $(n-i)/n = 1 - i/n < e^{-(i/n)}$, we have

$$
b(k-i+1)k^{1/2} \left(\frac{-i}{n}\right) + 2i + 1 < 0 \tag{4.9}
$$

By Equation (4.9), we need

$$
b > \frac{(2i+1)ak^{3/2}}{(k-i+1)k^{1/2}i} = \frac{(2i+1)ak}{(k-i+1)i} \tag{4.10}
$$

Since $b > 5a$, for $2 \leq i \leq (k+1)/2$, Equation (4.10) holds. It implies that

$$\Pr[\mathrm{E_A} \text{ and } \mathrm{A} \subset V_2' | \overline{\mathrm{E}}_1] = o(1)$$

as $k \to \infty$. Similarly, we can get a lower bound for b from the case of $A \subset V_1$ and the bound is satisfied by $b > 5a$.

For $\Pr[\mathrm{E}_4 | \overline{\mathrm{E}}_3 \wedge \overline{\mathrm{E}}_1]$, that is, $det(A) = 0$, we treat each coefficient, randomly chosen from Z_p, in the matrix K as a variable. Thus, $det(K)$ is a mutlivariate function. Since there is a perfect matching in the induced graph H', $det(K)$ is not identically zero (i.e. $det(K) \not\equiv 0$) and the degree of $det(K)$ is k. We use the Schwartz-Zeppel Theorem:

Lemma 2. *(Schwartz-Zeppel Theorem [50])*
Let $Q(x_1, x_2, \ldots, x_n) \in F[x_1, x_2, \ldots, x_n]$ be a multivariate polynomial of total degree d. Fix and finite set S and let $r_1, r_2, \ldots, r_n$ be chosen independently and uniformly at random from S. Then

$$\Pr[Q(r_1, r_2, \ldots, r_n) = 0 | Q(x_1, x_2, \ldots, x_n) \not\equiv 0] \leq \frac{d}{|S|}$$

From the Schwartz-Zeppel Theorem, the probability that the randomly chosen coefficients make $det(K) = 0$ is no more than k/p, i.e., $\Pr[\mathrm{E}_4 | \overline{\mathrm{E}}_3 \wedge \overline{\mathrm{E}}_1] \leq k/p$. Thus, we have

$$\Pr[\mathrm{E}_2 | \overline{\mathrm{E}}_1] \leq k/p + o(1)$$

and conclude the proof of Theorem 1. □

For another setting for v and u, where $n = ak$ and $m = k$, we have the following theorem.

Theorem 2. *Assume that there are k messages, n storage servers, and m key servers, where $n = ak$ for a fixed constant $a > 1$ and $m = t = k > 1$. For $v = b_1 \ln k$, $u = b_2 \ln k$, $b_1 > 5a$ and $b_2 > 4 + 3/\ln a$, the probability of a successful retrieval is at least $1 - k/p - o(1)$, where p is the size of the used group.*

Proof. By the proof of Theorem 1, we analyze two events E_1 and E_2 similarly. We have

$$\Pr[\mathrm{E}_2] < k/p + o(1).$$

To bound E_1, we start with

$$\Pr[\mathrm{E}_1] \leq C_{k-1}^{n} (\frac{k-1}{n})^{b_2 k \ln k}.$$

By the bound for C_{k-1}^{n} in the proof of Theorem 1 and $n = ak$, we obtain:

$$\begin{aligned}
\Pr[\mathrm{E}_1] &\leq \left(\frac{2n(n-k+2)}{k-1}\right)^{\frac{k+1}{2}} \left(\frac{k-1}{n}\right)^{b_2 k \ln k} \\
&\leq (4a^2 k)^{\frac{k+1}{2}} (\frac{k}{ak})^{b_2 k \ln k} \\
&= a^{k+1+[\log_a (4k)]\frac{k+1}{2} - b_2 k \ln k} \\
&= o(1) \qquad (\because b_2 > 4 + \frac{3}{\ln a}, k > 1)
\end{aligned}$$

as $k \to \infty$. Therefore, the probability of a successful retrieval is

$$1 - \Pr[\mathrm{E}_1] - \Pr[\mathrm{E}_2 | \overline{\mathrm{E}}_1] \Pr[\overline{\mathrm{E}}_1] \geq 1 - k/p - o(1).$$

□

In our settings, if we increase the value of u, the value of v can be decreased while keeping the same probability of the event of a successful data retrieval. Although a smaller v makes that a storage server contains less information on average. A higher u value makes more storage servers are queried by key servers. As a result, the probability of the event of a success data retrieval may remain.

4.3.3 Security Analysis

As mentioned in Section 3, the security of the cloud storage system relies on the underlying encryption scheme. We show the above threshold public key encryption scheme is secure in Theorem 3 as follows.

Theorem 3. *The above threshold public key encryption system is chosen plaintext secure (CPA secure) under the decisional bilinear Diffie-Hellman assumption in the standard model.*

Proof. We prove by contradiction. Assume that there is an algorithm $\mathcal{A}$ winning the CPA security game against our encryption scheme with advantage 2ϵ. We can construct an algorithm $\mathcal{A}'$ solving the decisional bilinear Diffie-Hellman problem with advantage ϵ. The reduction is illustrated in Figure **??** and described as follows.

- *Setup.* The input of $\mathcal{A}'$ is $(g, g^x, g^y, g^z, \mathbb{Q})$ with public parameters $(\tilde{e}, \mathbb{G}_1, \mathbb{G}_2, p)$. Then $\mathcal{A}'$ sends (μ, PK, t, n) to $\mathcal{A}$, where $\mu = (p, \mathbb{G}_1, \mathbb{G}_2, \tilde{e}, g)$, $\mathsf{PK} = g^x$, t is a threshold value and n is the number of secret key shares. This implicitly sets $\mathsf{SK} = x$.

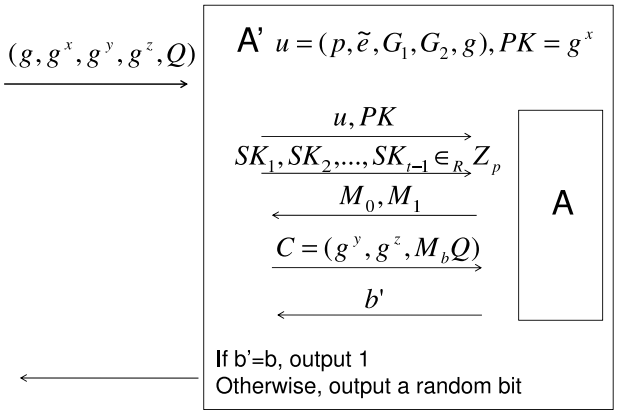


Figure 4.7: The reduction for the security of our first secure storage system. *The algorithm $\mathcal{A}'$ aims to solve the decisional Diffie-Hellman problem and uses $\mathcal{A}$ as a subroutine.*

- *Key share query.* To answer $\mathcal{A}$'s queries $q_1, q_2, \ldots, q_{t-1}$ for $(t-1)$ secret key shares, $\mathcal{A}'$ sets $\mathsf{SK}_{q_1}, \mathsf{SK}_{q_2}, \ldots, \mathsf{SK}_{q_{t-1}}$ as random values and sends them to $\mathcal{A}$. Wlog, assume that $q_1, q_2, \ldots, q_{t-1}$ are all different.

- *Challenge.* $\mathcal{A}$ gives two messages M_0 and M_1. $\mathcal{A}'$ randomly selects $b \in \{0, 1\}$ and encrypts M_b as:

$$C = \mathsf{Enc}(\mathsf{PK}, M_b) = (g^y, g^z, M_b \mathbb{Q})$$

- *Output.* $\mathcal{A}'$ sends C to $\mathcal{A}$ and gets $\mathcal{A}$'s output b'. If $b' = b$, then $\mathcal{A}'$ outputs 0 for guessing that $\mathbb{Q} = \mathbb{Q}_0 = \tilde{e}(g, g)^{xyz}$. If $b' \neq b$, then $\mathcal{A}'$ outputs 1 for guessing that $\mathbb{Q} = \mathbb{Q}_1 = \tilde{e}(g, g)^r$.

When $\mathbb{Q} = \mathbb{Q}_0 = \tilde{e}(g, g)^{xyz}$, C is a ciphertext of M_b; thus, $\mathcal{A}$ has advantage 2ϵ winning the game, i.e. $\Pr[b' = b | \mathbb{Q} = \tilde{e}(g, g)^{xyz}] = 1/2 + 2\epsilon$. When $\mathbb{Q} = \mathbb{Q}_1 = \tilde{e}(g, g)^r$ for some random r, the distributions of $(g^y, g^z, M_0 \mathbb{Q})$

and $(g^y, g^z, M_1\mathbb{Q})$ are identical because for any r, there exists r' such that $M_0\tilde{e}(g,g)^r = M_1\tilde{e}(g,g)^{r'}$. Thus, we have $\Pr[b' = b|\mathbb{Q} = \tilde{e}(g,g)^r] = 1/2$. The advantage of $\mathcal{A}'$ is:

$$
\begin{aligned}
&|\Pr[\mathcal{A}' \to 0|\mathbb{Q} = \mathbb{Q}_0]\Pr[\mathbb{Q} = \mathbb{Q}_0] \\
&\quad + \Pr[\mathcal{A}' \to 1|\mathbb{Q} = \mathbb{Q}_1]\Pr[\mathbb{Q} = \mathbb{Q}_1] - \frac{1}{2}| \\
&= |(\frac{1}{2} + 2\epsilon) \times \frac{1}{2} + \frac{1}{2} \times \frac{1}{2} - 1/2| \\
&= \epsilon
\end{aligned}
$$

□

Chapter 5

Secure Cloud Storage with Data Forwarding

We now consider an advanced secure cloud storage system, which additionally supports the data forwarding function. We design a threshold public key re-encryption scheme to protect the data confidentiality and support the data forwarding function in a confidential way. Again, the partial decryption in this encryption scheme is independently done by each key server. In this chapter, we introduce the notion of a threshold public key re-encryption scheme and present our advanced secure cloud storage system. We also analyze the performance and show that our system is secure.

5.1 Threshold Public Key Re-Encryption Scheme

A threshold public key re-encryption scheme contains two more algorithm, ReEnc and ReKeyGen, than a threshold public key encryption scheme. The

8 algorithms are SetUp, KeyGen, ShareKeyGen, ReEnc, Enc, ShareDec, and Combine. Recall that SetUp generates the public parameters of the whole system, and KeyGen generates a key pair, consisting of a public key PK and a secret key SK, for each user. New algorithm ReKeyGen generates a re-encryption key $\mathsf{RK}_{\mathsf{A}\to\mathsf{B},\mathsf{F}_{\mathsf{id}}}$ from a user A to another user B with respect to a message identifier F_{id}. Each user uses ShareKeyGen to share his secret key into n secret key shares such that any t of them can recover the secret key. Enc encrypts a given message by a public key PK, and outputs a level-0 ciphertext. Another new algorithm ReEnc can re-encrypt A's ciphertext to B's ciphertext by using the corresponding re-encryption key $\mathsf{RK}_{\mathsf{A}\to\mathsf{B},\mathsf{F}_{\mathsf{id}}}$. The re-encrypted ciphertext is a level-1 ciphertext. ShareDec partially decrypts a given ciphertext by a secret key share and outputs a decryption share. Combine takes a set of decryption shares as input and outputs the message if and only if there are at least t decryption shares.

Figure 5.1 illustrates the ReKeyGen algorithm and Figure 5.2 shows the ReEnc algorithm of our proxy re-encryption scheme. Figure 5.3 shows the decryption algorithm of the primitive proxy re-encryption scheme. In our system construction, we modify it into one with threshold decryption function by sharing the secret keys. Because the public key re-encryption scheme is tightly integrated with random erasure codes, we directly present the system construction.

$$PK_A = (g^{a_1}, h^{a_2}), SK_A = (a_1, a_2, a_3)$$
$$PK_B = (g^{b_1}, h^{b_2}), SK_B = (b_1, b_2, b_3)$$
$$RK_{A \to B, F_{ID}} = (h^{b_2(a_1(f(a_3, F_{ID})+e))}, h^{a_1 e})$$

Figure 5.1: The ReKeyGen algorithm of the proxy re-encryption scheme. *User* A *and user* B *have their own key pairs. The* ReEnc *algorithm generates a re-encryption key with respective to the message identifier* $\mathsf{F_{ID}}$.

$$Enc(PK_A, M) = C = (0, g^r, h^{f(a_3, F_{ID})}, M\tilde{e}(g^{a_1}, h^{rf(a_3, F_{ID})}))$$
$$\mathsf{ReEnc} \Downarrow RK_{A \to B, F_{ID}} = (h^{b_2(a_1(f(a_3, F_{ID})+e))}, h^{a_1 e})$$
$$(1, g^r, h^{f(a_3, F_{ID})}, h^{b_2(a_1(f(a_3+F_{ID})+e))}, M\tilde{e}(g^{a_1}, h^{rf(a_3, F_{ID})}) \cdot \tilde{e}(g^r, h^{a_1 e}))$$

Figure 5.2: The ReEnc algorithm of the proxy re-encryption scheme. *A ciphertext under* A*'s key can be re-encrypted to one under* B*'s key by using the* ReEnc *algorithm and the corresponding re-encryption key.*

5.2 System Construction

As described in the system model, there are 4 phases of our storage system. Those 4 phases are presented in details as follows.

System setup. The algorithm $\mathsf{SetUp}(1^\tau)$ generates the system parameters μ. A user uses $\mathsf{KeyGen}(\mu)$ to generate his public and secret key pair and uses $\mathsf{ShareKeyGen}(\mathsf{SK_A}, t, m)$ to share his secret key to a set of m key servers with a threshold t, where $k \leq t \leq m$.

- $\mathsf{SetUp}(1^\lambda)$: Run $\mathsf{Gen}(1^\lambda)$ to obtain $(g, h, \tilde{e}, \mathbb{G}_1, \mathbb{G}_2, p)$, where $\tilde{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ is a bilinear map, g is a generator of $\mathbb{G}_1$, and both $\mathbb{G}_1$ and $\mathbb{G}_2$ have the prime order p. Set $\mu = (g, h, \tilde{e}, \mathbb{G}_1, \mathbb{G}_2, p, f)$ where $h = g^\alpha$, $\alpha \in_R \mathbb{Z}_p$ and $f : \mathbb{Z}_p \times \{0,1\}^* \to \mathbb{Z}_p$ is a one-way hash function.

- $\mathsf{KeyGen}(\mu)$. For a user A, the algorithm selects $a_1, a_2, a_3 \in_R \mathbb{Z}_p$ and sets

$$\mathsf{PK_A} = (g^{a_1}, h^{a_2}), \mathsf{SK_A} = (a_1, a_2, a_3)$$

A decrypts the level 0 ciphertext

$$\frac{M\tilde{e}(g^{a_1}, h^{rf(a_3, F_{ID})})}{\tilde{e}(g^r, h^{f(a_3, F_{ID})})^{a_1}} = M$$

B decrypts the level 1 ciphertext

$$\frac{M\tilde{e}(g, h)^{ra_1(f(a_3+F_{ID})+e)}}{\tilde{e}(g^r, h^{b_2(a_1(f(a_3+F_{ID})+e))})^{b_2^{-1}}} = M$$

Figure 5.3: The Dec algorithm of the proxy re-encryption scheme.
The level-0 ciphertext under A*'s key can be decrypted by using* A*'s secret key* a_1*. The level-1 ciphertext under* B*'s key can be decrypted by using* B*'s secret key* b_2^{-1}*.*

- $\mathsf{ShareKeyGen}(\mathsf{SK_A}, t, m)$. This algorithm shares the secret key $\mathsf{SK_A}$ of a user A to a set of m key servers by using two polynomials $f_{\mathsf{A},1}(\mathrm{z})$ and $f_{\mathsf{A},2}(\mathrm{z})$ of degree $(t-1)$ over the finite field $\mathrm{GF}(p)$.

$$f_{\mathsf{A},1}(\mathrm{z}) = \mathsf{a_1} + \mathsf{v}_1\mathrm{z} + \mathsf{v}_2\mathrm{z}^2 + \cdots + \mathsf{v}_{t-1}\mathrm{z}^{t-1} \pmod{p}$$
$$f_{\mathsf{A},2}(\mathrm{z}) = \mathsf{a}_2^{-1} + \mathsf{v}_1\mathrm{z} + \mathsf{v}_2\mathrm{z}^2 + \cdots + \mathsf{v}_{t-1}\mathrm{z}^{t-1} \pmod{p},$$

  where $\mathsf{v}_1, \mathsf{v}_2, \ldots, \mathsf{v}_{t-1} \in_R \mathbb{Z}_p$. The key share of the secret key $\mathsf{SK_A}$ to the key server KS_i is $\mathsf{SK}_{\mathsf{A},i} = (f_{\mathsf{A},1}(i), f_{\mathsf{A},2}(i))$, where $1 \le i \le m$.

Data storage. When user A wants to store k messages $m_1, m_2, \ldots, m_k$ with the same identifier $\mathsf{F_{id}}$, he computes the encryption token $\tau = h^{f(a_3, \mathsf{F_{id}})}$ and performs the encryption algorithm $\mathsf{Enc}(\cdot)$ on τ and k messages to get the level-0 ciphertexts. Without loss of generality, assume that all k messages are in $\mathbb{G}_2$. A level-0 ciphertext is indicated by a leading bit 0 in it. User A sends each ciphertext to randomly chosen v storage servers. A storage server receives a set of level-0 ciphertexts with the same encryption token τ from A. The cardinality of the set may be less than k. When the storage server does not receive some ciphertext C_i, the storage server inserts $C_i = (0, 1, \tau, 1)$ to

the set. The storage server performs $\mathsf{Encode}(\cdot)$ on the set of k ciphertexts and stores the output.

- $\mathsf{Enc}(\mathsf{PK_A}, \tau, m_1, m_2, \ldots, m_k)$. For $1 \leq i \leq k$, this algorithm computes

$$C_i = (0, \alpha_i, \beta, \gamma_i) = (0, g^{r_i}, \tau, m_i \tilde{e}(g^{a_1}, \tau^{r_i})),$$

  where $r_i \in_R \mathbb{Z}_p, 1 \leq i \leq k$.

- $\mathsf{Encode}(C_1, C_2, \ldots, C_k)$. For each ciphertext C_i, where $1 \leq i \leq k$, the algorithm randomly selects a coefficient g_i. If the input entry for some ciphertext C_i is $(0, 1, \tau, 1)$, the coefficient g_i is set to 0. Let $C_i = (0, \alpha_i, \beta, \gamma_i)$. The encoding process is to compute a ciphertext C':

$$\begin{aligned} C' &= (0, \prod_{i=1}^{k} (\alpha_i^{g_i}), \beta, \prod_{i=1}^{k} (\gamma_i^{g_i})) \\ &= (0, g^{\sum_{i=1}^{k} g_i r_i}, \tau, \prod_{i=1}^{k} m_i^{g_i} \tilde{e}(g, \tau)^{\sum_{i=1}^{k} g_i r_i}) \\ &= (0, g^{r'}, \tau, M \tilde{e}(g, \tau)^{r'}), \end{aligned}$$

  where $M = \prod_{i=1}^{k} m_i^{g_i}$ and $r' = \sum_{i=1}^{k} g_i r_i$. The output is $(C', g_1, g_2, \ldots, g_k)$.

Data forwarding. When user A wants to forward the messages with the identifier $\mathsf{F_{id}}$ to another user B, he computes the re-encryption key $\mathsf{RK_{A \to B, F_{id}}}$ via the $\mathsf{ReKeyGen}(\cdot)$ algorithm and securely sends the re-encryption key to each storage server. A storage server stores $\mathsf{RK_{A \to B, F_{id}}}$ and makes a copy of all messages that have the identifier $\mathsf{F_{id}}$. By using $\mathsf{RK_{A \to B, F_{id}}}$, the storage server re-encrypts the encoded ciphertext C' with the identifier $\mathsf{F_{id}}$ as a re-encrypted

ciphertext C'' via the $\mathsf{ReEnc}(\cdot)$ algorithm such that C'' is decryptable by B's secret key. A re-encrypted ciphertext is indicated by the leading bit 1. Let the public key PK_B of user B be (g^{b_1}, h^{b_2}).

- $\mathsf{ReKeyGen}(\mathsf{PK_A}, \mathsf{SK_A}, \mathsf{F_{id}}, \mathsf{PK_B})$. This algorithm selects $e \in_R \mathbb{Z}_p$ and computes

$$\mathsf{RK}_{\mathsf{A}\to\mathsf{B},\mathsf{F_{id}}} = (\left(h^{b_2}\right)^{a_1(f(a_3,\mathsf{F_{id}})+e)}, h^{a_1 e})$$

- $\mathsf{ReEnc}(\mathsf{RK}_{\mathsf{A}\to\mathsf{B},\mathsf{F_{id}}}, C')$. Let $C' = (0, \alpha, \beta, \gamma) = (0, g^{r'}, \tau, M\tilde{e}(g^{a_1}, \tau^{r'}))$ for some r' and some M, and $\mathsf{RK}_{\mathsf{A}\to\mathsf{B},\mathsf{F_{id}}} = (\left(h^{b_2}\right)^{a_1(f(a_3,\mathsf{F_{id}})+e)}, h^{a_1 e})$ for some e. The level-1 ciphertext is computed as follows:

$$C'' = (1, \alpha, \left(h^{b_2}\right)^{a_1(f(a_3,\mathsf{F_{id}})+e)}, \gamma \cdot \tilde{e}(\alpha, h^{a_1 e}))$$

Data retrieval. There are two cases for the data retrieval phase. The first case is that a user A retrieves his own data. When user A wants to retrieve the k messages with the identifier $\mathsf{F_{id}}$, he informs all key servers with the encryption token τ. A key server first retrieves stored data from u randomly chosen storage servers and then performs the partial decryption $\mathsf{ShareDec}(\cdot)$ on every retrieved level-0 ciphertext C'. The result of the partial decryption is called a decryption share. The key server sends the decryption shares ζ and the coefficients to user A. After user A collects the replies from more than t key servers and there are k out of them originally from distinct storage servers, he executes $\mathsf{Combine}(\cdot)$ on the t decryption shares to recover the messages $m_1, m_2, \ldots, m_k$. The second case is that a user B retrieves

the data that are forwarded. When user B wants to retrieve data that are forwarded to him, he informs all key servers directly. The collection and combining parts are the same as the first case except that key servers retrieve level-1 ciphertexts and perform the partial decryption $\mathsf{ShareDec}(\cdot)$ on the level-1 ciphertexts.

- $\mathsf{ShareDec}(\mathsf{SK_j}, X_i)$. Let $X_i = (\mathsf{b}, \alpha, \beta, \gamma)$ be a level-b ciphertext and $\mathsf{SK_j} = (\mathsf{sk_0}, \mathsf{sk_1})$. By using the key share $\mathsf{SK_j}$, a decryption share $\zeta_{i,j}$ of X_i is generated as follows.

$$\zeta_{i,j} = (\mathsf{b}, \alpha, \beta, \beta^{sk_\mathsf{b}}, \gamma)$$

- $\mathsf{Combine}(\zeta_{i_1,j_1}, \zeta_{i_2,j_2}, \ldots, \zeta_{i_t,j_t})$. Let a decryption share $\zeta_{i,j}$ be $(\mathsf{b}, \alpha_{i,j}, \beta_{i,j}, \beta'_{i,j}, \gamma_{i,j})$. This algorithm combines t decryption shares, where $\beta_{i_1,j_1} = \beta_{i_2,j_2} = \cdots = \beta_{i_t,j_t} = \tau$, $j_1 \neq j_2 \neq \ldots \neq j_t$ and there are at least k distinct values in $\{i_1, i_2, \ldots, i_t\}$. Let $\mathsf{S_J} = \{j_1, j_2, \ldots, j_t\}$ and $\mathsf{S} = \{(i_1, j_1), (i_2, j_2), \ldots, (i_t, j_t)\}$. Without loss of generality, let $\mathsf{S_I} = \{i_1, i_2, \ldots, i_k\}$ be k distinct values in $\{i_1, i_2, \ldots, i_t\}$.

  In the first case, $\mathsf{b} = 0$ and user A wants to retrieve his own data. The algorithm combines the t values $(\beta'_{i_1,j_1}, \beta'_{i_2,j_2}, \ldots, \beta'_{i_t,j_t})$ to obtain $\tau^{a_1} = \tau^{f_{\mathsf{A},1}(0)}$ via the Lagrange interpolation over exponents:

$$\tau^{a_1} = \prod_{(i,j)\in\mathsf{S}} \left((\beta'_{i,j})^{\prod_{r\in\mathsf{S_J}, r\neq j} \frac{-j}{r-j}} \right)$$

  For each of the decryption shares $\zeta_{i,j}$, where $i \in \mathsf{S_I}$, the algorithm

computes an encoded message

$$w_i = \gamma_{i,j} / \tilde{e}(\alpha_{i,j}, \tau^{f_{\mathsf{A},1}(0)}) \tag{5.1}$$

Observe that $w_i = m_1^{g_{1,i}} m_2^{g_{2,i}} \cdots m_k^{g_{k,i}}$ for $i \in \mathsf{S_I}$, and there are k such equations. Consider the square matrix $K = [g_{i,j}]$ where $1 \leq i \leq k, j \in \mathsf{S_I}$. The decoding process is to compute K^{-1} and output the messages $m_1, m_2, \ldots, m_k$. The algorithm fails when the square matrix K is non-invertible.

In the second case, $\mathsf{b} = 1$ and user B wants to retrieve the re-encrypted messages. The algorithm does the following computation to obtain:

$$\begin{aligned} h^{(f(a_3,\mathsf{F_{id}})+e)a_1} &= \prod_{(i,j)\in\mathsf{S}} \left((\beta'_{i,j})^{\prod_{r\in\mathsf{S_J}, r\neq j} \frac{-j}{r-j}} \right) \\ &= h^{(f(a_3,\mathsf{F_{id}})+e)a_1 b_2 f_{\mathsf{B},2}(0)}, \end{aligned}$$

where $f_{\mathsf{B},2}(0) = b_2^{-1}$. Again, for each of $\zeta_{i,j}$, where $i \in \mathsf{S_I}$, the algorithm computes an encoded message.

$$w_i = \gamma_{i,j} / \tilde{e}(\alpha_{i,j}, h^{(f(a_3,\mathsf{F_{id}})+e)a_1}) \tag{5.2}$$

The rest in the second case is the same as that in the first case.

5.2.1 Correctness

There are two cases for correctness. Firstly, the owner A correctly retrieves the messages with an overwhelming probability. The correctness of the en-

cryption and the decryption for user A is that any ciphertext $C' = (0, g^r, \tau, w\tilde{e}(g^{a_1}, \tau^r))$ can be correctly decrypted to w, when there are more than t active key servers who have shares of SK_A. This correctness can be seen from Equation (5.1). The user combines t decryption shares and then correctly gets the encoded message. Secondly, user B correctly retrieves the forwarded messages with an overwhelming probability. The correctness of the re-encryption and the decryption for user B is that any re-encrypted ciphertext $C'' = (1, g^{r'}, \left(h^{b_2}\right)^{r''}, w\tilde{e}(g,h)^{r'r''})$ can be decrypted to w, when there are more than t available key servers who have shares of B's secret key SK_B. The correctness can be seen in Equation (5.2). The user can correctly compute the encoded message from the k decryption shares.

5.3 Analysis

We analyze the complexity of our storage system in terms of the storage cost and the computation cost. We analyze the probability of a successful retrieval and show the security of our cloud storage system.

5.3.1 Performance Analysis

Let the bit-length of the element in the group $\mathbb{G}_1$ be l_1 and $\mathbb{G}_2$ be l_2. Let the coefficients $g_{i,j}$ be randomly chosen from $\{0,1\}^{l_3}$.

Storage cost. We measure the storage cost in a storage server in bit. To store a set of k messages, a storage server SS_j stores a ciphertext $(\mathtt{b}, \alpha_j, \tau, \gamma_j)$ and the coefficient vector $(g_{1,j}, g_{2,j}, \ldots, g_{k,j})$. The total cost in the storage server is $(1 + 2l_1 + l_2 + kl_3)$ bits, where $\alpha_j, \tau \in \mathbb{G}_1$, and $\gamma_j \in \mathbb{G}_2$; hence,

Operation	Computation cost
Enc	k Pairing + k Exp_1 + k Mult_2
Encode (for each storage server)	k Exp_1 + k Exp_2 + $(k-1)$ Mult_1 + $(k-1)$ Mult_2
ShareDec (for t key servers)	t Exp_1
ReEnc (for each storage server)	1 Pairing +1 Mult_2
Combine	k Pairing + t Mult_1 + $(t-1)$ Exp_1 +O(t^2+k^3) F_p+ k^2 Exp_2 + $(k+1)k$ Mult_2

- Pairing: a pairing computation of $\tilde{e}$.
- Exp_1 and Exp_2: a modular exponentiation computation in $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively.
- Mult_1 and Mult_2: a modular multiplication computation in $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively.
- F_p: an arithmetic operation in $GF(p)$.

Table 5.1: Computation cost of each algorithm in our advanced secure cloud storage system.

the average cost for a message bit is $(1 + 2l_1 + l_2 + kl_3)/kl_2$ bits, which is dominated by l_3/l_2 for a sufficient large k. In practice, $g_{i,j}$'s are chosen from a much smaller set, i.e., $l_3 < l_2$. The small coefficients reduce the storage cost in each storage server.

Computation cost. We measure the computation cost for algorithms in each phase in the number of the pairing operations, the modular exponentiations in $\mathbb{G}_1$ and $\mathbb{G}_2$, the modular multiplications in $\mathbb{G}_1$ and $\mathbb{G}_2$, and the arithmetic operations over $GF(p)$. Those operations are denoted as Pairing, Exp_1, Exp_2, Mult_1, Mult_2, and F_p, respectively. The cost is summarized in Table 5.1. F_p has much lower cost than the Mult_1 and Mult_2. One Exp_1 is about

$1.5\lceil \log p \rceil$ Mult_1 on average (by using the square-and-multiply algorithm). Similarly, Exp_2 is about $1.5\lceil \log p \rceil$ Mult_2 on average. $\mathsf{Pairing}$ is considered as a more expensive operation. Some improved algorithms [48, 49, 51] are proposed for accelerating the pairing operation.

In the data storage phase, a user runs the $\mathsf{Enc}(\cdot)$ algorithm and each storage server performs the $\mathsf{Encode}(\cdot)$ algorithm. In the $\mathsf{Enc}(\cdot)$ algorithm, generating each α_i requires one Exp_1, and generating each γ_i requires one Exp_1, one $\mathsf{Pairing}$, and one Mult_2. Hence, for k messages, the cost is (k $\mathsf{Pairing}$ + $2k$ Exp_1 + k Mult_2). For the $\mathsf{Encode}(\cdot)$ algorithm, we use a maximum estimation that each storage server encodes the k ciphertexts. The cost is k $\mathsf{Exp}_1 + (k-1)$ Mult_1 for computing α and k $\mathsf{Exp}_2 + (k-1)$ Mult_2 for computing γ.

In the secure forwarding phase, a user runs $\mathsf{ReKeyGen}(\cdot)$ and each storage server performs $\mathsf{ReEnc}(\cdot)$. In the $\mathsf{ReKeyGen}(\cdot)$ algorithm, the computation is one Exp_1 while in the $\mathsf{ReEnc}(\cdot)$ algorithm, the computation cost is a $\mathsf{Pairing}$ and a Mult_1.

In the data retrieval phase, each key server runs the $\mathsf{ShareDec}(\cdot)$ algorithm and the user performs the $\mathsf{Combine}(\cdot)$ algorithm. In the $\mathsf{ShareDec}(\cdot)$ algorithm, each key server performs one Exp_1 to get $\beta^{sk_{\mathsf{b}}}$ for a level-b ciphertext. For a successful retrieval, t key servers would be sufficient; hence, for this step, we consider the total cost of t key servers. The cost is t Exp_1. In the $\mathsf{Combine}(\cdot)$ algorithm, it includes the computation of the Lagrange interpolation over exponents in $\mathbb{G}_1$, the computation of the encoded messages w_j's from the decryption shares $\tilde{\zeta}_{i,j}$'s, and the decoding computation which includes the matrix inversion and the computation of the messages m_i's from

the encoded messages w_j's. The Lagrange interpolation over exponents in $\mathbb{G}_1$ needs $O(t^2)$ F_p, t Exp_1 , and $(t-1)$ Mult_1. Computing a encoded message w_j needs one $\mathsf{Pairing}$ and one modular division, which takes 2 Mult_2. As for the decoding computation, the matrix inversion takes $O(k^3)$ arithmetic operations over $GF(p)$, and the decoding for each message takes k Exp_2 and $(k-1)$ Mult_2.

5.3.2 Successful Retrieval Probability

Again, we analyze the probability of a successful retrieval event. Here we give our result for the setting in Theorem 4.

Theorem 4. *Assume that there are k messages, n storage servers, and m key servers, where $n = ak^c$, $m \geq t \geq k$, $c \geq 1.5$ and a is a constant with $a > \sqrt{2}$. For $v = bk^{c-1} \ln k$ and $u = 2$ with $b > 5a$, the probability of a successful retrieval is at least $1 - o(1) - k/p$.*

Proof. The methodology of this proof is similar to the one for the first system. However, here we consider a superseded parameter setting for $n = ak^c$, where $c \geq 1.5$. This setting makes the proof different in the probability bounds on every "bad" event.

To retrieve all k messages, there are two conditions that must be satisfied. First, the key servers have to get k stored data from k different storage servers $\mathsf{SS}_{j_1}, \mathsf{SS}_{j_2}, \ldots, \mathsf{SS}_{j_k}$ and perform the $\mathsf{ShareDec}(\cdot)$ algorithm. Second, the $k \times k$ matrix K formed by chosen coefficients needs to be invertible in order to solve the k messages. We define two events E_1 and E_2 to capture the complements of the two conditions such that a successful retrieval happens when neither

E_1 nor E_2 happens. Define E_1 be the event that less than k distinct storage servers are queried by the key servers. For the generator matrix G implicitly generated by the owner and the storage servers, let E_2 be the event that the matrix K is non-invertible. Thus, the probability of a successful retrieval by the owner is

$$1 - \Pr[\mathrm{E}_1] - \Pr[\mathrm{E}_2|\overline{\mathrm{E}}_1]\Pr[\overline{\mathrm{E}}_1] \tag{5.3}$$

To analyze $\Pr[\mathrm{E}_1]$, we consider that each storage server is a bin and each key server has u balls, where $u = 2$. When a key server queries a storage server, we consider that the key server throws a ball into the bin. Because the key servers make queries randomly, those balls are randomly thrown into n bins. The probability that less than k bins contain balls is:

$$\begin{aligned}
\Pr[\mathrm{E}_1] &\leq C_{k-1}^{n} \left(\frac{k-1}{n}\right)^{2m} \\
&\leq \left[\frac{2n(n-k+2)}{k}\right]^{\frac{k-1}{2}} \left(\frac{k-1}{n}\right)^{2m} && (2a^2k^{2c-1} - 2ak^c + 4ak^{c-1} \geq 1, n = ak^c) \\
&\leq \left(2a^2k^{2c-1} - 2ak^c + 4ak^{c-1}\right)^{\frac{k}{2}} \left(\frac{k-1}{ak^c}\right)^{2k} \\
&= \left[\frac{2a^2k^{2c-1} - 2ak^c + 4ak^{c-1}}{(ak^{c-1})^4}\right]^{\frac{k}{2}} \left(\frac{k-1}{k}\right)^{2k} && (\frac{k-1}{k} \leq 1) \\
&= o(1)
\end{aligned} \tag{5.4}$$

The event E_2 under the condition $\overline{\mathrm{E}}_1$ can be modeled by forming a perfect matching in the random bipartite graph H with respect to G. The random bipartite graph H is constructed as follows. Let each ciphertext C_i be a

vertex $v_{1,i}$ and V_1 be the set of all vertices for all ciphertexts. Let each storage server SS_j be a vertex $v_{2,j}$ and V_2 be the set of all vertices for all storage servers. When a ciphertext C_i is distributed to the storage server SS_j, there is an edge $(v_{1,i}, v_{2,j})$. The matrix K induces a subgraph H' of the bipartite graph H. Consider the subset $V_2' \subset V_2$ that V_2' is a subset of queried storage servers with cardinality k. The subgraph H' consists of all vertices in V_1 and V_2', and edges $(v_{1,i}, v_{2,j})$ for all $v_{1,i} \in V_1$ and $v_{2,j} \in V_2'$. If H' has no perfect matching, K is not invertible. If H' has a perfect matching, K is non-invertible if and only if $det(K) = 0$. The value of $det(K)$ depends on the random coefficients chosen by the storage servers. Let E_3 be the event that H' has no perfect matching, and E_4 be the event that $det(K) = 0$. We have,

$$\begin{aligned}\Pr[\mathrm{E}_2|\overline{\mathrm{E}}_1] &= \Pr[\mathrm{E}_3|\overline{\mathrm{E}}_1] + \Pr[\mathrm{E}_4|\overline{\mathrm{E}}_3 \wedge \overline{\mathrm{E}}_1]\Pr[\overline{\mathrm{E}}_3|\overline{\mathrm{E}}_1] \\ &\leq \Pr[\mathrm{E}_3|\overline{\mathrm{E}}_1] + \Pr[\mathrm{E}_4|\overline{\mathrm{E}}_3 \wedge \overline{\mathrm{E}}_1] \qquad (5.5)\end{aligned}$$

We analyze the probability of E_3 conditioned on $\overline{\mathrm{E}}_1$ by using the result of the Hall's Lemma described in Chapter 4. Again, there are two cases that H' has no perfect matching. First, H' has at least one isolated vertex. Second, H' has no isolated vertex and a set A satisfies the above conditions. Let $\mathrm{E}_{\ddot{\mathrm{I}}}$ be the event that H' has at least one isolated vertex and E_{A} be the event that there is a set A satisfying the conditions. We obtain

$$\Pr[\mathrm{E}_3|\overline{\mathrm{E}}_1] \leq \Pr[\mathrm{E}_{\ddot{\mathrm{I}}}|\overline{\mathrm{E}}_1] + \Pr[\mathrm{E}_{\mathrm{A}}|\overline{\mathrm{E}}_1] \qquad (5.6)$$

Starting from $\mathrm{E}_{\ddot{\mathrm{I}}}$, we consider each vertex in V_2 as a bin and each edge

from V_1 to V_2 as a ball. When an edge connects to a vertex in V_2, a ball is thrown into the bin. Consider the subset B of bins corresponding to the subset V_2' of V_2. $\mathrm{E}_{\ddot{\mathrm{I}}}$ means that there is one or more empty bins in B, where B contains k bins. For a fixed bin in B, the probability of the bin being empty is $(1-1/n)^{bk^c \ln k}$ since there are $bk^c \ln k$ balls. By using the union bound on k bins, we have the probability of $\mathrm{E}_{\ddot{\mathrm{I}}}$ conditioned on $\overline{\mathrm{E}}_1$ as:

$$\begin{aligned}
\Pr[\mathrm{E}_{\ddot{\mathrm{I}}}|\overline{\mathrm{E}}_1] &\le k(1-1/n)^{bk^c \ln k} \\
&\le k(e^{-\frac{b}{a}\ln k}) \qquad (1-x \le e^{-x}, n = ak^c) \\
&= o(1) \qquad (b > 5a) \qquad (5.7)
\end{aligned}$$

As for $\Pr[\mathrm{E_A}|\overline{\mathrm{E}}_1]$, we separate the event into two sub-events by $\mathrm{A} \subset V_1$ and $\mathrm{A} \subset V_2'$. Thus,

$$\begin{aligned}
\Pr[\mathrm{E_A}|\overline{\mathrm{E}}_1] &= \Pr[\mathrm{E_A \text{ and } A} \subset V_1|\overline{\mathrm{E}}_1]\Pr[\mathrm{A} \subset V_1] + \Pr[\mathrm{E_A \text{ and } A} \subset V_2'|\overline{\mathrm{E}}_1]\Pr[\mathrm{A} \subset V_2'] \\
&\le \Pr[\mathrm{E_A \text{ and } A} \subset V_1|\overline{\mathrm{E}}_1] + \Pr[\mathrm{E_A \text{ and } A} \subset V_2'|\overline{\mathrm{E}}_1]
\end{aligned}$$

For $\Pr[\mathrm{E_A \text{ and } A} \subset V_2'|\overline{\mathrm{E}}_1]$, we further divide the event into sub-events according to the size of A and use the union bound again. Consider a set $A \subset V_2'$ with $|A| = i$. The event $\mathrm{E_A}$ conditioned on $\overline{\mathrm{E}}_1$ can be overestimated by the event that $\Gamma(A) \subset V_1$ and $|\Gamma(A)| = i-1$. In other words, there is a set $A' \subset V_1$ with $|A'| = i-1$ such that all vertices in $V_1 \backslash A'$ only connect to

vertices in $V_2 \backslash A$. Thus, we have

$$
\begin{aligned}
&\Pr[\mathrm{E_A} \text{ and } \mathrm{A} \subset V_2' | \overline{\mathrm{E}}_1] \\
&\leq \sum_{i=2}^{(k+1)/2} \Pr[\mathrm{E_A}, \mathrm{A} \subset V_2' \text{ and } |A| = i | \overline{\mathrm{E}}_1] \\
&\leq \sum_{i=2}^{(k+1)/(2)} (\frac{ek}{i})^{2i} (\frac{n-i}{n})^{(k-i+1)bk^{c-1}\ln k} \qquad (C_i^k \leq (\frac{ek}{i})^i) \\
&\leq \max_i \{\exp\left(2i(1-\ln i) - \ln 2 + \ln k \left[b(k-i+1)k^{c-1} \ln\left(\frac{n-i}{n}\right) + 2i + 1\right]\right)\}
\end{aligned}
\tag{5.8}
$$

To achieve $\Pr[\mathrm{E_A} \text{ and } \mathrm{A} \subset V_2' | \overline{\mathrm{E}}_1] = o(1)$ as $k \to \infty$, it is sufficient to have that for all $2 \leq i \leq (k+1)/2$

$$
b(k-i+1)k^{c-1} \ln\left(\frac{n-i}{n}\right) + 2i + 1 < 0 \tag{5.9}
$$

Equation 5.9 holds for $b > 5a$. It implies that $\Pr[\mathrm{E_A} \text{ and } \mathrm{A} \subset V_2' | \overline{\mathrm{E}}_1] = o(1)$ as $k \to \infty$.

Similarly, we can get a lower bound for b from the case of $A \subset V_1$ and the bound is satisfied by $b > 5a$.

For $\Pr[\mathrm{E}_4 | \overline{\mathrm{E}}_3 \wedge \overline{\mathrm{E}}_1]$, that is, $det(A) = 0$, we treat each coefficient in the matrix K as a variable. Thus, $det(K)$ is a multivariate function. Since there is a perfect matching in the induced graph H', $det(K)$ is a non-zero function and the degree of $det(K)$ is k. From the Schwartz-Zeppel Theorem, the probability that the random chosen coefficients make $det(K) = 0$ is no more than k/p, i.e., $\Pr[\mathrm{E}_4 | \overline{\mathrm{E}}_3 \wedge \overline{\mathrm{E}}_1] \leq k/p$. Thus, we have $\Pr[\mathrm{E}_2 | \overline{\mathrm{E}}_1] \leq o(1) + k/p$ and conclude the proof of Theorem 4. □

5.3.3 Security Analysis

The data confidentiality of our secure cloud storage system is guaranteed even if all storage servers and up to $(t-1)$ key servers are compromised by the attacker. Recall the security game illustrated in Figure 3.5. We prove that our cloud storage system is secure under the decisional bilinear Diffie-Hellman assumption in Theorem 5.

Theorem 5. *Our constructed cloud storage is secure under the decisional bilinear Diffie-Hellman assumption in the standard model.*

Proof. We prove by contradiction. Assume an attacker $\mathcal{A}$ wins the security game with probability $1/2+\epsilon$, an algorithm $\mathcal{S}$ can solve the decisional Diffie-Hellman problem with advantage $\frac{\epsilon}{2P(\lambda)}$ in polynomial time, where $P(\lambda)$ is the number of users.

The algorithm $\mathcal{S}$ takes $(\tilde{e}, g, g^x, g^y, g^z, \mathbb{Q})$ and the corresponding bilinear map $\tilde{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ as an instance of the problem input and aims to decide whether $\mathbb{Q} = \tilde{e}(g,g)^{xyz}$. $\mathcal{S}$ runs the following phases to simulate the environment for $\mathcal{A}$:

- *Setup.* $\mathcal{S}$ selects a one-way keyed hash function $f : \mathbb{Z}_p \times \{0,1\}^* \rightarrow \mathbb{Z}_p$ and sets $\mu = (g, h, \tilde{e}, \mathbb{G}_1, \mathbb{G}_2, p, f)$ where $h = g^y$ and p is the prime order of the group $\mathbb{G}_1$ and $\mathbb{G}_2$. $\mathcal{S}$ predicts the target user T and sets $\mathsf{PK}_\mathrm{T} = (g^x, h^{t_2})$, and $\mathsf{SK}_\mathrm{T} = (x, t_2, t_3)$, where $t_2, t_3 \in_R \mathbb{Z}_p$ and x is implicit set ($\mathcal{S}$ does not know x). If $\mathcal{A}$ does not choose the user T, $\mathcal{S}$ terminates the simulation and outputs a random value; otherwise, $\mathcal{S}$ continues this simulation. $\mathcal{S}$ creates a polynomial number of users and generates their keys. $\mathcal{S}$ sends μ and all public keys to $\mathcal{A}$.

- *Key query.* Firstly, $\mathcal{S}$ generates random values for the secret key shares $\mathsf{SK}_{\mathrm{T},i}$, for $1 \leq i \leq (t-1)$. For the secret key queries, $\mathcal{S}$ generates the key pairs as the scheme defines and replies the secret key directly. For the re-encryption key queries, there are two cases: a re-encryption key from a non-target user to T; a re-encryption key from a non-target user to another non-target user. For the first case, $\mathcal{S}$ generates the re-encryption key from user B to T, where the public key of B is (g^{b_1}, h^{b_2}):

$$\mathsf{RK}_{\mathsf{B}\to\mathrm{T},\mathsf{F_{id}}} = (h^{t_2 b_1 (f(a_3,\mathsf{F_{id}})+e)}, h^{b_1 e})$$

  For the second case, since $\mathcal{S}$ knows all secret keys of the non-target users, the re-encryption key can be generated as the scheme defines.

- *Challenge.* $\mathcal{A}$ decides the messages M_0, M_1 and the class identities $\mathsf{F_{id_0}}, \mathsf{F_{id_1}}$ and sends them to $\mathcal{S}$. $\mathcal{S}$ first throws a random coin $b \in \{0,1\}$ and then computes the encryption token and the ciphertext of M_b under T's key.

$$\tau_b = h^{f(a_3,\mathsf{F_{id_b}})}, \mathsf{Enc}(\mathsf{PK_T}, \tau_b, M_b) = (0, g^z, \tau_b, M_b \mathbb{Q}^{f(a_3,\mathsf{F_{id_b}})})$$

  $\mathcal{S}$ sends $\mathsf{Enc}(\mathsf{PK_T}, \tau_b, M_b)$ to the attacker $\mathcal{A}$.

- *Output.* After $\mathcal{A}$ outputs b', $\mathcal{S}$ outputs 0 if $b' = b$; otherwise, $\mathcal{S}$ outputs 1.

When $\mathbb{Q} = \mathbb{Q}_0 = \tilde{e}(g,g)^{xyz}$, the ciphertext is an encryption of M_b since

$$M_b \tilde{e}(g^x, h^{z f(a_3,\mathsf{F_{id_b}})}) = M_b \tilde{e}(g,g)^{xyz f(a_3,\mathsf{F_{id_b}})} = M_b \mathbb{Q}^{f(a_3,\mathsf{F_{id_b}})}$$

Hence, $\mathcal{A}$ has an advantage ϵ winning the game, i.e., $\Pr[b' = b|\mathbb{Q} = \tilde{e}(g,g)^{xyz}] = 1/2 + \epsilon$. When $\mathbb{Q} = \mathbb{Q}_1 = \tilde{e}(g,g)^r$ for some random value r, the distributions of $(g^z, h^{f(a_3,\mathsf{F}_{\mathsf{id}_0})}, M_0\mathbb{Q}^{f(a_3,\mathsf{F}_{\mathsf{id}_0})})$ and $(g^z, h^{f(a_3,\mathsf{F}_{\mathsf{id}_1})}, M_1\mathbb{Q}^{f(a_3,\mathsf{F}_{\mathsf{id}_1})})$ are identical because for any $r \in \mathbb{Z}_p$, there exists a unique $r' \in \mathbb{Z}_p$ such that

$$M_0\tilde{e}(g,g)^{rf(a_3,\mathsf{F}_{\mathsf{id}_0})} = M_1\tilde{e}(g,g)^{r'f(a_3,\mathsf{F}_{\mathsf{id}_1})}$$

Thus, we have $\Pr[b' = b|\mathbb{Q} = \tilde{e}(g,g)^r] = 1/2$. Let b" be the output bit of $\mathcal{S}$. The advantage of $\mathcal{S}$ is:

$$\begin{aligned}
&|\Pr[\mathcal{S} \to b", \mathbb{Q} = \mathbb{Q}_{b"}, T \neq T'] + \Pr[\mathcal{S} \to b", \mathbb{Q} = \mathbb{Q}_{b"}, T = T'] - \frac{1}{2}| \\
&= |\Pr[T \neq T'] \Pr[\mathcal{S} \to b", \mathbb{Q} = \mathbb{Q}_{b"} | T \neq T'] \\
&\quad + \Pr[T = T', \mathbb{Q} = \mathbb{Q}_0] \Pr[\mathcal{S} \to 0 | \mathbb{Q} = \mathbb{Q}_0, T = T'] \\
&\quad + \Pr[T = T', \mathbb{Q} = \mathbb{Q}_1] \Pr[\mathcal{S} \to 1 | \mathbb{Q} = \mathbb{Q}_1, T = T'] - \frac{1}{2}| \\
&= |(1 - \frac{1}{P(\lambda)}) \times \frac{1}{2} + \frac{1}{2P(\lambda)} \times (\frac{1}{2} + \epsilon) + \frac{1}{2P(\lambda)} \times \frac{1}{2} - \frac{1}{2}| \\
&= \frac{\epsilon}{2P(\lambda)}
\end{aligned}$$

Since ϵ is non-negligible, $\mathcal{S}$ solves the decisional bilinear Diffie-Hellman problem with a non-negligible advantage $\frac{\epsilon}{2P(\lambda)}$ in polynomial time. It makes a contradiction. □

Chapter 6

Discussion

In this chapter, we discuss the security features that our cloud storage systems have and the data integrity issue that is not addresses in our current storage systems. We provide current results on the data integrity checking and discuss the potential method to modify our systems to support these requirements. We also discuss two features the cloud has and how our systems fit into the cloud environment.

6.1 Security Features of Our Storage Systems

Both of our storage systems have security features: data confidentiality, data availability (robustness), and decentralized storage control. It is possible to improve our systems in many ways.

6.1.1 Data Confidentiality

In our storage systems, each storage server stores data that are encrypted by the described public key encryption schemes. Even if all storage servers collude together, the data content is kept secret from all storage servers. Different from other storage systems that employ the encryption at rest (i.e. the storage servers know the decryption keys), our storage systems have the strong data confidentiality against collusion of all storage servers. However, the encryption schemes we used only achieve the security against the chosen plaintext attacks. In the cryptography community, a stronger property called "secure against the chosen ciphertext attacks" is preferred. Designing a chosen ciphertext secure public key encryption scheme which is both homomorphic and threshold decryptable is a potential method. In particular, fully homomophic encryption schemes [52, 53] are proposed recently. Modifying one of them as a chosen ciphertext secure version may work.

6.1.2 Data Availability

Our storage systems use a variant of the random erasure codes for data availability. The data are available as long as there are k storage servers that store codeword symbols (in the encrypted form). The coding parameters u and v are already explored and analyzed in our research. However, we did not include a repair mechanism for machine failures in our storage systems. In general, once the system is aware of failure of some storage server, a repair process should be executed. A trivial method is to re-distribute all data that the failed storage server stores. Retrieving the data then storing again is too

expensive and it requires the involvement of the data owner. Better repair mechanism could be designed. In another direction, we did not address the general errors which include random noises and intentional data alteration. Public-key locally-decodable codes [54] provide a possible way to not only tolerate but also correct errors when data are encrypted. Fully homomorphic encryption schemes [55, 56, 53] enable encoding and decoding when the encoding and decoding operations are multiplications and additions.

6.1.3 Decentralized Storage Control

The storing process and retrieving process are performed independently among all storage servers and key servers in our storage systems. The distributed storage for the decryption key shares provide a good structure against corrupted key servers as long as the number of corrupted key servers is less than the threshold t. When the key servers are distributed over many networks and guarded by different strong security mechanisms, the chance that an attacker breaks more than t key servers is small.

6.2 Data Integrity

Our cloud storage systems address the data confidentiality against collusion of all storage servers because it is unrealistic to fully trust all of the storage servers. Similarly, the data integrity issue arises because the storage servers may compromise their promise on the data availability. Any accidental break or intentional alteration could cause permanent data loss or data errors.

Juels and Kaliski mentioned the concept of "proof of retrievability" (POR)

for the remotely stored large files [57]. The key idea of their method is as follows. Before a user stores a file into the system, he divides the file into blocks and inserts some checking blocks called *sentinel*s. The user employs an error correction code and a pseudo random permutation on the file blocks and sentinels. The result of the permutation is then sent to the storage system. Later, the user can ask a subset of random (sentinel) blocks to see if the data are stored. Because the storage server can not distinguish the file blocks and the sentinel blocks, he must keep all of them or he may be caught with an overwhelming probability. The resulting proof information has a length linear in the number of selected blocks.

At the same time as Juels and Kaliski, Ateniese et al. [58] also proposed the concept of "provable data possession" (PDP) that allows the storage servers proving that the data are completely and correctly stored. They use a quite different approach from the method in [57]. The main idea of their method is the use of a homomorphic signature scheme. After a user divides the file into blocks, he generates a signature on each file block. The signature is considered as a tag for the file block. The storage server keeps the file blocks with the tags. When the user wants to check the data integrity, he randomly selects c indices of the file blocks and c random values as the challenge message. The server then linearly combines the chosen file blocks and aggregates the corresponding signatures via the indicated coefficients. The result of the combination and the aggregated signature are verified by the user. The homomorphic property of the signature scheme contributes to the "aggregation" operation and saves the communication cost. The resulting proof information has a length independent of how many data are checked.

After the work in [57], Ateniese et al. proposed another token-based solution for the provable data possession protocol [59]. They use random tokens (like the sentinels) as checking points and randomly permute the tokens and the file blocks such that the storage server cannot distinguish them. The resulting PDP protocol supports dynamic data structure. Any modification operation is allowed in their storage system. The robust data checking protocol [60] encodes the file blocks by using an error correction code before proceeding any PDP protocols.

Athos [61] authenticates the outsource file system while the system supports data dynamic. The main idea is to represent the file blocks by using a special data structure, skip list. An improved data structure, ranked-based skip list, is used in the dynamic PDP [62]. Similarly, another approach [63] that makes a dynamic PDP system uses the Merkle tree structure.

On the other hand, many PDP or POR protocols [64, 65, 66] take the homomorphic signature or homomorphic hash approach. Shacham and Waters [65] propose a homomorphic signature based construction by using a bilinear map. They left an open question for a POR system that is provable secure without random oracle model. Later, Dodis et al. [66] solve the open problem by using an abstract concept "homomorphic linear authenticator scheme". Taking one step further, Ateniese et al. [67] show that a "proof of storage" system can be constructed by using any homomorphic linear authenticator scheme. A concrete construction based on the hardness of the factorization problem is given in their work. A variant of PDP protocol in [64] considers a better data availability guarantee. The storage system should prove not only the data possession but also the replica possession.

Because replication is a common fault tolerant method, the storage system should guarantee that the replicas are completely stored.

All constructions we described above allow the user checking the data integrity by him. Some of them using public key based schemes allow a third party checking the data integrity for the data owner. Wang [68] et al. bring a privacy preserving property for the public auditing function. In their construction, a third party auditor can check the data integrity for the data owner while he gets no data content during the auditing process.

HAIL [69] is a PDP scheme for distributed storage systems. Bowers et al. consider a cloud storage system where each storage server only stores several codeword symbols and provide HAIL for the integrity checking function. They use a keyed hash function for the codeword integrity checking, where the hash function is homomorphic on the same key.

6.3 Supporting Integrity Checking

One trivial way for providing the integrity checking in our storage systems is that treating the ciphertexts as RAW data and use the homomorphic signature based PDP schemes. Before a user stores the ciphertexts into the system, he also generates the signature for each ciphertext. Later, he or any one possessing a verification key can verify the data integrity. This trivial construction achieves the privacy-preserving property directly because the checked data are ciphertexts. However, this method is neither sophisticated nor efficient. We leave the integrity checking function as a future work.

6.4 Cloud Features

Two major features a cloud has is *virtualization* and *data migration*. An end-user of a cloud server never needs to know the physical structure of the used cloud. He only has the virtualized view of the whole system. For example, a user uses a web-mail service. He only has the view of the inbox and never knows where those emails are and how they are stored. As for the data migration, it happens when the system provider wants to change the number of the online servers. For example, when a server is overloaded, some of the running tasks will be transferred to some other server. The data used by those transferred tasks must be moved. Hence, the data migration happens.

In our cloud storage systems, the data distribution process can be done by an agent in the cloud. As a result, the user only has to upload the ciphertexts and the agent will perform the next steps for the user. Similarly, when the user wants to retrieve the stored data, he can ask some agent in the cloud to collect the decryption shares for him. As a result, the user has no idea about the structure of the cloud storage system and can still use the storage service.

The data migration is not covered in our cloud storage systems so far. In our cloud storage systems, each storage server stores the data in the encrypted and encoded form. When the data migration happens, how to re-distribute those data among storage servers such that the data can be retrieved is an interesting issue for our future work.

Chapter 7

Summary and Future Work

We have presented our integration of public key encryptions schemes and random erasure codes and introduced our secure cloud storage systems. Our systems provide both of the storage service and the key management service. Both system constructions are fully decentralized: each encrypted message is distributed independently; each storage server performs the encoding process in a decentralized way; each key server queries the storage servers independently. Moreover, the secure distributed networked storage system guarantees the privacy of messages even if all storage servers are compromised. Our storage systems securely store data for a long period of time on un-trusted storage servers in the distributed network structure. The advance storage system additionally supports the data forwarding in a confidential way.

We explored the relationship between the number of storage servers n, the number of messages k, the number of key servers m, the number of storage servers a key server queries u, and the number of message copies v and presented our suggestions for them:

1. $n = ak^{3/2}$, $a > \sqrt{2}$, $m \geq t \geq k > 1$, $v = bk^{1/2} \ln k$, $u = 2$ with $b > 5a$

2. $n = ak^{c}$, $a > \sqrt{2}$, $c \geq 1.5$, $m \geq t \geq k > 1$, $v = bk^{c-1} \ln k$, $u = 2$ with $b > 5a$

3. $n = ak$, $a > 1$, $m = t = k > 1$, $v = b_1 \ln k$, $u = b_2 \ln k$ with $b_1 > 5a$ and $b_2 > 4 + 3/\ln a$

Actually, the result of setting 2 supersedes the result of setting 1. As a result, the setting 2 is suitable for more applications with different system scales.

The constructions of secure cloud storage systems justify my dissertation statement.

A cryptographic secure cloud storage system built on a decentralized architecture provides strong data confidentiality against collusion of all storage servers and is robust with low storage overhead. Furthermore, the system is adaptable to allow data forwarding inside the cloud in a confidential way.

In our current secure cloud storage systems, certain security features can be improved and the data integrity checking function is required as we discussed in previous chapter. We believe that these requirements are strongly needed for a cloud storage system. Further study on supporting these requirements is interesting. We summarize them as follows:

- *Data Confidentiality.* The confidentiality degree can be upgraded via designing a chosen ciphertext secure public key encryption scheme which allows encoding operation over ciphertexts and parallel threshold decryption.

- *Data Robustness.* Not only erasure errors, cloud storage systems should also handle alteration errors. Error detection and correction mechanisms are required.

- *Data Integrity.* As we discussed in previous chapter, many results on the remote data integrity are proposed. It is desired to have one solution for data stored in the encrypted form.

- *Access Control.* In our current advanced system, the messages are simply gathered as groups and a user can forward messages group by group. However, the organization of the storage space may be more complicated and users need a finer-grant control of forwarding functionality. For example, the user may want to attach certain attributes to messages when he stores the messages. Later, he can forward messages with certain attribute values.

- *Data Dynamics.* When we design our storage systems, data archive is our implicit target application. However, a cloud storage system should provide more. After the data are encrypted, supporting data dynamic requires a sophisticated solution.

Bibliography

[1] Google. A webmail system. http://mail.google.com.

[2] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles - SOSP 2003*, pages 29–43. ACM, 2003.

[3] David R. Brownbridge, Lindsay F. Marshall, and Brian Randell. The newcastle connection or unixes of the world unite! *Software Practice and Experience*, 12(12):1147–1162, 1982.

[4] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon. Design and implementation of the sun network filesystem, 1985.

[5] Frank Gens. IT Cloud Services User Survey, pt.2: Top Benefits and Challenges. http://blogs.idc.com/ie/?p=210, October 2008.

[6] Matt Blaze. A cryptographic file system for unix. In *Proceedings of the ACM Conference on Computer and Communications Security - CCS 1993*, pages 9–16. ACM, 1993.

[7] Andrew D. McDonald and Markus G. Kuhn. Stegfs: A steganographic file system for linux. In *Proceedings of the 3rd International Workshop*

on on Information Hiding - IH 1999, volume 1768 of *Lecture Notes in Computer Science*, pages 462–477. Springer, 1999.

[8] Giuseppe Cattaneo, Luigi Catuogno, Aniello Del Sorbo, and Pino Persiano. The design and implementation of a transparent cryptographic file system for unix. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, pages 199–212. USENIX Association, 2001.

[9] Charles P. Wright, Michael C. Martino, and Erez Zadok. Ncryptfs: A secure and convenient cryptographic file system. In *Proceedings of the General Track: 2003 USENIX Annual Technical Conference*, pages 197–210. USENIX, 2003.

[10] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation - OSDI 2002*, pages 1–14, 2002.

[11] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, Mahadev Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, 1988.

[12] Mahadev Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Steere. Coda: A highly available

file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, 1990.

[13] David Mazières, Michael Kaminsky, M. Frans Kaashoek, and Emmett Witchel. Separating key management from file system security. In *Proceedings of the ACM Symposium on Operating Systems Principles - SOSP 1999*, pages 124–139. ACM, 1999.

[14] Mahesh Kallahalla, Erok Riedel, Ram Swaminathan, Qian Wang, and Kevin Fu. Plutus: Scalable secure file sharing on untrusted storage. In *Proceeding of the 2nd Conference on File and Storage Technologies - FAST 2003*. USENIX, 2003.

[15] Joseph A. Cooley, Jeremy L. Mineweaser, Leslie D. Servi, and Eushiuan T. Tsung. Software-based erasure codes for scalable distributed storage. In *IEEE Symposium on Mass Storage Systems*, pages 157–164, 2003.

[16] Noga Alon, Haim Kaplan, Michael Krivelevich, Dahlia Malkhi, and Julien P. Stern. Scalable secure storage when half the system is faulty. *Information and Computation.*, 174(2):203–213, 2002.

[17] Mario Blaum, Jim Brady, Jehoshua Bruck, and Jai Menon. Evenodd: An efficient scheme for tolerating double disk failures in raid architectures. *IEEE Transactions on Computers*, 44(2):192–202, 1995.

[18] Chih-Shing Tau and Tzone-I Wang. An alternative decoding algorithm for evenodd code in raid architectures. In *Proceedings of the 21st In-*

ternational Multi-Conference on Applied Informatics - AI 2003, pages 887–892. IASTED/ACTA Press, 2003.

[19] Cheng Huang and Lihao Xu. Star : An efficient coding scheme for correcting triple storage node failures. *IEEE Transactions on Computers*, 57:889–901, 2007.

[20] Michael G. Luby, Michael Mitzenmacher, M. Amin Shokrollahi, Daniel A. Spielman, and Volker Stemann. Practical loss-resilient codes. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing - STOC 1997*, pages 150–159. ACM, 1997.

[21] Michael Luby. Tornado codes: Practical erasure codes based on random irregular graphs. In *Proceedings of Randomization and Approximation Techniques in Computer Science, Second International Workshop - RANDOM 1998*, volume 1518 of *Lecture Notes in Computer Science*, page 171. Springer, 1998.

[22] Matthew Woitaszek and Henry M. Tufo. Tornado codes for maid archival storage. *Mass Storage Systems and Technologies, IEEE / NASA Goddard Conference on*, 0:221–226, 2007.

[23] James S. Plank and Michael G. Thomason. A practical analysis of low-density parity-check erasure codes for wide-area storage applications. pages 115–124, 2004.

[24] Peter Druschel and Antony I. T. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 8th Workshop*

on Hot Topics in Operating Systems - HotOS-VIII 2001, pages 75–80. IEEE Computer Society, 2001.

[25] Andreas Haeberlen, Alan Mislove, and Peter Druschel. Glacier: Highly durable, decentralized storage despite massive correlated failures. In *Proceedings on the 2nd Symposium on Networked Systems Design and Implementation - NSDI 2005*. USENIX, 2005.

[26] Byung-Gon Chun, Frank Dabek, Andreas Haeberlen, Emil Sit, Hakim Weatherspoon, M. Frans Kaashoek, John Kubiatowicz, and Robert Morris. Efficient replica maintenance for distributed storage systems. In *Proceedings on the 3rd Symposium on Networked Systems Design and Implementation - NSDI 2006*. USENIX, 2006.

[27] Patrick R. Eaton, Hakim Weatherspoon, and John Kubiatowicz. Efficiently binding data to owners in distributed content-addressable storage systems. In *Proceedings of the 3rd International IEEE Security in Storage Workshop - SISW 2005*, pages 40–51. IEEE Computer Society, 2005.

[28] Jeremy Stribling, Jinyang Li, Isaac G. Councill, M. Frans Kaashoek, and Robert Morris. Overcite: A distributed, cooperative citeseer. In *Proceedings on the 3rd Symposium on Networked Systems Design and Implementation - NSDI 2006*. USENIX, 2006.

[29] Emil Sit, Robert Morris, and M. Frans Kaashoek. Usenetdht: A low-overhead design for usenet. pages 133–146, 2008.

[30] Emil Sit Andreas, Andreas Haeberlen, Frank Dabek, Byung gon Chun, Hakim Weatherspoon, Robert Morris, M. Frans Kaashoek, and John Kubiatowicz. Proactive replication for data durability. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems - IPTPS 2006*, 2006.

[31] Alessandro Duminuco, Ernst Biersack, and Taoufik En-Najjary. Proactive replication in distributed storage systems using machine availability estimation. In *Proceedings of the ACM Conference on Emerging Network Experiment and Technology - CoNEXT 2007*, page 27. ACM, 2007.

[32] Szymon Acedanski, Supratim Deb, Muriel Medard, and Ralf Koetter. How good is random linear coding based distributed network storage. In *Proceeding of IEEE International Symposium on Network Coding - NetCod 2005*, 2005.

[33] Alexandros G. Dimakis, Vinod Prabhakaran, and Kannan Ramchandran. Ubiquitous access to distributed data in large-scale sensor networks through decentralized erasure codes. In *Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks - IPSN 2005*, pages 111–117. IEEE Computer Society, 2005.

[34] Alexandros G. Dimakis, Vinod Prabhakaran, and Kannan Ramchandran. Decentralized erasure codes for distributed networked storage. *IEEE/ACM Transactions on Networking*, 14:2809–2816, 2006.

[35] Salah A. Aly, Zhenning Kong, and Emina Soljanin. Fountain codes based distributed storage algorithms for large-scale wireless sensor net-

works. In *Proceedings of the 7th International Conference on Information Processing in Sensor Networks - IPSN 2008*, pages 171–182. IEEE Computer Society, 2008.

[36] Zooko Wilcox-O'Hearn and Brian Warner. Tahoe: the least-authority filesystem. In *Proceedings of the 4th ACM international workshop on Storage security and survivability - StorageSS 2008*, pages 21–26. ACM, 2008.

[37] Salah A. Aly, Zhenning Kong, and Emina Soljanin. Raptor codes based distributed storage algorithms for wireless sensor networks. *CoRR*, abs/0903.0445, 2009.

[38] Zhenning Kong, Salah A. Aly, and Emina Soljanin. Decentralized coding algorithms for distributed storage in wireless sensor networks. *IEEE Journal on Selected Areas in Communications*, 28(2):261–267, 2010.

[39] Amin Shokrollahi. Raptor codes. *IEEE/ACM Transactions on Networking*, 14(SI):2551–2567, 2006.

[40] Dejan Vukobratovic, Čedomir Stefanovic, Miloš Stojakovic, and Vladimir Stankovic. Raptor packets: a packet-centric approach to distributed raptor code design. In *Proceedings of the 2009 IEEE international conference on Symposium on Information Theory - ISIT'09*, pages 2336–2340. IEEE Press, 2009.

[41] John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon,

Westly Weimer, Christopher Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the 9th international Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS 2000*, pages 190–201. ACM, 2000.

[42] Ranjita Bhagwan Kiran, Kiran Tati, Yu chung Cheng, Stefan Savage, and Geoffrey M. Voelker. Total recall: System support for automated availability management. In *Proceedings on the 31st Symposium on Networked Systems Design and Implementation - NSDI 2003*, pages 337–350, 2004.

[43] Charles Blake and Rodrigo Rodrigues. High availability, scalable storage, dynamic peer networks: Pick two. In *HotOS*, pages 1–6. USENIX, 2003.

[44] Rodrigo Rodrigues and Barbara Liskov. High availability in dhts: Erasure coding vs. replication. In *Proceedings on the 4th International Workshop on Peer-to-Peer Systems IV - IPTPS 2005*, volume 3640 of *Lecture Notes in Computer Science*, pages 226–239. Springer, 2005.

[45] Chris Williams, Philippe Huibonhoa, JoAnne Holliday, Andy Hospodor, and Thomas Schwarz. Redundancy management for p2p storage. In *Proceedings on the IEEE International Symposium on Cluster Computing and the Grid*, pages 15–22. IEEE Computer Society, 2007.

[46] Ranjita Bhagwan, Stefan Savage, and Geoffrey M. Voelker. Understanding availability. In *Proceedings of the 2nd International Workshop on*

Peer-to-Peer Systems, volume 2735 of *Lecture Notes in Computer Science*, pages 256–267. Springer Berlin / Heidelberg, 2003.

[47] Server Protectors LLC. Secure cloud storage. http://serverprotectors.com/solutions/secs.

[48] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Efficient implementation of pairing-based cryptosystems. *Journal of Cryptology*, 17(4):321–334, 2004.

[49] Victor S. Miller. The weil pairing, and its efficient calculation. *Journal of Cryptology*, 17(4):235–261, 2004.

[50] Rajeev Motwani and Prabhakar Raghavan. Cambridge University Press, 1995.

[51] Piotr Szczechowiak, Anton Kargl, Michael Scott, and Martin Collier. On the application of pairing based cryptography to wireless sensor networks. In *Proceedings of the Second ACM Conference on Wireless Network Security - WISEC 2009*, pages 1–12. ACM, 2009.

[52] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM symposium on Theory of computing - STOC 2009*, pages 169–178, 2009.

[53] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. Cryptology ePrint Archive, Report 2009/616, 2009.

[54] Brett Hemenway and Rafail Ostrovsky. Public-key locally-decodable codes. In *Proceedings of the 28th Annual International Cryptology Conference on Advances in Cryptology - CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 126–143. Springer, 2008.

[55] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM symposium on Theory of computing - STOC 2009*, pages 169–178. ACM, 2009.

[56] N.P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. Cryptology ePrint Archive, Report 2009/571, 2009.

[57] Ari Juels and Burton S. Kaliski Jr. Pors: Proofs of retrievability for large files. pages 584–597, 2007.

[58] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM Conference on Computer and communications security - CCS 2007*, pages 598–609. ACM, 2007.

[59] 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 2008*, pages 1–10. ACM, 2008.

[60] Reza Curtmola, Osama Khan, and Randal Burns. Robust remote data checking. In *Proceedings of the 4th International Workshop on Storage Security and Survivability - StorageSS 2008*, 2008.

[61] Michael T. Goodrich, Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Athos: Efficient authentication of outsourced file systems. In *Proceedings on the 11th International Conference on Information Security - ISC 2008*, volume 5222 of *Lecture Notes in Computer Science*, pages 80–96. Springer, 2008.

[62] C. Christopher Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession. In *ACM Conference on Computer and Communications Security*, pages 213–222. ACM, 2009.

[63] Qian Wang, Cong Wang, Jin Li, Kui Ren, and Wenjing Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *Proceedings of the 14th European Symposium on Research in Computer Security - ESORICS 2009*, volume 5789 of *Lecture Notes in Computer Science*, pages 355–370. Springer, 2009.

[64] Reza Curtmola, Osama Khan, Randal Burns, and Giuseppe Ateniese. Mr-pdp: Multiple-replica provable data possession. In *Proceedings of International Conference on Distributed Computing Systems*, pages 411–420. IEEE Computer Society, 2008.

[65] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *Proceedings of the 14th International Conference on the Theory*

and Application of Cryptology and Information Security - ASIACRYPT 2008, volume 5350 of *Lecture Notes in Computer Science*, pages 90–107. Springer, 2008.

[66] Yevgeniy Dodis, Salil P. Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification. In *Proceeding of the 6th Theory of Cryptography Conference on Theory of Cryptography - TCC 2009*, volume 5444 of *Lecture Notes in Computer Science*, pages 109–127. Springer, 2009.

[67] Giuseppe Ateniese, Seny Kamara, and Jonathan Katz. Proofs of storage from homomorphic identification protocols. In *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security- ASIACRYPT 2009,*, volume 5912 of *Lecture Notes in Computer Science*, pages 319–333. Springer, 2009.

[68] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for data storage security in cloud computing. In *Proceedings of the 28th IEEE International Conference on Computer Communications - INFOCOM 2009*. IEEE, 2010.

[69] Kevin D. Bowers, Ari Juels, and Alina Oprea. Hail: a high-availability and integrity layer for cloud storage. In *Proceedings of the 2009 ACM Conference on Computer and Communications Security - CCS 2009*, pages 187–198. ACM, 2009.