Replication, Security, and Integrity of Outsourced Data in Cloud Computing Systems

by

Ayad Fekry Barsoum

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2013

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In the current era of digital world, the amount of sensitive data produced by many organizations is outpacing their storage ability. The management of such huge amount of data is quite expensive due to the requirements of high storage capacity and qualified personnel. Storage-as-a-Service (SaaS) offered by cloud service providers (CSPs) is a *paid* facility that enables organizations to outsource their data to be stored on remote servers. Thus, SaaS reduces the maintenance cost and mitigates the burden of large local data storage at the organization's end.

For an increased level of scalability, availability and durability, some customers may want their data to be replicated on multiple servers across multiple data centers. The more copies the CSP is asked to store, the more fees the customers are charged. Therefore, customers need to have a strong guarantee that the CSP is storing *all* data copies that are agreed upon in the service contract, and these copies remain intact.

In this thesis we address the problem of creating multiple copies of a data file and verifying those copies stored on untrusted cloud servers. We propose a pairing-based provable multi-copy data possession (PB-PMDP) scheme, which provides an evidence that all outsourced copies are actually stored and remain intact. Moreover, it allows authorized users (*i.e.*, those who have the right to access the owner's file) to seamlessly access the file copies stored by the CSP, and supports public verifiability.

We then direct our study to the dynamic behavior of outsourced data, where the data owner is capable of not only archiving and accessing the data copies stored by the CSP, but also updating and scaling (using block operations: modification, insertion, deletion, and append) these copies on the remote servers. We propose a new map-based provable multi-copy dynamic data possession (MB-PMDDP) scheme that verifies the intactness and consistency of outsourced dynamic multiple data copies. To the best of our knowledge, the proposed scheme is the first to verify the integrity of multiple copies of dynamic data over untrusted cloud servers.

As a complementary line of research, we consider protecting the CSP from a dishonest owner, who attempts to get illegal compensations by falsely claiming data corruption over cloud servers. We propose a new cloud-based storage scheme that allows the data owner to benefit from the facilities offered by the CSP and enables mutual trust between them. In addition, the proposed scheme ensures that authorized users receive the latest version of the outsourced data, and enables the owner to grant or revoke access to the data stored by cloud servers.

Acknowledgements

All praise and glory is due to God for blessing, leading, and strengthening me every single moment of my life. God is always here for me whenever I needed help and guidance.

My deepest gratitude to my supervisor, Prof. Anwar Hasan whose ceaseless support, encouragement, and flexibility I shall never forget. Prof. Hasan has guided me throughout the years of my graduate studies. The cornerstone to complete this thesis was my regular meetings with Prof. Hasan to discuss and review my work several times to be in the absolutely best form we can see. I do appreciate his generosity for making time for my questions on any matter. His internal peace and quiet made my life easier during the hard times I passed through. Prof. Hasan was always encouraging me and expecting more from me, in every step of my work. As I proceed in my life, Prof. Hasan will always be my mentor and a model of how a successful supervisor should be.

I am deeply grateful to my thesis committee members, Prof. Guang Gong, Prof. Mahesh Tripunitara, Prof. Ken Salem, and Prof. Rei Safavi-Naini for their invaluable time, feedback and suggestions. Special thanks to Dr. Sanjit Chatterjee for useful discussions during early stage of my work. I am indebted to my friends Tamer Abdel-Kader and Mohamed Elsalih for their great support and guidance through my first days in Waterloo. I also thank my labmates Marwa and Abdulaziz for their help in my first course.

Thanks are not enough to be given to my parents, brothers, and sisters for their ongoing and endless love. There are no words that are valuable to equalize their love for me.

This thesis would not have been possible without the support of my lovely wife, Evon. She has sacrificed a lot to make me a successful person. My wife has given me everything without expecting anything in return. She has been there for me every step on the way. Without her love and continuous support, I would not have been able to achieve any success in my life. She is always my first resort whenever I am tired or frustrated. I enjoyed sharing every moment with her, and I hope I can be a source of her happiness at all times. Such a wonderful wife beside me is a precious gift from God.

Evon, your love and encouragement have been and will always be an endless source of inspiration in my life.

To Evon, my beloved wife, and Joyce, my little angel

Contents

List of Tables

List of Figures

List of Acronyms

AaaS	Application-as-a-Service
AES	Advanced Encryption Standard
CDH	Computational Diffie-Hellman
CSP	Cloud Service Provider
DL	Discrete Logarithm
HLA	Homomorphic Linear Authenticator
IaaS	Infrastructure-as-a-Service
MAC	Message Authentication Code
MB-PMDDP	Map-Based Provable Multi-Copy Dynamic Data Possession
MR-PDP	Multiple-Replica Provable Data Possession
MVT	Map-Version Table
OSMR	One-Sender-Multiple-Receiver
PaaS	Platform-as-a-Service
PB-PMDP	Pairing-Based Provable Multi-Copy Data Possession
PDDP	Provable Dynamic Data Possession
PDP	Provable Data Possession
POR	Proof of Retrievability

PRF	Pseudo-Random Function
PRP	Pseudo-Random Permutation
SaaS	Storage-as-a-Service
TB-PMDDP	Tree-Based Provable Multi-Copy Dynamic Data Possession
TTP	Trusted Third Party

Chapter 1

Introduction

This chapter briefly describes the cloud computing paradigm, which is an emerging computing model over a shared pool of resources (Section 1.1). It presents a number of key advantages offered by cloud computing compared with traditional means of local computing (Section 1.2). It also summarizes the major challenges facing cloud computing from being widely deployed and used (Section 1.3). In addition, this chapter describes our motivation and research problem (Section 1.4). At the conclusion, it outlines our main contributions (Section 1.5) and thesis organization (Section 1.6).

1.1 Overview of Cloud Computing

Cloud computing is a distributed computational model over a large pool of shared-virtualized computing resources (*e.g.*, storage, processing power, memory, applications, services, and network bandwidth), where customers are provisioned and de-provisioned recourses as they need. Cloud computing represents a vision of providing computing services as public utilities like water and electricity. The architecture of cloud computing can be split in two: front-end and back-end. The front-end represents cloud customers, organizations, or applications (*e.g.*, web browsers) that use the cloud services. The back-end is a huge network of data centers with many different applications, system programs, and data storage systems. It is metaphorically believed that, cloud service providers (CSPs) have almost *infinite* computation power and storage capacity. A conceptual framework of cloud computing architecture is illustrated in Figure 1.1 with its two main parts.

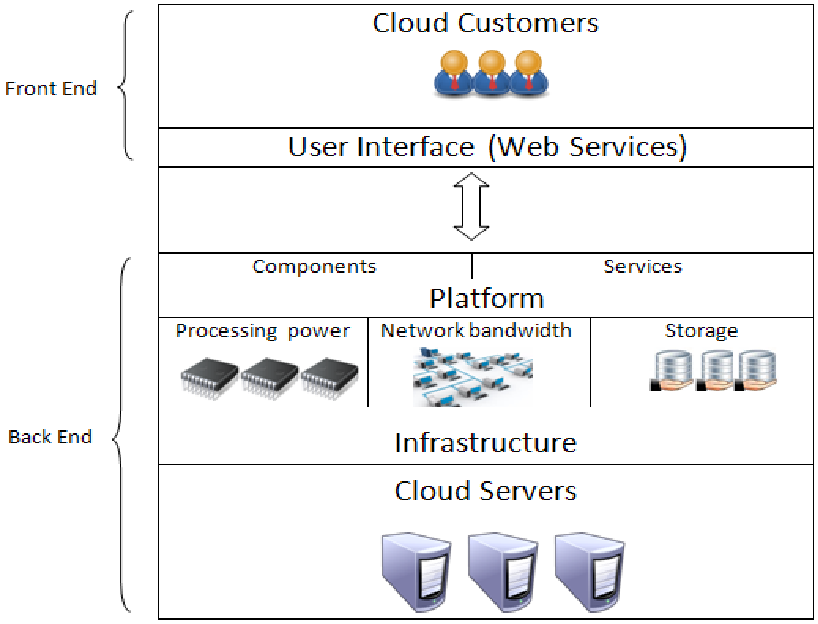


Figure 1.1: Conceptual framework for Cloud Computing architecture.

Cloud computing services can be categorized into [64]:

- Application-as-a-Service (AaaS).
- Platform-as-a-Service (PaaS).
- Infrastructure-as-a-Service (IaaS).

The widely used model of cloud computing services is the AaaS model, in which the customers have access to the applications running on the cloud provider's infrastructure. Google Docs, Google Calendar, and Zoho Writer are known examples of this model. In the PaaS model, the customers can deploy their applications on the provider's infrastructure under condition that these applications are created using tools supported by the provider. The cloud service provider (CSP) hosts a set of software and development tools on its servers to be used by the developers to create their own applications. Google Apps is one of the best known PaaS models. IaaS model enables customers to rent and use the provider's resources (storage, processing, and network). Hence, the customers can deploy any applications including operating systems.

The cloud computing architecture can be deployed under different models [64]:

- *Public cloud.* The infrastructure of the CSP is publicly accessible by general customers and organizations in exchange for pre-specified fees according to the usage of the CSP's services.
- *Private cloud.* The cloud infrastructure is dedicated to an organization which may manage the infrastructure or leave this management to a third party.
- *Hybrid cloud.* The cloud infrastructure is composed of two or more clouds (private or public). The organizations provide and handle some internal and external resources. For example, an organization can use a public cloud service as Amazon Elastic Compute Cloud (Amazon EC2) [5] to perform the general computation, while the data files are stored within the organization's local data center in a private cloud.

1.2 Cloud Computing Characteristics

The considerable attention of cloud computing paradigm is due to a number of key advantages, which make it an interesting research area in both academia and industry. The following are some of the essential characteristics of cloud computing paradigm:

- Supplies cost-effective means of doing business over a shared pool of resources, where users can avoid capital expenditure on hardware, software, and services as *they pay only for what they use* [64].

- Provides low management overhead and immediate access to a wide range of applications.
- Reduces maintenance cost as a third party is responsible for everything from running the cloud to storing data.
- Supports flexibility to scale up and down information technology (IT) capacity over time to business needs.
- Offers more mobility where customers can access information wherever they are, rather than having to remain at their desks.
- Allows organizations to store more data on remote servers than on private computer systems. Organizations will no longer be worried about constant server updates and other computing issues [88].

There are a variety of areas where cloud computing has a significant role: virtual worlds which require excessive amount of computing powers, e-bushiness where scalability can be achieved by assigning new servers as needed, social network, and searching. Figure 1.2 represents a survey made by International Data Corporation (IDC) [55] to indicate why customers value cloud computing paradigm as a new approach of doing business. Cost effectiveness and easiness of deployment are among the main benefits that customers believe they can gain from moving to cloud computing as a new attitude of IT architecture.

1.3 Cloud Computing Challenges

Cloud computing has received considerable attention from research communities in academia as well as industry; however, there are many challenges facing cloud computing to be widely deployed and used. The major challenge is security, which is related to *infrastructure* and *data*. A recently conducted survey about the challenges of cloud computing has indicated that security represents 87.5% of users' cloud fears [55]. Among the other challenges that may hinder the broad use of cloud computing are [33]:

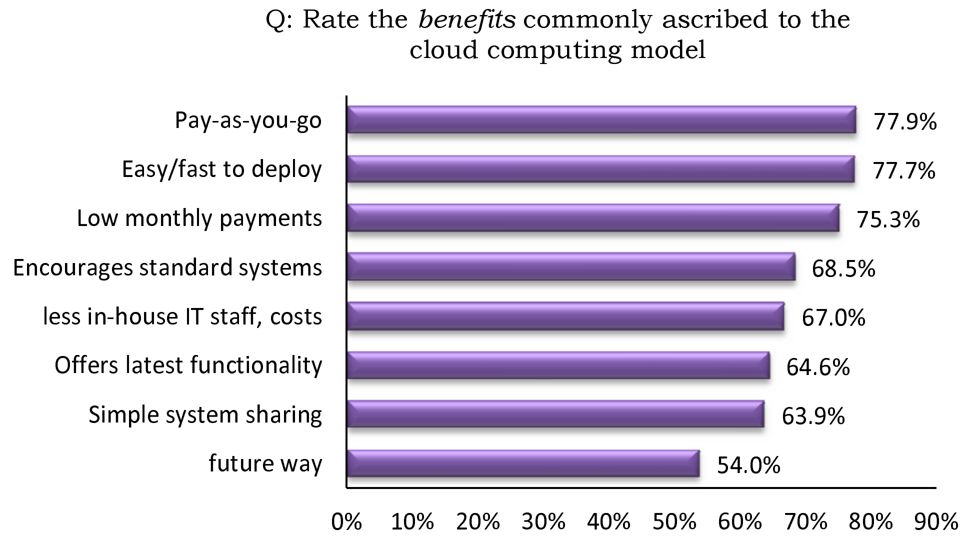


Figure 1.2: Benefits commonly ascribed to Cloud Computing [55].

- **Availability**. Cloud computing model encourages single points of failure where cloud services are subject to more attacks. Among the well-publicized incidents of cloud outages are Gmail (one-day outage) [43] and Amazon Simple Storage Service (Amazon S3), which was down for over 7 hours [51]. Therefore, it is of significant importance to develop new methods and techniques for sustained availability and speedy recovery from attacks.

- **Computational Integrity**. Outsourcing computation is a growing trend for resource-constrained clients to benefit from powerful cloud servers. The ability to verify outsourced computations and validate the returned results is a key requirement of cloud customers. Another imperative point is that the amount of work performed by the clients to verify the outsourced computations must be substantially cheaper than performing the actual computations on the client side.

- **Authentication**. The development of cloud computing encourages the use of resource-constrained devices (*e.g.*, PDA and cell phones) on the client side. Rather than data storage and software installation on local devices, users will authenticate in order to be able to access the data and use cloud applications. This computing model makes software piracy more difficult and enables centralized monitoring. Although cloud computing architecture stimulates mobility of users, it increases the need of secure authentication. User authentication based on passwords in not an efficient approach for sensitive data/applications on the cloud. The use of passwords is a major point of vulnerability in computer security, as passwords are often easy to guess by automated programs running dictionary attacks [32]. Moreover, users cannot remember very long passwords, and usually they use some meaningful passwords making them subject to dictionary attacks.

- **Auditing**. The internal operations of the CSP are opaque, and thus the auditing process is a major challenge. Customers with constrained computing resources and capabilities resort to external audit party to check the integrity of their outsourced data. They need to assure that there is no information leakage even by this third party. Third party auditing process should bring in no new vulnerabilities towards the privacy of client's data

Figure 1.3 represents the results of a survey made to indicate the challenges ascribed to cloud computing model, and the different percentages of users' cloud fears [55].

1.4 Motivation and Research Problem

1.4.1 Research Motivation

In our current digital world, various organizations produce a large amount of *sensitive* data including personal information, electronic health records, and financial data. The amount of digital data is increasing at a staggering rate; doubling almost every year and a half [85], and outpacing the storage ability of many organizations. This data often needs to be stored at multiple locations for a long time due to operational purposes and regulatory compliance. The local management of such huge amount of data is problematic and costly

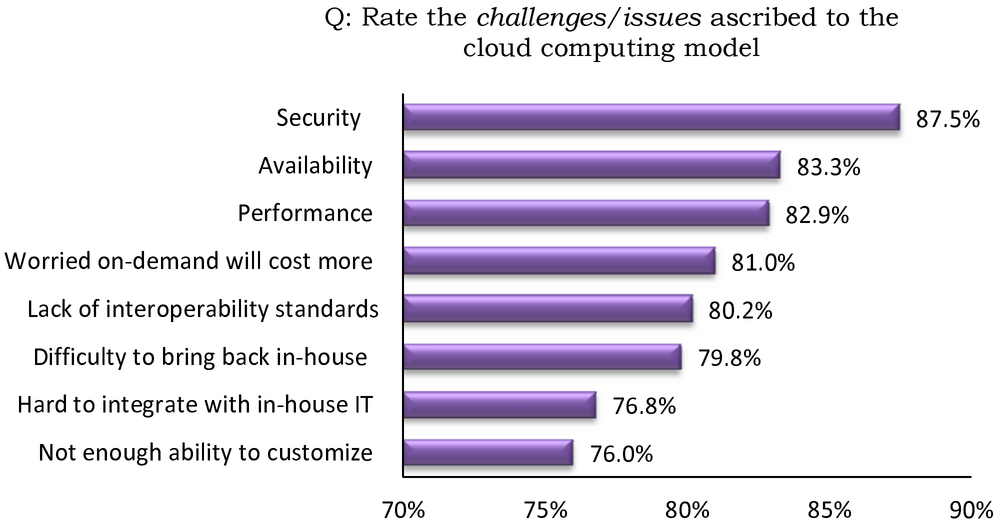


Figure 1.3: Challenges commonly ascribed to Cloud Computing [55].

due to the requirements of high storage capacity and qualified personnel. While there is a steady drop in the cost of storage hardware, the management of storage has become more complex and represents approximately 75% of the total ownership cost [85]. Storage-as-a-Service (sort of IaaS) offered by CSPs is an emerging solution to mitigate the burden of large local data storage and reduce the maintenance cost by means of outsourcing data storage.

Through outsourcing data storage scenario, organizations delegate the storage and management of their data to a CSP in exchange for pre-specified fees metered in GB/month. Such outsourcing of data storage enables organizations to store more data on remote servers than on private computer systems. In addition, some organizations may create large data files that must be archived for many years but are rarely accessed, and thus there is no need to store such files on the local storage of the organizations. More importantly, the CSP often provides better disaster recovery by replicating the data on multiple servers across multiple data centers achieving a higher level of availability. Therefore, many authorized users are allowed to access the remotely stored data from different geographic locations making it more convenient for them. A relatively recent survey indicates that IT outsourcing has grown by a staggering 79% as organizations seek to focus more on their core competencies and reduce costs [89].

However, the fact that data owners no longer physically possess their sensitive data raises new challenges to the tasks of data confidentiality and integrity in cloud computing systems. Unauthorized access and misuse of customers' confidential data are serious concerns regarding data outsourcing; hence, it is of significant importance to be aware of data administrators (CSPs) and their extend of data access right. In some practical applications, data confidentiality is not only a privacy concern, but also a juristic issue. For example, in e-Health applications inside the USA the usage and exposure of protected health information should meet the policies admitted by Health Insurance Portability and Accountability Act (HIPAA) [2], and thus keeping the data private on the remote storage servers is not just an option, but a demand. The confidentiality feature can be guaranteed by the owner via encrypting the data before outsourcing to remote servers. As such, it is a crucial demand of customers to have a strong evidence that the cloud servers still possess their data and it is not being tampered with or partially deleted over time, especially because the internal operation details of the CSP may not be known to cloud customers.

The completeness and correctness of customers' data in the cloud may be at risk due to the following reasons. First, the CSP – whose goal is to make a profit and maintain a reputation – has an incentive to hide data loss (due to hardware failure, management errors, various attacks) or reclaim storage by discarding data that has not been or is rarely accessed. Second, a dishonest CSP might delete some of the data or might not store all data in a high performance storage required by the contract with certain customers, *i.e.*, place it on low cost (and hence slow) media. Third, the cloud infrastructures are subject to a wide range of internal and external security threats. Incidences of security breaches of cloud services surface from time to time [51, 58]. In short, although outsourcing data to the cloud is attractive from the view point of cost and complexity of long-term large-scale data storage, it does not offer sufficient guarantee on data integrity. This problem, if not properly handled, may hinder the successful deployment and wide acceptance of the cloud paradigm.

Once customers' data has been outsourced to remote servers, efficient verification of the completeness and correctness of the outsourced data becomes a formidable challenge. Traditional cryptographic primitives for data integrity and availability based on hashing and signature schemes are not applicable to outsourced data without having a local copy. It is impractical for the owners to download all stored data to validate its integrity; this would require an expensive I/O operations and immense communication overheads across the network. Therefore, efficient techniques are needed to verify the integrity of outsourced data with reduced communication, computation, and storage overheads. Consequently, many researchers have focused on the problem of provable data possession (PDP), and proposed different schemes to audit the data on remote storage sites (PDP will be discussed in more details in Chapter 2).

The main focus of the most work done in the PDP area is to verify the integrity of a *single* outsourced data copy. A small number of researchers have addressed the integrity verification of multiple data copies stored over remote servers. In addition, protecting the CSP from a dishonest owner – who attempts to get illegal compensations by falsely claiming data corruption over cloud servers – is an imperative concern to be addressed. This concern, if not properly handled, can cause the CSP to go out of business [73].

1.4.2 Research Problem

For an increased level of scalability, availability and durability, some customers may want their data to be replicated on multiple servers across multiple data centers. Data replication varies according to the nature of data; more copies are needed for critical data that cannot easily be reproduced, while non-critical, reproducible data are stored at reduced levels of redundancy. The more copies the CSP is asked to store, the more fees the customers are charged. For example, Amazon S3 maintains more copies of customers' data than that of Amazon Reduced Redundancy Storage (Amazon RRS), which enables customers to reduce their costs. The pricing for Amazon S3 is approximately 40% higher than that of Amazon RRS [6] (see Table 1.1). Therefore, customers need to have a strong guarantee that the CSP is storing all data copies that are agreed upon in the service contract and all these copies remain intact.

One of the core design principles of outsourcing data is to provide dynamic scalability of data for various applications. This means that the remotely stored data can be not only accessed by authorized users (*i.e.*, those who have the right to access the owner's file), but also updated and scaled by the data owner. Thus, there must be a guarantee that all outsourced copies are consistent with the most recent modifications issued by the owner.

Another important issue is that the CSP needs to be safeguarded from any false accusation that may be claimed by a data owner to get illegal compensations. Moreover, authorized users have to receive the latest version of the outsourced dynamic data, and there must be a mechanism to grant or revoke access to the outsourced data.

In this thesis, we study the problem of creating multiple copies of a data file and verifying those copies stored on untrusted cloud servers. In addition, we address the integrity verification for multiple copies of dynamic data, where the data owner issues *block-level* dynamic requests to update the data on the CSP side. To complement our research, we consider achieving mutual trust between the data owner and the CSP, where the owner is enabled to utilize the facilities offered by the CSP, and release concerns regarding confidentiality, integrity, and access control of the outsourced data. Meanwhile, a dishonest owner is not able to falsely accuse the CSP and claim data corruption over cloud servers to get illegal compensations.

Table 1.1: Amazon storage pricing [6]

	Standard Storage	Reduced Redundancy Storage
Tier	Pricing	Pricing
First 1 TB/month	$0.125/GB	$0.093/GB
Next 49 TB/month	$0.110/GB	$0.083/GB
Next 450 TB/month	$0.095/GB	$0.073/GB
Next 500 TB/month	$0.090/GB	$0.063/GB
Next 4000 TB/month	$0.080/GB	$0.053/GB
Over 5000 TB/month	$0.055/GB	$0.037/GB

1.5 Thesis Contributions

In this section we present a summary of our major contributions. The work done in this thesis has contributions in three main directions related to outsourcing data storage to remote cloud servers: multiple static data copies, multiple dynamic data copies, and mutual trust for cloud-based storage systems.

1.5.1 Multiple Static Data Copies

- We propose a pairing-based provable multi-copy data possession (PB-PMDP) scheme. This scheme provides an adequate guarantee that the CSP stores all copies that are agreed upon in the service contract, and these copies are intact. The authorized users can seamlessly access the copies received from the CSP. The PB-PMDP scheme supports public verifiability, *i.e.*, anyone who knows the owner's public key can challenge the remote server and verify that the server is still possessing the owner's files.

- We justify the performance of the proposed PB-PMDP scheme through theoretical analysis, experimental results on a commercial cloud platform, and comparison with the multiple-replica provable data possession (MR-PDP) scheme due to Curtmola *et al.* [37].

- We show the security of the PB-PMDP scheme against colluding servers. In addition, we discuss a slight modification of the proposed scheme to identify corrupted copies.

1.5.2 Multiple Dynamic Data Copies

- We propose a map-based provable multi-copy dynamic data possession (MB-PMDDP) scheme. The MB-PMDDP scheme supports outsourcing of dynamic data, *i.e.*, it supports block-level operations such as block modification, insertion, deletion, and append. Moreover, it ensures that all outsourced copies are consistent with the most recent modifications issued by the data owner. To the best of our knowledge, the

MB-PMDDP scheme is the first to address the integrity verification of multiple copies of dynamic data over untrusted cloud servers.

- We give a thorough comparison of MB-PMDDP with a reference scheme, which one can obtain by extending existing PDP models for dynamic single-copy data. We also report our implementation and experiments using Amazon cloud platform.
- We prove the security of the MB-PMDDP scheme against colluding servers depending on the security of the computational Diffie-Hellman and the discrete logarithm problems.

1.5.3 Mutual Trust for Cloud-Based Storage Systems

- We design and implement a cloud-based storage scheme that has the following features:
    - Allows a data owner to outsource the data to a remote CSP, and perform full dynamic operations at the block-level (block modification, insertion, deletion, and append)
    - Ensures the newness property, *i.e.*, the authorized users receive the most recent version of the outsourced data
    - Establishes *indirect* mutual trust between the data owner and the CSP since each party resides in a different trust domain
    - Enforces the access control for the outsourced data
- We discuss the security features of the proposed scheme. Besides, we justify its performance through theoretical analysis and a prototype implementation on Amazon cloud platform to evaluate storage, communication, and computation overheads.

1.6 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 contains a literature survey for different PDP schemes, the rationale behind these schemes, their features and limitations. We start with PDP schemes for static data, then we direct our survey to models that deal with dynamic data. We also highlight the concept of proof of retrievability, which is a complementary approach to PDP.

In Chapter 3, we consider the integrity verification for multiple data copies, for which we start with a basic provable multi-copy data possession scheme followed by a review of the MR-PDP scheme due to Curtmola *et al.* [37]. This chapter also describes our proposed PB-PMDP scheme, gives the system model and assumptions, and presents the security analysis and the performance evaluation for the PB-PMDP scheme. In addition, a slight modification of the proposed PB-PMDP scheme is discussed to identify the indices of corrupted copies.

Our proposed MB-PMDDP scheme to verify the integrity verification of multiple copies of dynamic data is elaborated in Chapter 4. Moreover, we present an extension to dynamic single-copy PDP models to work in the setting of dynamic multiple data copies. This extension servers as a reference model for comparison with the proposed MB-PMDDP scheme. The performance analysis of the MB-PMDDP scheme – validated through experimental results – and the security proof against colluding servers are also presented.

Chapter 5 describes our proposed cloud-based storage scheme to achieve mutual trust between a data owner and a CSP. We present related work and review some techniques pertaining to the construction of our proposed scheme. We detail the performance analysis and the prototype implementation on Amazon EC2 and Amazon S3. In addition, we investigate the security of the proposed scheme by analyzing its fulfillment of the assigned security requirements, namely, confidentiality, integrity, newness, access control, and CSPs defence.

Finally, Chapter 6 summarizes our conclusions and gives a number of directions for future work.

Chapter 2

Review of Provable Data Possession

In this chapter we explain the concept of provable data possession (PDP) (Section 2.1). We present a review for different PDP schemes for static data (Section 2.2). We then direct our survey to PDP models that deal with dynamic data (Section 2.3). We also highlight the concept of proof of retrievability, which is a complementary approach to PDP (Section 2.4). We finally present a short summary of this chapter (Section 2.5).

2.1 Introduction

Provable data possession (PDP) is a technique that allows an entity to prove that the data is in its possession for validating data integrity over remote servers. In a typical PDP model, the data owner generates some metadata/information for a data file to be used later for verification purposes through a *challenge-response* protocol with the remote/cloud server. The owner sends the file to be stored on a remote server which may be untrusted, and deletes the local copy of the file. As a proof that the server is still possessing the data file in its original form, it needs to correctly compute a response to a challenge vector sent from a verifier – who can be the original data owner or a trusted entity that shares some information with the owner. Shortly, PDP schemes allow a verifier to efficiently, periodically, and securely validate that a remote server – which supposedly stores the owner's potentially very large amount of data – is actually storing the data intact.

The problem of data integrity over remote servers has been addressed for many years and there is a simple solution to tackle this problem as follows. The data owner computes a message authentication code (MAC) of the whole file before outsourcing to a remote server. The owner keeps only the computed MAC on his local storage, sends the file to the remote server, and deletes the local copy of the file. Later, whenever a verifier needs to check the data integrity, he sends a request to retrieve the file from the archive service provider, re-computes the MAC of the whole file, and compares the re-computed MAC with the previously stored value. Alternatively, instead of computing and storing the MAC of the whole file, the data owner divides the file F into blocks $\{b_1, b_2, \ldots, b_m\}$, computes a MAC σ_j for each block b_j: $\sigma_j = \text{MAC}_{sk}(j||b_j)_{1\leq j\leq m}$, sends both the data file F and the MACs $\{\sigma_j\}_{1\leq j\leq m}$ to the remote/cloud server, deletes the local copy of the file, and stores only the secret key *sk*. During the verification process, the verifier requests for a set of randomly selected blocks and their corresponding MACs, re-computes the MAC of each retrieved block using sk, and compares the re-computed MACs with the received values from the remote server [92]. The rationale behind the second approach is that checking part of the file is much easier than the whole of it. However both approaches suffer from a severe drawback; the communication complexity is linear with the queried data size which is impractical especially when the available bandwidth is limited.

2.2 Provable Static Data Possession

In this section, we review different PDP schemes for *static* data. We provide the rationale behind these schemes, their features and limitations. We then give a comparison between the presented schemes from different perspectives.

2.2.1 PDP Schemes of Deswarte *et al.*

MAC-based approaches for remote data integrity are associated with high communication overhead. Deswarte *et al.* [39] thought of a technique to reduce the communication cost by using two functions f and H', where H' is a one-way function and f is another function.

The relation between H' and f is that $f(C, H'(File)) = h(C||File)$, where h is any secure hash function and C is a random challenge number sent from the verifier to the remote server. Thus, the data owner has to compute $H'(File)$ and store it on his local storage. To audit the file, the verifier generates a random challenge C, computes $V = f(C, H'(File))$, and sends C to the remote server. Upon receiving the challenge C, the server computes $S = h(C||File)$ and sends the response S to the verifier. To validate the file integrity, the verifier checks $V \overset{?}{=} S$. At least one of the two functions f and H' must be kept secret because if both were public, it would be easy for a malicious server to compute and store only $H'(File)$ that is not the entire file, and then dynamically responds with a valid value $f(C, H'(File))$ that is not the expected one $h(C||File)$.

Unfortunately, Deswarte *et al.* [39] have not found such functions f, H', and h satisfying the desired verification rule. To workaround this problem, a finite number $\widetilde{N}$ of random challenges are generated offline for the file to be checked, and the corresponding responses $h(C_i||File)_{1 \leq i \leq \widetilde{N}}$ are pre-computed and stored on the verifier local storage. To audit the file, one of the $\widetilde{N}$ challenges is sent to the remote server and the received response is compared with the pre-computed one (previously stored on the verifier side). However, this solution limits the number of times a particular data file can be checked by the number of random challenges $\widetilde{N}$. Once all random challenges $\{C_i\}_{1 \leq i \leq \widetilde{N}}$ are consumed, the verifier has to retrieve the data file from the storage server to compute new responses, but this is unworkable.

Deswarte *et al.* [39] provided another protocol to overcome the problem of limited number of audits per file. In this protocol the data file is represented as an integer d. Figure 2.1 illustrates the scheme presented in [39].

The main limitation in the protocol of Deswarte *et al.* [39] is the computation overhead on the server side. In each verification, the remote server has to do the exponentiation over the entire file. Thus, if we are dealing with huge files, *e.g.*, in order of Terabytes (as most practical applications require) this exponentiation will be heavy. The data owner can reduce the exponent part in the computation $M = a^d \bmod N$ by utilizing the Fermat-Euler theorem [59], where $a^d \equiv a^{d \bmod \phi(N)} \bmod N$ and $\phi(N) = (p-1)(q-1)$ is the Euler's totient function. The remote server cannot use this trick because $\phi(N)$ is not known in public.

Data owner:

- Represents the data file as an integer d
- Generates RSA modulus $N = pq$ (p & q are prime numbers)
- Pre-computes and stores $M = a^d \bmod N$ ($a \in_R \mathbb{Z}_N$)
- Sends the file value d to the remote server

Challenge Response

Verifier | Remote Server

1. Picks $r \in_R \mathbb{Z}_N$
2. Computes a challenge $A = a^r \bmod N$

$\xrightarrow{A}$

3. Computes a response $S = A^d \bmod N$

$\xleftarrow{B}$

4. Computes $V = M^r \bmod N$
5. Checks $V \overset{?}{=} S$

Figure 2.1: The PDP protocol by Deswarte *et al.* [39].

2.2.2 More RSA-Based PDP Schemes

Filho *et al.* [48] proposed a scheme to verify data integrity using the RSA-based homomorphic hash function. A function $\widehat{H}$ is homomorphic if, given two operations $+$ and $\times$, we have $\widehat{H}(d + d') = \widehat{H}(d) \times \widehat{H}(d')$. The protocol in [48] is illustrated in Figure 2.2.

The server's response $S = \widehat{H}(d)$ is a homomorphic function; $\widehat{H}(d+d') \equiv r^{d+d'} \equiv r^d r^{d'} \equiv \widehat{H}(d)\widehat{H}(d') \bmod N$. To find a collision for this hash function, one has to find two messages $d,\ d'$ such that $r^d \equiv r^{d'}$, *i.e.*, $r^{d-d'} \equiv 1 \bmod N$. Thus, $d - d'$ must be multiple of $\phi(N)$. Finding such two messages $d,\ d'$ is believed to be difficult since the factorization of N is unknown. The limitation of the protocol presented in [48] is similar to that of the protocol in [39]: the archive service provider has to exponentiate the entire data file, which is a heavy computation overhead especially for large files.

Data owner:

- Generates RSA modulus $N = pq$ (p & q are prime numbers)
- Computes $\phi(N) = (p-1)(q-1)$
- Pre-computes and stores $\bar{h}(d) = d \bmod \phi(N)$ (d is the data file)
- Sends the data file d to the remote server

Challenge Response

Verifier		Remote Server
1. Picks $r \in_R \mathbb{Z}_N$		
	$\xrightarrow{r}$	
		2. Computes a response $S = \widehat{H}(d) = r^d \bmod N$
	$\xleftarrow{R}$	
3. Computes $V = r^{\bar{h}(d)} \bmod N$		
4. Checks $V \overset{?}{=} S$		

Figure 2.2: The PDP protocol by Filho *et al.* [48].

To circumvent the problem of exponentiating the entire file, Sebé *et al.* [80] presented a scheme to remotely verify data integrity by first fragmenting the file into blocks, fingerprinting each file block, and then using an RSA-based hash function on the blocks. Thus, the data file F is divided into a set of m blocks: $F = \{b_1, b_2, \ldots, b_m\}$, where m fingerprints $\{M_j\}_{1 \le j \le m}$ are generated for the file and stored on the verifier local storage. Their scheme does not require the exponentiation of the entire file. Figure 2.3 demonstrates the protocol of Sebé *et al.* [80].

Although the protocol presented by Sebé *et al.* [80] does not require exponentiation of the entire file, a local copy of the fingerprints – whose size is linear in the number of file blocks – must be stored on the verifier side. The verifier has to store the fingerprints $\{M_j\}_{1 \le j \le m}$, each of size $|N|$ bits consuming $m|N|$ bits from the verifier local storage, which may impede the verification process when using small devices like PDAs or cell phones.

Moreover, this protocol supports only private verifiability, *i.e.*, only the data owner can challenge the remote server and validate the data possession. If there is a dispute regarding data integrity, we cannot resort to a trusted third party auditor to resolve such a dispute.

Data owner:

- Generates RSA modulus $N = pq$ (p & q are prime numbers)
- Computes $\phi(N) = (p-1)(q-1)$
- Divides the data file F into m blocks: $F = \{b_1, b_2, \ldots, b_m\}$
- Pre-computes and stores $M_j = b_j \bmod \phi(N)$ $(1 \leq j \leq m)$
- Sends the data file F to the remote server

Challenge Response

Verifier — Remote Server

1. Picks $r \in_R \mathbb{Z}_N$
2. Generates $l(\leq m)$ random values $\{c_j\}_{1 \leq j \leq l}$

$$\xrightarrow{r, \{c_j\}_{1 \leq j \leq l}}$$

3. Computes $a = \sum_{j=1}^{l} c_j \cdot b_j$
4. Computes $S = r^a \bmod N$

$$\xleftarrow{R}$$

5. Computes $a' = \sum_{j=1}^{l} c_j \cdot M_j \bmod \phi(N)$
6. Computes $V = r^{a'} \bmod N$
7. Checks $V \overset{?}{=} S$

Figure 2.3: The PDP protocol by Sebé *et al.* [80].

2.2.3 Data Storage Commitment Schemes

Golle *et al.* [52] provided a scheme to verify data storage commitment, a concept that is weaker than integrity. They investigated "storage-enforcing commitment scheme". Through their scheme a storage server demonstrates that it is making use of storage space as large as the client's data, but not necessarily the same exact data. The storage server does not directly prove that it is storing a file F, but proves that it has committed sufficient resources to do so. Their scheme is based on n-Power Computational Diffie-Hellman (n-PCDH) assumption: for a group $\mathbb{Z}_p$ (p is a prime number) with a generator g, there is no known probabilistic polynomial time algorithm A that can compute g^{x^n} given $g^x, g^{x^2}, \ldots, g^{x^{n-1}}$ with non-negligible probability. Figure 2.4 illustrates the scheme of Golle *et al.* [52].

Each file block $b_j \in \mathbb{Z}_p$ can be represented by $\lceil \log_2 \ p \rceil$ bits, and thus the total number of bits to store the file $F = m\lceil \log_2 \ p \rceil$ bits. For the storage server to cheat by storing all the possible values of f_k (*i.e.*, $m+1$ values), it needs $(m+1)\lceil \log_2 \ p \rceil$ bits which is slightly larger than the size of the original file.

The guarantee provided by the protocol in [52] is weaker than data integrity since it only ensures that the server is storing something at least as large as the original data file but not necessarily the file itself. In addition, the verifier's public key is about twice as large as the data file.

2.2.4 Privacy-Preserving PDP Schemes

Shah *et al.* [82, 83] presented privacy-preserving PDP protocols. Using their schemes, an external third party auditor (TPA) can verify the integrity of files stored by a remote server without knowing any of the file contents. The data owner first encrypts the file, then sends both the encrypted file along with the encryption key to the remote server. Moreover, the data owner sends the encrypted file along with a key-commitment that fixes a value for the key without revealing the key to the TPA. The primary purposes of the schemes presented in [82, 83] are to ensure that the remote server is correctly possessing the client's data along with the encryption key, and to prevent any information leakage

Setup

- File $F = \{b_1, b_2, \dots, b_m\}, b_j \in \mathbb{Z}_p$
- Let $n = 2m + 1$
- Secret key $sk = x \in_R \mathbb{Z}_p$
- Public key $pk = (g^x, g^{x^2}, \dots, g^{x^n}) = (g_1, g_2, \dots, g_n)$
- Data owner computes and stores $f_0 = \prod_{j=1}^{m} g_j^{b_j} \bmod p$

Challenge Response

Verifier		Remote Server
1. Picks a random $k \in [0, m]$	$\xrightarrow{k}$	
		2. Computes $f_k = \prod_{j=1}^{m} g_{j+k}^{b_j}$
	$\xleftarrow{f_k}$	
3. Checks $f_0^{x^k} \overset{?}{=} f_k$		

Figure 2.4: The PDP protocol by Golle *et al.* [52].

to the TPA which is responsible for the auditing task. Thus, clients – especially with constrained computing resources and capabilities – can resort to external audit party to check the integrity of outsourced data, and this third party auditing process should bring in no new vulnerabilities towards the privacy of client's data. In addition to the auditing task of the TPA, it has another primary task which is extraction of digital contents. For the auditing task, the TPA interacts with the remote server to check that the stored data is intact. For the extraction task, the TPA interacts with both the remote server and the data owner to first check that the data is intact then delivers it to the owner. The protocols presented by Shah *et al.* [82, 83] are illustrated in Figure 2.5.

Setup

- Data owner sends a key K and the encrypted file $E_K(F)$ to the remote server
- Data owner sends a key-commitment value g^K and the encrypted file $E_K(F)$ to the TPA (g is a generator for $\mathbb{Z}_p$)
- The TPA generates a list L of random values and HMACs: $L = \{(R_i, \widetilde{H}_i)\}_{1 \leq i \leq \widetilde{N}}$, $\widetilde{H}_i = \text{HMAC}(R_i, E_K(F))$, and R_i is a random number.
- TPA keeps $\{L, h(E_K(F)), g^K\}$ and can discard $E_K(F)$ (h is a secure hash function)

TPA **Remote Server**

Checking Data Integrity

1. Picks any $(R_i, \widetilde{H}_i)$ from L and updates $L = L \backslash \{(R_i, \widetilde{H}_i)\}$

$$\xrightarrow{\hspace{2em}R_i\hspace{2em}}$$

2. Computes $\widetilde{H}_s$=HMAC$(R_i, E_K(F))$

$$\xleftarrow{\hspace{2em}\widetilde{H}_s\hspace{2em}}$$

3. Checks $\widetilde{H}_i \overset{?}{=} \widetilde{H}_s$

Checking Key Integrit

1. Generates $\beta \in_R \mathbb{Z}_p$

$$\xrightarrow{\hspace{2em}g^\beta\hspace{2em}}$$

2. Computes $W_s = (g^\beta)^K$

$$\xleftarrow{\hspace{2em}W_s\hspace{2em}}$$

3. Checks $(g^K)^\beta \overset{?}{=} (W_s)$

Data Extraction

$$\xleftarrow{\hspace{2em}D_s = E_K(F)\hspace{2em}}$$

- Checks the hash of its local cached copy:
  $h(E_K(F)) \overset{?}{=} h(D_s)$. If valid, sends $E_K(F)$ to the owner

Key Extraction

- Assume that the owner and the server agree on a shared random secret X

$$\xleftarrow{\hspace{2em}K+X,\ g^X\hspace{2em}}$$

- Checks $g^{K+X} \overset{?}{=} g^K \cdot g^X$. If valid, sends $K + X$ to the owner
- Owner gets $K = (K + X) - X$

Figure 2.5: The PDP protocols by Shah *et al.* [82, 83].

The protocols presented in [82, 83] achieve privacy-preserving towards third party auditing process and extract digital contents from remote servers, but have some limitations:

- Limited number of verifications for a particular data item (must be fixed beforehand).
- Storage overhead on the TPA; it has to store $\widetilde{N}$ hash values for each file to be audited.
- Lack of support for *stateless* verification; the TPA has to update its state (the list L) between audits to prevent using the same random number or the same HMAC twice.
- High communication complexity to retrieve $E_K(F)$ if the TPA wants to regenerate a new list of hash values to achieve unlimited number of audits.

2.2.5 PDP in Database Context

In the database outsourcing scenario, the database owner stores data at a storage service provider and the database users send queries to the service provider to retrieve some tuples/records that match the issued query. Data integrity is an imperative concern in the database outsourcing paradigm; when a user receives a query result from the service provider, it is crucial to verify that the received tuples are not being tampered with by a malicious service provider. Mykletun *et al.* [69] investigated the notion of signature aggregation to validate the integrity of the query result. Signature aggregation enables bandwidth- and computation-efficient integrity verification of query replies. In the scheme presented in [69], each database record is signed before outsourcing the database to a remote service provider.

Mykletun *et al.* [69] provided two aggregation mechanisms: one is based on RSA [76] and the other is based on BLS signature [27]. For the scheme based on the RSA signature, each record in the database is signed as: $\sigma_j = h(b_j)^d \bmod N$, where h is a one-way hash function, b_j is the data record, d is the RSA private key, and N is the RSA modulus. A user issues a query to be executed over the outsourced database, the server processes the query and computes an aggregated signature $\sigma = \sum_{j=1}^{t} \sigma_j \bmod N$, where t is the number of records in the query result. The server sends the query result along with the aggregated signature to the user. To verify the integrity of the received records, the user checks $\sigma^e \overset{?}{=} \prod_{j=1}^{t} \sigma_j \bmod N$, where e is the RSA public key.

The second scheme presented by Mykletun *et al.* [69], which is based on the BLS signature [27] is similar to the first scheme but the record signature $\sigma_j = h(b_j)^x$, where $x \in_R \mathbb{Z}_p$ is a secret key. To verify the integrity of the received records, the user checks $\hat{e}(\sigma, g) \overset{?}{=} \hat{e}(\prod_{j=1}^{t} h(b_j), y)$, where g is a generator of the group $\mathbb{Z}_p$, $y = g^x$(public key), and $\hat{e}$ is a computable bilinear map (will be explained later in the thesis).

Correctness and Completeness are imperative concerns in the database outsourcing paradigm. Completeness means that the service provider should send *all* records that satisfy the query criteria not just subset of them. The completeness requirement was not considered by the schemes presented in [69], and it has been addressed by other researchers (see for example [89, 60]).

The schemes provided in [69] depend on the retrieved records of the query result to verify the integrity of the outsourced database. On the other hand, efficient PDP schemes require *blockless* verification, *i.e.*, the verifier has to have the ability to validate data integrity even though he neither possesses nor retrieves any of the file blocks. Blockless verification is a main concern to minimize the required communication cost over the network.

2.2.6 PDP Schemes Based on Homomorphic Verifiable Tags

Ateniese *et al.* [8] presented a model to overcome some of the limitations of other PDP protocols: limited number of audits per file determined by fixed challenges that must be specified in advance, expensive server computation by doing the exponentiation over the entire file, storage overhead on the verifier side by keeping some metadata to be used later in the auditing task, high communication complexity, and lack of support for blockless verification. Ateniese *et al.* [8] provided a PDP model in which the data owner fragments the file F into blocks $\{b_1, b_2, \ldots, b_m\}$ and generates metadata (a tag) for each block to be used for verification. The file is then sent to be stored on a remote/cloud server, which may be untrusted and the data owner may delete the local copy of the file. The remote server provides a proof that the data has not been tampered with or partially deleted by responding to challenges sent from the verifier. The scheme presented in [8] provides *probabilistic* guarantee of data possession, where the verifier checks a random subset of stored file blocks with each challenge (spot checking).

Homomorphic verifiable tags (HVTs)/homomorphic linear authenticators (HLAs) are the basic building blocks of the PDP scheme presented in [8]. Briefly, the HVTs/HLAs are unforgeable verification metadata constructed from the file blocks in such a way that the verifier can be convinced that a linear combination of the file blocks is accurately computed by verifying only the aggregated tag/authenticator. In the work of [8], the authors differentiate between the concept of *public verifiability* and *private verifiability*. In public verifiability anyone – not necessarily the data owner – who knows the owner's public key can challenge the remote server and verify that the server is still possessing the owner's files. On the other side, private verifiability allows only the original owner to perform the auditing task. Two main PDP schemes are presented in [8]: sampling PDP (S-PDP) and efficient PDP (E-PDP) schemes. In fact, there is a slight difference between these two models, but the E-PDP scheme provides a weaker guarantee of data possession. The E-PDP protocol guarantees only the possession of the *sum* of file blocks and not necessarily the possession of each one of the blocks being challenged. Both protocols presented in [8] are illustrated in Figure 2.6.

The schemes of Ateniese *et al.* [8] have resolved many constraints of other PDP protocols. However, their schemes are based on RSA, which make the HVTs relatively long; each file block has an HVT in order of $|N|$ bits. Thus, to achieve 80-bit security level, the generated tag should be of size 1024 bits. Shacham and Waters [81] presented an attack against the E-PDP scheme, which enables a malicious server to cheat with non-negligible probability requiring no more storage than an honest server to store the file.

Ateniese *et al.* [10] showed that the HLAs can be constructed from homomorphic identification protocols. They provided a ”compiler-like” transformation to build HLAs from homomorphic identification protocols and showed how to turn the HLA into a PDP scheme. As a concrete example, they applied their transformation to a variant of an identification protocol proposed by Shoup [84] yielding a factoring-based PDP scheme.

I. S-PDP scheme

Setup

- $N = pq$ is the RSA modulus (p & q are prime numbers)
- g is a generator of QR_N (QR_N is the set of quadratic residues modulo N)
- Public key $pk = (N, g, e)$, secret key $sk = (d, v)$, $v \in_R \mathbb{Z}_N$, and $ed \equiv 1 \bmod (p-1)(q-1)$
- π is a pseudo-random permutation, f is a pseudo-random function, H is a hash-and-encode function ($H : \{0,1\}^* \rightarrow QR_N$), and h is a cryptographic hash function.
- File $F = \{b_1, b_2, \ldots, b_m\}$
- Data owner generates a tag T_j for each block b_j: $T_j = (H(v||j) \cdot g^{b_j})^d \bmod N$
- Data owner sends $F = \{b_j\}_{1 \leq j \leq m}$ and $\{T_j\}_{1 \leq j \leq m}$ to the remote server

Challenge Response

Verifier — Remote Server

1. Picks two keys k_1(key for π), k_2(key for f), c(# of blocks to be challenged), and $g_s = g^s \bmod N (s \in_R \mathbb{Z}_N)$

$$\xrightarrow{c,\, k_1,\, k_2,\, g_s}$$

2. Computes challenged block indices: $\{j_i\} = \pi_{k_1}(i)_{1 \leq i \leq c}$
3. Computes random values: $\{a_i\} = f_{k_2}(i)_{1 \leq i \leq c}$
4. Computes $T = \prod_{i=1}^{c} T_{j_i}^{a_i} \bmod N$
5. Computes $\rho = h(g_s^{\sum_{i=1}^{c} b_{j_i} \cdot a_i} \bmod N)$

$$\xleftarrow{T,\, \rho}$$

6. Computes $\{j_i\} = \pi_{k_1}(i)_{1 \leq i \leq c}$ and $\{a_i\} = f_{k_2}(i)_{1 \leq i \leq c}$
7. Computes $\tau = \dfrac{T^e}{\prod_{i=1}^{c} H(v||j_i)^{a_i}}$
8. Checks $h(\tau^s \; mod \; N) \overset{?}{=} \rho$

$\Longrightarrow$ ***continue***

II. E-PDP scheme

The only difference between the E-PDP and the S-PDP is that : $\{a_i\}_{1 \leq i \leq c} = 1$, and thus

- Step 4 : $T = \prod_{i=1}^{c} T_{j_i} \bmod N$
- Step 5 : $\rho = H(g_s^{\sum_{i=1}^{c} b_{j_i}} \bmod N)$
- Step 7 : $\tau = \frac{T^e}{\prod_{i=1}^{c} H(v||j_i)}$

Figure 2.6: The S-PDP and E-PDP protocols by Ateniese *et al.* [8].

2.2.7 Comparison

Table 2.1 provides a comparison between the PDP schemes presented in this section. This comparison is based on the following:

- Owner pre-computation: the operations performed by the data owner to process the file before being outsourced to a remote server.
- Verifier storage overhead: the extra storage required to store some metadata on the verifier side to be used later during the verification process.
- Server storage overhead: the extra storage on the server side required to store some metadata – not including the original file – sent from the owner.
- Server computation: the operations performed by the server to provide the data possession guarantee.
- Verifier computation: the operations performed by the verifier to validate the server's response.
- Communication cost: bandwidth required during the challenge response phase.

- Unbounded challenges: whether the scheme allows unlimited number of auditing the data file, or a fixed number of challenges.

- Fragmentation: whether the file is treated as one chunk, or divided into smaller blocks.

- Type of guarantee: whether the guarantee provided from the remote server is deterministic guarantee, which requires to access all file blocks, or probabilistic guarantee that depends on spot checking.

- Prove data possession: whether the scheme proves the possession of the file itself, or proves that the server is storing something at least as large as the original file.

We use the notations EXF to indicate the EXponentiation of the entire File, DET to indicate deterministic guarantee, and PRO to indicate probabilistic guarantee. For simplicity, the security parameter is not included as a factor for the relevant costs.

Table 2.1: Comparison of PDP schemes for a file containing m blocks, c is the number of blocks to be challenged, and $\widetilde{N}$ is a finite number of random challenges.

Scheme	[39]	[48]	[80]	[52]	[82, 83]	[8]
Owner pre-computation	EXF	$O(1)$	$O(m)$	$O(m)$	$O(1)$	$O(m)$
Verifier storage overhead	$O(1)$	$O(1)$	$O(m)$	$O(1)$	$O(\widetilde{N})$	-
Server storage overhead	-	-	-	-	-	$O(m)$
Server computation	EXF	EXF	$O(c)$	$O(m)$	$O(1)$	$O(c)$
Verifier computation	$O(1)$	$O(1)$	$O(c)$	$O(1)$	$O(1)^{\dagger}$	$O(c)$
Communication cost	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Unbounded challenges	✓	✓	✓	✓	×	✓
Fragmentation	×	×	✓	✓	×	✓
Type of guarantee	DET	DET	DET/ PRO	DET	DET	PRO‡
Prove data possession	✓	✓	✓	×	✓	✓

[†] Verifier pre-computation is $O(\widetilde{N})$ to generate a list L of HMACs.

[‡] The scheme in [8] can be easily modified to support deterministic guarantee.

2.3 Provable Dynamic Data Possession

One of the core design principles of outsourcing data is to provide dynamic scalability of data for various applications. In this section we review different provable dynamic data possession (PDDP) schemes, where the data owner can issue requests to update and scale the outsourced data.

2.3.1 Hash-Based PDDP Schemes

Ateniese *et al.* [11] proposed a dynamic version of the PDP scheme based on cryptographic hash function and symmetric key encryption. Their scheme is efficient but allows only a fixed number of challenges due to the fact that through the scheme setup they come up with all future challenges and store pre-computed responses as tokens. These tokens can be stored either at the verifier side in a plain form or at the server side in an encrypted form. Block insertion in [11] cannot explicitly be supported (append operation is supported). Figure 2.7 summarizes the scheme presented in [11].

2.3.2 PDDP Schemes Based on Authenticated Data Structures

Rank-Based Authenticated Skip Lists

A skip list is a hierarchical structure of linked lists [74], and is used to store a *sorted* set of items. Each node v in a normal skip list stores two pointers (right and down) denoted by $\mathsf{rgt}(v)$ and $\mathsf{dwn}(v)$ to be used during the searching procedure for a specific target in the leaf nodes (nodes at the base/bottom level). On the other hand, each node v in an *authenticated* skip list stores $\mathsf{rgt}(v)$, $\mathsf{dwn}(v)$, and a label $\mathsf{f}(v)$ computed by recursively applying a hash function to $\mathsf{f}(\mathsf{rgt}(v))$ and $\mathsf{f}(\mathsf{dwn}(v))$. The authenticated skip list can provide a proof to indicate whether a specific element belongs to the set represented by the list or not. In addition to $\mathsf{rgt}(v)$, $\mathsf{dwn}(v)$ and $\mathsf{f}(v)$, each node v in a *rank-based* authenticated skip list stores the number of nodes at the bottom level that can be reached from v. This number is called the rank of v: $\mathsf{r}(v)$.

Setup

- Data file F is a set of blocks $\{b_1, b_2, \ldots, b_m\}$
- g is a pseudo-random permutation, f is a pseudo-random function, and h is a cryptographic hash function. f is used to generate keys for g and to generate random numbers.
- E_K and E_K^{-1} are encryption and decryption algorithms under a key K
- Two master keys W and Z
- Data owner generates t random challenges and their corresponding responses/tokens $\{\nu_i\}_{1\leq i\leq t}$ as follows.

```
  for $i = 1$ to $t$ do
    1. Generate $k_i = f_W(i)$ and $c_i = f_Z(i)$
    2. $\nu_i = h(c_i, 1, b_{g_{k_i}(1)}) \oplus \cdots \oplus h(c_i, r, b_{g_{k_i}(r)})$ /* $r$ is # of blocks per token */
    3. $\nu'_i = E_k(ctr, i, \nu_i)$ /* ctr is an integer counter */
```

- Owner sends the file $F = \{b_j\}_{1\leq j\leq m}$ and $\{\nu'_i\}_{1\leq i\leq t}$ to the remote server.

Challenge Response

Data owner | Remote Server

Begin challenge i

1. Generates $k_i = f_W(i)$ and $c_i = f_Z(i)$

$$\xrightarrow{k_i, c_i}$$

2. $z = h(c_i, 1, b_{g_{k_i}(1)}) \oplus \cdots \oplus h(c_i, r, b_{g_{k_i}(r)})$

$$\xleftarrow{z, \nu'_i}$$

3. Computes $\nu = E_K^{-1}(\nu'_i)$
4. Checks $\nu \overset{?}{=} (ctr, i, z)$

End challenge i

$\Longrightarrow$ ***continue***

Dynamic Operations

Modify

/ Assume that block b_j is to be updated to b'_j */*

```
Data owner                                                Remote Server
                 $\xleftarrow{1.\ \{\nu'_i\}_{1 \le i \le t}}$
2. $ctr = ctr + 1$
3. for $i = 1$ to $t$ do
    3.1 $z'_i = E_K^{-1}(\nu'_i)$
        /* if decryption fails, exit*/
        /* if $z'_i$ is not prefixed by (ctr-1) and $i$, exit */
    3.2 extracts $\nu_i$ from $z'_i$
    3.3 computes $k_i = f_W(i)$ and $c_i = f_Z(i)$
  /* update all tokens even if they do not include the block to be updated */
    3.4 for $l = 1$ to $r$ do
            if($g_{k_i}(l) == j$) then
                $\nu_i = \nu_i \oplus h(c_i, l, b_j) \oplus h(c_i, l, b'_j)$
    3.5 $\nu'_i = E_k(ctr, i, \nu_i)$
                 $\xrightarrow{j,\ b'_j,\ \{\nu'_i\}_{1 \le i \le t}}$
```

Delete

/ Assume that block b_j is to be deleted */*

The logic of Delete is similar to Modify operation but replaces the block to be modified with a special block "DBlock". So, the inner ***for*** loop (step 3.4) will be:

```
3.4 for $l = 1$ to $r$ do
    if($g_{k_i}(l) == j$) then
    $\nu_i = \nu_i \oplus h(c_i, l, b_j) \oplus h(c_i, l, \mathsf{DBlock})$
```

Insert

Physical insert is not supported. Append operation is allowed by viewing the data file as a two dimensional structure (matrix), and appending the new block in a round robin fashion.

Figure 2.7: The PDDP protocol by Ateniese *et al.* [11].

Figure 2.8 [42] shows an example of rank-based skip list, where the number inside the node represents its rank. The top leftmost node (w_7) of the skip list is considered to be the *start* node. To access any node at the bottom level, searching should begin from the start node.

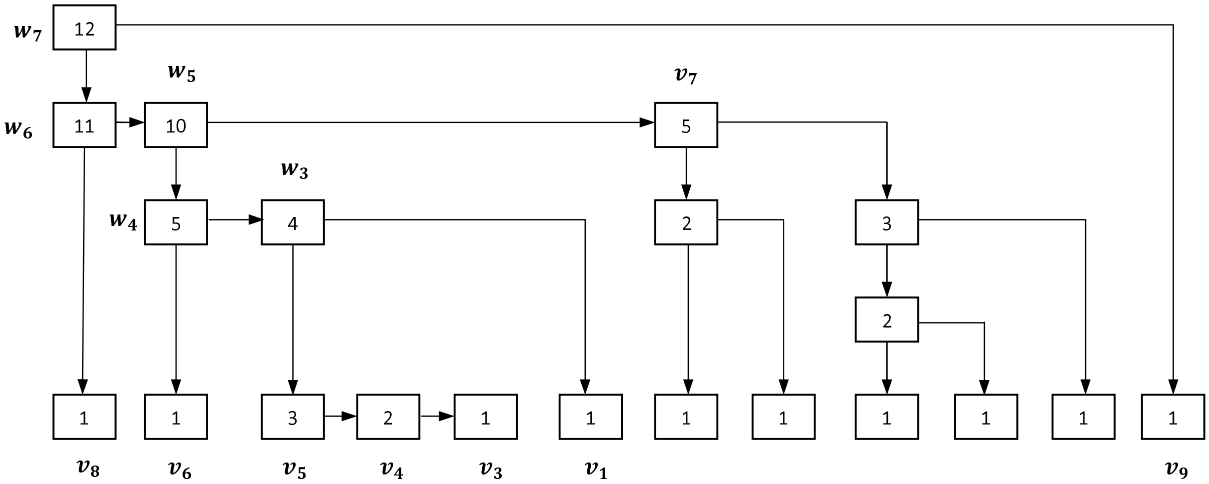


Figure 2.8: Example of rank-based skip list [42].

PDDP Schemes of Erway *et al.*

Erway *et al.* [42] constructed a PDDP scheme based on the PDP model of [8] to support provable updates of stored data files using rank-based authenticated skip lists. Their protocol supports block insertion by eliminating the index information in the tag computation of [8]. The purpose of using the rank-based authenticated skip list in [42] is to authenticate the tag information of the blocks to be updated or challenged.

In the PDDP scheme of [42], the File F is fragmented into m blocks $\{b_1, b_2, \ldots, b_m\}$. A representation/tag $\mathcal{T}(b_j)$ of block b_j is computed as $\mathcal{T}(b_j) = g^{b_j} \bmod N$ (N is the RSA modulus and g is an element of high order in $\mathbb{Z}_N^*$). The block representation $\mathcal{T}(b_j)$ is stored at the j^{th} bottom-level node of the authenticated skip list and the block itself is stored elsewhere by the server. The tags protect the integrity of file blocks, while the authenticated list ensures the security and integrity of tags.

During the challenge phase, the client requests the server to prove the integrity of randomly selected c blocks $\{b_{j_i}\}_{1 \leq j_1,\ldots,j_c \leq m}$. The server sends the tags $\{\mathcal{T}(b_{j_i})\}_{1 \leq j_1,\ldots,j_c \leq m}$ along with their search/verification paths. The server also sends a combined block $M = \sum_{i=1}^{c} a_i.b_{j_i}$, where $\{a_i\}_{1 \leq i \leq c}$ are random values sent by the client as part of the challenge. The owner verifies the search/verification paths of the block tags using metadata $\mathcal{M}_c$, which is the label of the start node. Besides, the owner computes $T = \prod_{i=1}^{c} \mathcal{T}(b_{j_i})^{a_i} \bmod N$. Data integrity is valid only if the search paths are verified and $T = g^M \bmod N$.

The authenticated skip list is used to modify, insert, and delete the block tags achieving the dynamic behavior of the data file. Nodes of skip list along the search/verification path – from the start node to the node associated with the block to be updated – are only affected by the dynamic operations of file blocks.

The scheme presented in [42] can be summarized by the following procedures:

- *Key-generation.* This procedure is run by the data owner (the client) and outputs a secret key sk and public key pk. The secret key is kept by the client and the public key is sent to the remote server.

- *Update-preparation.* This procedure is run by the client to prepare a part of the file for storage on a remote server, which may not be trustworthy. The input parameters to this procedure are sk, pk, a part of the file F, updates to be performed (*e.g.*, full re-write, modify block, delete block, or insert block), and the previous metadata $\mathcal{M}_c$. The output is an encoded version of that part of the file $\mathsf{e}(F)$, encoded information $\mathsf{e}(\mathsf{info})$ about the update, and the new metadata $\mathsf{e}(\mathcal{M})$. The output of this procedure is sent to the remote server.

- *Update-execution.* Upon receiving an update request from the data owner, the server runs this procedure in response to the owner's request. The input parameters to this procedure are pk, the previous version of the file denoted as F_{i-1}, the metadata $\mathcal{M}_{i-1}$, and the values produced by the client during the *Update-preparation* algorithm. The outputs of this procedure are the new version of the file denoted as F_i , metadata $\mathcal{M}_i$, and metadata $\mathcal{M}'_c$ to be sent to the owner along with its proof $P_{\mathcal{M}'_c}$.

- *Update-verification.* This procedure is run by the client to verify the server's behavior during the updates. The input parameters to this procedure are all inputs of *Update-preparation* algorithm, the metadata $\mathcal{M}'_c$, and the proof $P_{\mathcal{M}'_c}$ ($\mathcal{M}'_c$ and $P_{\mathcal{M}'_c}$ are sent from the server as outputs of the *Update-execution* algorithm). The output of the *Update-verification* algorithm is either acceptance or rejection signal.

- *Challenge.* This procedure is run by the client to challenge the server and verify the integrity of the remotely stored data file. It takes as input sk, pk, and the latest client metadata $\mathcal{M}_c$. The output is a challenge c that is sent from the client to the server.

- *Proof-computation.* Upon receiving the challenge c from the client, the server runs the *Proof-computation* algorithm in response to the owner's challenge. The input parameters to this procedure are pk, the latest version of the file, the metadata, and the challenge c. It outputs a proof P that is sent to the client.

- *Proof-verification.* This procedure is run by the client to validate the proof P received from the server. The input parameters to this procedure are sk, pk, the client metadata $\mathcal{M}_c$, the challenge c, and the proof P sent by the server. The output of this procedure is "accept" to indicate that the server still possesses the file intact or "reject" otherwise.

In their work, Erway *et al.* [42] presented a variant of the PDDP scheme using RSA trees instead of rank-based authenticated lists. Wang *et al.* [87] used Merkle hash trees [67] (instead of skip lists) and homomorphic authenticators built from BLS signatures [27] to construct a PDDP scheme.

2.3.3 RSA-Based PDDP Schemes

Hao *et al.* [54] adapted the protocol presented in [80] to support both data dynamic and public verifiability. The latter allows that anyone who knows the owner's public key can challenge the remote server and verify that the server is still possessing the owner's files.

If a dispute regarding data integrity occurs between the owner and the CSP, a third party auditor can determine whether the data integrity is maintained or not. This third party auditing process should bring in no new vulnerabilities towards the privacy of owner's data. The protocol presented in [54] ensures that the data is kept private during the third party verification, where no private information contained in the data is leaked. Figure 2.9 summarizes the protocol presented in [54].

Setup

- $N = pq$ is the RSA modulus (p and q are prime numbers)
- g is a generator of Q_{RN} (Q_{RN} is the set of quadratic residues modulo N)
- Public key $pk = (N, g)$ and secret key $sk = (p, q)$.
- f is a pesudo-random function
- File $F = \{b_1, b_2, \ldots, b_m\}$.
- Data owner generates a tag D_j for each block b_j, where $D_j = g^{b_j} \bmod N$
- The tags are stored on the owner side and the file is sent to the remote server.

Challenge Response

Verifier — Remote Server

1. Generates a random key r
2. Computes $g_s = g^s \bmod N$ ($s \in_R \mathbb{Z}_N$)

$\xrightarrow{r, g_s}$

3. Generates random coefficients $\{a_j = f_r(j)\}_{1 \leq j \leq m}$
4. Computes $P = (g_s)^{\sum_{j=1}^{m} a_j . b_j} \bmod N$

$\xleftarrow{R}$

5. Generates a set of random coefficients $\{a_j = f_r(j)\}_{1 \leq j \leq m}$
6. Computes $\acute{P} = \prod_{j=1}^{m} (D_j^{a_j} \bmod N) \bmod N$
7. Computes $V = \acute{P}^s \bmod N$
8. Checks $V \overset{?}{=} P$

$\Longrightarrow$ ***continue***

Dynamic Operations

Modify

/ Assume that block b_j is to be updated to b'_j */*

- Server updates b_j to b'_j
- Owner computes a new block tag $D'_j = g^{b'_j} \bmod N$.
  So, the new block tags are $\{D_1, D_2, \ldots, D'_j, \ldots, D_m\}$

Insert

/ Assume a new block $\hat{b}$ is to be inserted after position j or appended at the end */*

- The server updates its file to be
  $\{b_1, b_2, \ldots, b_j, \hat{b}, \ldots, b_{m+1}\}$ (insert: $b_{j+1} = \hat{b}$) or $\{b_1, b_2, \ldots, b_m, \hat{b}\}$ (append).
- The owner computes a new block tag $\widehat{D} = g^{\hat{b}} \bmod N$, and changes the block tags to $\{D_1, D_2, \ldots, D_j, \widehat{D}, \ldots, D_{m+1}\}$ (insert: $D_{j+1} = \widehat{D}$) or $\{D_1, D_2, \ldots, D_m, \widehat{D}\}$ (append)

Delete

/ Assume a block at position j is to be deleted*/*

- Server deletes the block b_j
- Owner deletes the corresponding tag D_j

Figure 2.9: The PDDP protocol by Hao *et al.* [54].

2.3.4 Cooperative PDP Schemes

Zhu *et al.* [93] addressed the construction of cooperative PDP scheme on hybrid clouds to support scalability of service and data migration. A hybrid cloud is a deployment model in which an organization provides and handles some internal and external resources. For example, an organization can use a public cloud service like Amazon EC2 [5] to perform the general computation, while the data files are stored within the organization's local data center in a private cloud. In their work, Zhu *et al.* [93] consider the existence of multiple CSPs that cooperatively store customers' data. The data owners are allowed to dynamically access and update their data for various applications, and the verification process is performed for the owners in hybrid clouds.

Remark. Generally speaking, PDP and PDDP schemes are considered to be secure if (i) a polynomial-time algorithm that can cheat the verifier and pass the auditing procedure with non-negligible probability does not exist; and (ii) there exists a polynomial-time extractor that can repeatedly execute the challenge response protocol until it extracts the original data file.

2.4 Proof of Retrievability

Proof of retrievability (POR) is a complementary approach to PDP, and is stronger than PDP in the sense that the verifier can reconstruct the entire data file from the responses that are reliably transmitted from the server. This is due to encoding of the data file, for example using *erasure codes*, before outsourcing to allow more error-resiliency. Thus, if it is a crucial demand to detect any modification or deletion of tiny parts of the data file, then encoding could be applied before outsourcing data to remote servers.

Schwartz and Miller [79] have proposed the use of algebraic signatures to verify data integrity across multiple servers. Through keyed algebraic encoding and stream cipher encryption, they are able to detect file corruptions. The communication complexity in the model of [79] is an issue for it is linear with respect to the queried data size. Moreover, the security of their proposal is not proven and remains in question [92].

The work done by Juels and Kaliski [56] is one of the first efforts to consider formal models for POR schemes. In their model, the data is first encrypted then disguised blocks (called sentinels) are embedded into the ciphertext. The sentinels are hidden among the regular file blocks in order to detect data modification by the server. In the auditing phase, the verifier requests for randomly picked sentinels and checks whether they are corrupted or not. If the server corrupts or deletes parts of the data, then sentinels would also be influenced with a certain probability. The scheme in [56] allows only for a limited number of challenges on the data files, which is specified by the number of sentinels embedded into the data file. This limited number of challenges is due to the fact that sentinels and their position within the file must be revealed to the server at each challenge and the verifier cannot reuse the revealed sentinels.

Shacham and Waters [81] proposed a compact proof of retrievability model that enables the verifier to unboundedly challenge the server addressing the limitation of [56]. Among the main contributions of [81] is the construction of HLAs that enable the server to aggregate the tags of individual file blocks and to generate a single *short* tag as a response to the verifier's challenge. Shacham and Waters [81] proposed two HLAs: one is based on the pseudo-random function, and the other is based on the BLS signature [27].

Bowers *et al.* [29] presented a distributed cryptographic system known as HAIL (High-Availability and Integrity Layer), which improves upon POR deployed on individual servers. Their system allows a set of servers to prove to a data owner that the outsourced data is intact and retrievable. Various POR schemes can be found in the literature, *e.g.*, [36, 30, 41]

2.5 Summary

In this chapter, we have described the concept of PDP as a technique to verify the integrity of data stored on remote sites. We have reviewed different PDP schemes designed for static data. To efficiently validate the integrity of outsourced data, a number of challenges have to be addressed: (i) the computation overhead on the server side to prove data possession, (ii) the verifier's computations complexity to check server responses, (iii) the storage overhead on both the verifier and server sides, (iv) the communication cost to send a challenge vector and receive a response; and (v) the permission for unlimited number of data audits.

Moreover, in this chapter we have presented some PDDP models that deal with dynamic data. Through these models the data owner is able to send requests to the remote server for updating/scaling the stored data. The verifier is enabled to make sure that the outsourced data is consistent with the most recent modifications issued by the owner. We have also highlighted the concept of POR as a complementary approach to PDP. The main idea of POR schemes is to apply encoding to data files before outsourcing, which allows to reconstruct the entire data file utilizing the responses that are reliably transmitted from the server.

Finally, the PDP, PDDP, and POR schemes presented in this chapter focus on a *single copy* of the file and provide no proof that the CSP stores multiple copies of the owner's file. The problem of creating multiple copies of a data file and auditing those copies to verify their completeness and correctness was outside the scope of the schemes presented in this chapter. There are some previous work on maintaining file copies through distributed systems to achieve availability and durability (*e.g.*, [34, 61]), but it does not focus on guaranteeing that multiple copies of the data file are actually stored.

Chapter 3

Integrity Verification for Multiple Data Copies

In this chapter, we consider the problem of verifying the integrity of multiple data copies stored on cloud/remote servers, and describe the scheme we proposed in [20, 15, 18]. Section 3.1 highlights the motivation of this work. Section 3.2 presents a basic provable multi-copy data possession scheme, and a review of the multiple-replica provable data possession (MR-PDP) scheme due to Curtmola *et al.* [37]. Our system model and assumptions are presented in Section 3.3. Our proposed scheme for verifying the integrity of multiple data copies is elaborated in Section 3.5. Section 3.6 contains the security analysis of the proposed scheme. The performance analysis is shown in Section 3.7. Section 3.8 presents the implementation and experimental results. How to identify the corrupted copies is discussed in Section 3.9. A summary is given in Section 3.10.

3.1 Introduction

Storage-as-a-Service offered by cloud service providers (CSPs) enables customers to store and retrieve almost unlimited amount of data by paying fees metered in GB/month. For an increased level of scalability, availability and durability, some customers may want their data to be replicated on multiple servers across multiple data centers. The more copies the CSP is asked to store, the more fees are charged. Therefore, customers need to have a strong guarantee that the CSP is storing all data copies that are agreed upon in the service contract.

In this chapter, we propose a pairing-based provable multi-copy data possession (PB-PMDP) scheme, which provides an evidence that all outsourced copies are actually stored and remain intact. Moreover, it allows authorized users (*i.e.*, those who have the right to access the owner's file) to seamlessly access the file copies stored by the CSP, and supports public verifiability. The proposed scheme is proved to be secure against colluding servers. We illustrate the performance of the PB-PMDP scheme through theoretical analysis, which is then validated by experimental results. The verification time of the proposed scheme is practically independent of the number of file copies. Additionally, we discuss how to identify corrupted copies by slightly modifying the proposed PB-PMDP scheme.

3.2 Provable Multi-Copy Data Possession Schemes

In this section, we consider the case of provable possession for multiple data copies, for which we start with a basic provable multi-copy data possession scheme followed by a review of the scheme due to Curtmola *et al.* [37].

3.2.1 Basic Provable Multi-Copy Data Possession Scheme

Suppose that a CSP offers to store n copies of an owner's file on different servers for pre-specified fees according to the used storage space. Thus, the data owner needs a strong evidence to ensure that the CSP is actually storing no less than n copies, all these copies are complete and correct, and the owner is not paying for a service that he does not get. A straightforward solution to this problem is to use a single-copy PDP scheme to separately challenge and verify the integrity of each copy on each server. This is not a workable solution, since the CSP can convince the data owner that n copies of the file are stored, while there is only one copy. Whenever a request for a PDP scheme execution is made to one of the n severs, it is forwarded to the server which actually possesses the stored copy. The core of this cheating is that the n copies are identical making it trivial for the CSP to deceive the owner. Therefore, a step towards the solution is to leave the control of the file copying operation in the owner's hand to create unique *differentiable* copies.

In the basic provable multi-copy data possession scheme, the data owner creates n distinct copies by encrypting the file under n different keys. Hence, the CSP cannot use one copy to answer the challenges for another. This natural solution enables the verifier to separately challenge each copy on the remote servers, and ensure that the CSP is possessing not less than n copies.

Although the above basic scheme is a workable solution, it is impractical and has the following drawbacks:

- *Data access and key management* are serious problems with the basic scheme. Since the file is encrypted under n different keys, the owner has to keep these keys secret from the CSP, and share the n keys with each authorized user for each data file. Moreover, when an authorized user interacts with the CSP to retrieve the data file, it is not necessarily to receive the same copy each time. According to the load balancing mechanism used by the CSP to organize the work of the servers, the authorized user's request is directed to the server with the lowest congestion. Consequently, each copy should contain some indicator about its encryption key to enable the authorized user to properly decrypt and access the received copy.
- The computation and communication complexities of the verification task are linear with the number of copies.

3.2.2 Multiple-Replica Provable Data Possession Scheme

Curtmola *et al.* [37] were the first to present a multiple-replica provable data possession (MR-PDP) scheme that creates multiple copies of an owner's file and audit them. The MR-PDP scheme increases data availability; a corrupted data copy can be reconstructed using duplicated copies on other servers. The interaction between authorized users (those who have the right to access the owner's file) and the CSP was not considered in [37]. The MR-PDP scheme supports only private verifiability, *i.e.*, only the data owner can check data possession. Public verifiability is a key feature in remote data checking schemes to avoid disputes that may arise between the data owner and the CSP. Delegating the auditing process (without revealing secret keys) to a trusted third party for verifying the data integrity can resolve such disputes.

The MR-PDP scheme of [37] is based on the single-copy PDP model of [8]. In [37] distinct copies of a data files are created by first *encrypting* the file using one key, then *masking* the encrypted version (n times) with different randomness generated from a pseudo-random function.

Initially, a file F is fragmented into blocks $\{b_j\}_{1\leq j\leq m}$. The owner encrypts F using a key K to obtain an encrypted version $\widetilde{F} = \{\tilde{b}_j\}_{1\leq j\leq m}$, where $\tilde{b}_j = E_K(b_j)$. The owner generates n distinct copies $\{\widehat{F}_i\}_{1\leq i\leq n}$, where $\widehat{F}_i = \{\hat{b}_{ij}\}_{1\leq j\leq m}$, $\hat{b}_{ij} = \tilde{b}_j + r_{ij}$ (added as large integers in $\mathbb{Z}$), and $r_{ij} = f_x(i||j)$. f_x is a pseudo-random function keyed with a secret key x. Figure 3.1 gives a summary of the MR-PDP scheme.

In the MR-PDP scheme, if an authorized user interacts with the CSP to access an owner's file, the CSP retrieves one of the available copies. Upon receiving this copy, the authorized user has to know the copy index to properly *unmask* it before decryption. Due to the opaqueness of the internal operations of the CSP, the authorized users cannot recognize which copy has been received. If i (the copy index) is attached with each copy forming the structure $(i||\widehat{F}_i)$, corrupting or swapping copy indices hinder the correct unmasking process. Thus, the authorized users are unable to access the data file.

For verification purposes, portion of the set $\{r_{ij}\}$ is needed to be generated ($r_{chal} = \sum_{j\in A} r_{zj}$ in Figure 3.1). These random values cannot be publicly known, otherwise the CSP can derive the encrypted version $\widetilde{F}$, and store only one copy. Hence, only private verifiability is supported.

Setup

- File $F = \{b_j\}_{1 \leq j \leq m}$.
- $N = \acute{p}\acute{q}$ is the RSA modulus ($\acute{p}$ & $\acute{q}$ are prime numbers).
- $\acute{g}$ is a generator of QR_N (QR_N is the set of quadratic residues modulo N).
- Public key $pk = (N, \acute{g}, e)$, secret key $sk = (d, v, x)$, $v, x \in_R \mathbb{Z}_N$, and $ed \equiv 1 \bmod (\acute{p} - 1)(\acute{q} - 1)$.
- π_k is a pseudo-random permutation keyed with a key k.
- f_x is a pseudo-random function keyed with the secret key x.
- H is a hash function ($H : \{0, 1\}^* \rightarrow QR_N$).
- E_K is an encryption algorithm under a key K.

Data Owner

- Encrypts the data file F under the key K to obtain an encrypted version $\widetilde{F} = \{\tilde{b}_j\}_{1 \leq j \leq m}$, where $\tilde{b}_j = E_K(b_j)$
- Uses the encrypted version $\widetilde{F}$ to create a set of tags $\{T_j\}_{1 \leq j \leq m}$ for all copies: $T_j = (H(v||j) \cdot \acute{g}^{\tilde{b}_j})^d \bmod N$
- Generates n distinct copies $\{\widehat{F}_i\}_{1 \leq i \leq n}$, $\widehat{F}_i = \{\hat{b}_{ij}\}_{1 \leq j \leq m}$ utilizing random masking:

```
for $i = 1$ to $n$ do
    for $j = 1$ to $m$ do
        1. Computes a random value $r_{ij} = f_x(i||j)$
        2. Computes the replica's block $\hat{b}_{ij} = \tilde{b}_j + r_{ij}$ (added as large integers in $\mathbb{Z}$)
```

- Sends the copy $\widehat{F}_i$ to a server S_i, $i :\ 1 \rightarrow\ n$

$\Longrightarrow$ ***continue***

Checking possession of a replica $\widehat{F}_z$

Owner — Remote Server S_z

1. Picks a key k for the function π, c (# of blocks to be challenged), and $\acute{g}_s = \acute{g}^s \bmod N$ ($s \in_R \mathbb{Z}_N$)

$$\xrightarrow{c,\, k,\, \acute{g}_s}$$

2. Computes a set A of random indices: $A = \{j\} = \pi_k(l)_{1 \leq l \leq c}$
3. Computes $T = \prod_{j \in A} T_j \bmod N$
4. Computes $\rho = \acute{g}_s^{\sum_{j \in A} \hat{b}_{zj}} \bmod N$

$$\xleftarrow{T,\, \rho}$$

5. Computes $A = \{j\} = \pi_k(l)_{1 \leq l \leq c}$
6. Checks $\left(\frac{T^e}{\prod_{j \in A} H(v||j)} \cdot \acute{g}^{r_{chal}}\right)^s \stackrel{?}{=} \rho$, where $r_{chal} = \sum_{j \in A} r_{zj}$

Figure 3.1: The MR-PDP scheme by Curtmola *et al.* [37].

3.3 Our System and Assumptions

System components. The cloud computing storage model considered in this work consists of three main components as illustrated in Figure 3.2: (i) a data owner that can be an individual or an organization originally possessing sensitive data to be stored in the cloud; (ii) a CSP who manages cloud servers and provides paid storage space on its infrastructure to store the owner's files; and (iii) authorized users – a set of owner's clients who have the right to access the remote data.

The storage model used in this work can be adopted by many practical applications. For example, e-Health applications can be envisioned by this model where the patients' database that contains large and sensitive information can be stored on cloud servers. In these types of applications, the e-Health organization can be considered as the data owner, and the physicians as the authorized users who have the right to access the patients'

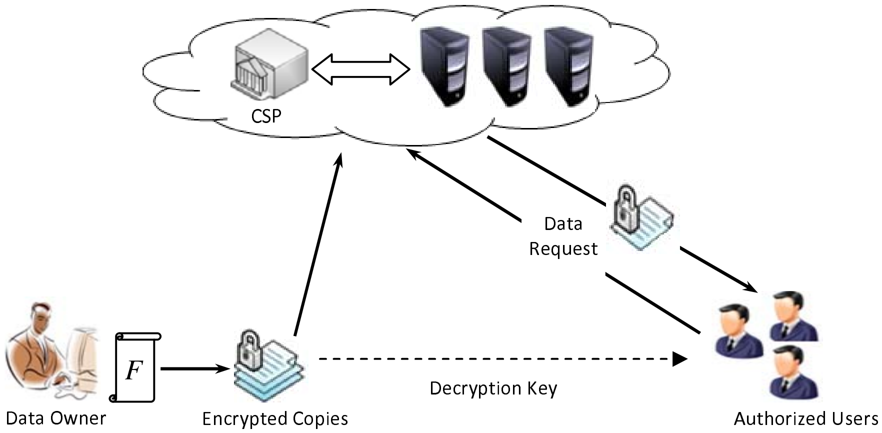


Figure 3.2: Cloud computing data storage system model.

medical history. Many other practical applications like financial, scientific, and educational applications can be viewed in similar settings.

In this work, we focus on *sensitive* archived and *warehoused* data, which is essential in many applications such as digital libraries and astronomical/medical/scientific/legal repositories. Such data are subject to infrequent change, so we treat them as static.

Data Redundancy. Data redundancy can be achieved using replication or coding schemes, where the former is the simplest way that can be adopted by many storage systems. For a data file with size $|F|$ bits, the storage cost for n copies over cloud servers is $n|F|$ bits. In erasure codes, the file is divided into m blocks and encoded into ℓ blocks, where $\ell > m$ [3]. The encoded blocks are stored at ℓ different servers (one code block per server to prevent simultaneous failure of all blocks), and thus the storage cost is $\frac{|F|}{m}\ell$ bits. The original file can be reconstructed from any m out of the ℓ servers.

In the context of this work, we are considering economically-motivated CSPs that may attempt to use less storage than required by the service contract through deletion of a few copies of the file. The CSPs have almost no financial benefit by deleting only a small portion of a copy of the file. Redundancy using erasure codes has less storage cost; however, duplicating data file across multiple servers achieves scalability in the sense that if the number of users grows, then with more copies of data the user access time can be kept below a certain threshold. Such scalability is a fundamental customer requirement in cloud computing systems. A file that is duplicated and stored strategically on multiple servers – located at various geographic locations – can help reduce access time and communication cost for users. On the other hand, in responding to a data access request for coding-based systems, the CSP has to access at least m servers to reconstruct the original data file, and thus increased time overhead (network latency and computation time to decode data blocks) occurs at the CSP side.

More importantly, in case of data corruption, erasure codes require the precise identification of failed/corrupted blocks. Without the ability to identify which blocks have been corrupted, there is potentially a factorial combination of blocks to try to reconstruct the original data file; that is $\binom{n}{\ell}$. For replication-based systems, a server's copy can be reconstructed even from a complete damage using duplicated copies on other servers. As a result of the aforementioned reasons, in our work we do not apply erasure codes to the data file before outsourcing.

Outsourcing and accessing. The data owner has a file F consisting of m blocks and the CSP offers to store n copies $\{\widetilde{F}_1, \widetilde{F}_2, \ldots, \widetilde{F}_n\}$ of the owner's file on different servers – to prevent simultaneous failure of all copies – in exchange for pre-specified fees metered in GB/month. The number of copies depends on the nature of data; more copies are needed for critical data that cannot easily be reproduced, and to achieve a higher level of scalability. This critical data should be replicated on multiple servers across multiple data centers. On the other hand, non-critical, reproducible data are stored at reduced levels of redundancy. The CSP pricing model is related to the number of data copies.

For data confidentiality, the owner encrypts his data before outsourcing to the CSP. An authorized user of the outsourced data sends a data-access request to the CSP and receives a file copy in an encrypted form that can be decrypted using a secret key shared with the owner. According to the load balancing mechanism used by the CSP to organize the work of the servers, the data-access request is directed to the server with the lowest congestion, and thus the authorized user is not aware of which copy has been received.

We assume that the interaction between the owner and the authorized users to authenticate their identities and share the secret key has already been completed, and it is not considered in this work. Throughout this chapter, the terms cloud server and cloud service provider are used interchangeably.

Threat model. The completeness and correctness of customers' data in the cloud may be at risk due to the following reasons. First, the CSP – whose goal is likely to make a profit and maintain a reputation – has an incentive to hide data loss (due to hardware failure, management errors, various attacks) or reclaim storage by discarding data that has not been or is rarely accessed. Second, a dishonest CSP may store fewer copies than what has been agreed upon in the service contact with the data owner, and try to convince the owner that all copies are correctly stored intact. Third, the cloud infrastructures are subject to a wide range of internal and external security threats. Incidences of security breaches of cloud services surface from time to time [51, 58].

In short, although outsourcing data to the cloud is attractive from the view point of cost and complexity of long-term large-scale data storage, it does not offer sufficient guarantee on data integrity. This problem, if not properly handled, may hinder the successful deployment and wide acceptance of the cloud paradigm. The goal of the proposed scheme is to detect (with *high probability*) the CSP misbehavior by validating the number and integrity of file copies.

Underlying algorithms. The proposed scheme consists of five polynomial time algorithms: KeyGen, CopyGen, TagGen, Prove, and Verify.

- $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\kappa)$. This algorithm is run by the data owner. It takes as input a security parameter 1^κ, and returns a public key pk (publicly known) and a private key sk (kept secret by the owner).
- $\widetilde{\mathbb{F}} \leftarrow \mathsf{CopyGen}(CN_i, F)_{1\leq i\leq n}$. This algorithm is run by the data owner. It takes as input a copy number CN_i and a file F, and generates n copies $\widetilde{\mathbb{F}} = \{\widetilde{F_i}\}_{1\leq i\leq n}$. The owner sends the copies $\widetilde{\mathbb{F}}$ to the CSP to be stored on cloud servers.

– $\Phi \leftarrow \mathsf{TagGen}(sk, \widetilde{\mathbb{F}})$. This algorithm is run by the data owner. It takes as input the private key sk and the file copies $\widetilde{\mathbb{F}}$, and outputs tags/authenticators set Φ, which is an ordered collection of tags for the data blocks. The owner sends Φ to the CSP to be stored along with the copies $\widetilde{\mathbb{F}}$.

– $\mathbb{P} \leftarrow \mathsf{Prove}(\widetilde{\mathbb{F}}, \Phi, chal)$. This algorithm is run by the CSP. It takes as input the file copies $\widetilde{\mathbb{F}}$, the tags set Φ, and a challenge $chal$ (sent from a verifier). It returns a proof $\mathbb{P}$, which guarantees that the CSP is actually storing n copies and all these copies are intact.

– $\{1, 0\} \leftarrow \mathsf{Verify}(pk, \mathbb{P})$. This algorithm is run by a verifier (original owner or any other trusted auditor). It takes as input the public key pk, and the proof $\mathbb{P}$ returned from the CSP. The output is 1 if the integrity of all file copies is correctly verified or 0 otherwise.

3.4 Security Model

Following [81], we would like the remote data checking scheme to be *correct* and *sound*. These two requirements are defined as follows:

- Correctness requires that the verifier accepts valid server responses.
- Soundness requires that any cheating server that passes the verification process is actually storing the owner's data intact.

The security of the proposed scheme can be stated using a "game" that captures the data possession property [8, 42, 81]. The data possession game between an adversary $\mathcal{A}$ (acts as a malicious CSP) and a challenger $\mathcal{C}$ (acts as a verifier) consists of the following:

- SETUP. $\mathcal{C}$ runs the KeyGen algorithm to generate a key pair (pk, sk), and sends pk to $\mathcal{A}$.

- INTERACT. $\mathcal{A}$ interacts with $\mathcal{C}$ to get the file copies and the verification tags set Φ. $\mathcal{A}$ adaptively selects a file F and sends it to $\mathcal{C}$. $\mathcal{C}$ runs the two algorithms $\mathsf{CopyGen}$ and TagGen to create n distinct copies $\widetilde{\mathbb{F}}$ along with the tags set Φ, and returns both $\widetilde{\mathbb{F}}$ and Φ to $\mathcal{A}$.

  Moreover, $\mathcal{A}$ can request challenges $\{chal_i\}_{1 \leq i \leq L}$ for some parameter $L \geq 1$ of his choice, and return proofs $\{\mathbb{P}_i\}_{1 \leq i \leq L}$ to $\mathcal{C}$. $\mathcal{C}$ runs the Verify algorithm and provides the verification results to $\mathcal{A}$. The INTERACT step between $\mathcal{A}$ and $\mathcal{C}$ can be repeated polynomially-many times.

- CHALLENGE. $\mathcal{A}$ decides on a file F previously used during the INTERACT step, requests a challenge $chal$ from $\mathcal{C}$, and generates a proof $\mathbb{P} \leftarrow \mathsf{Prove}(\widetilde{\mathbb{F}}', \Phi, chal)$, where $\widetilde{\mathbb{F}}'$ is $\widetilde{\mathbb{F}}$ except that at least one of its file copies (or a portion of it) is missing or tampered with. Upon receiving the proof $\mathbb{P}$, $\mathcal{C}$ runs the Verify algorithm and if $\mathsf{Verify}(pk, \mathbb{P})$ returns 1, then $\mathcal{A}$ has won the game. The CHALLENGE step can be repeated polynomially-many times for the purpose of data extraction.

The proposed scheme is secure if the probability that any polynomial-time adversary $\mathcal{A}$ wins the game is negligible. In other words, if a polynomial-time adversary $\mathcal{A}$ can win the game with non-negligible probability, then there exists a polynomial-time extractor that can repeatedly execute the CHALLENGE step until it extracts the blocks of data copies.

File swapping attack. In this type of attacks, the remote server tries to prove the possession of the data using blocks from different files. A remote data checking scheme must be secure against such an attack.

3.5 Proposed PB-PMDP Scheme

3.5.1 Overview and Rationale

Generating unique differentiable copies of the data file is the core to design a provable multi-copy data possession scheme. Identical data copies enable the CSP to simply deceive the owner by storing only one copy and pretending that it stores multiple copies. Using a simple yet *efficient* way, the proposed scheme generates distinct copies utilizing the *diffusion* property of any secure encryption scheme. The diffusion property ensures that the output bits of the ciphertext depend on the input bits of the plaintext in a very complex way, *i.e.*, there will be an unpredictable complete change in the ciphertext, if there is a single bit change in the plaintext [35]. The interaction between the authorized users and the CSP is considered through this methodology of generating distinct copies, where the former can decrypt and access a file copy received from the CSP. In the proposed scheme, the authorized users need only to keep a single secret key – shared with the data owner – to decrypt the file copy, and it is not necessarily to recognize the index of the received copy.

3.5.2 Notations

- F is a data file to be outsourced, and is composed of a sequence of m blocks, *i.e.*, $F = \{b_1, b_2, \dots, b_m\}$.
- $\pi_{key}(\cdot)$ is a pseudo-random permutation (PRP): $key \times \{0,1\}^{\log_2(m)} \to \{0,1\}^{\log_2(m)}$.
- $\psi_{key}(\cdot)$ is a pseudo-random function (PRF): $key \times \{0,1\}^* \to \mathbb{Z}_p$ (p is a prime number).
- **Bilinear Map/Pairing**. Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups of prime order p. Let $\bar{g}$ and g be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. A bilinear pairing is a map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following properties [65, 25]:

  1. *Bilinear*: $\hat{e}(u^a, v^b) = \hat{e}(u, v)^{ab}\ \forall\ u \in \mathbb{G}_1, v \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_p$
  2. *Non-degenerate*: $\hat{e}(\bar{g}, g) \neq 1$
  3. *Computable*: there exists an efficient algorithm for computing $\hat{e}$.

- $\mathcal{H}(\cdot)$ is a map-to-point hash function : $\{0,1\}^* \to \mathbb{G}_1$.
- E_K is an encryption algorithm with strong *diffusion* property and a key K, *e.g.*, AES (Advanced Encryption Standard) [38].

Remark 1. Homomorphic linear authenticators (HLAs) [81, 41, 10] are basic building blocks in the proposed scheme. Informally, the HLA is a fingerprint/tag computed by the owner for each file block b_j that enables a verifier to validate the data possession on remote servers by sending a challenge vector $chal$ of c elements: $chal = \{r_1, r_2, \ldots, r_c\}$. As a response, the servers can homomorphically construct a tag authenticating the value $\sum_{j=1}^{c} r_j \cdot b_j$. The response is validated by a verifier, and accepted only if the servers honestly compute the response using the owner's file blocks. The proposed scheme in this work utilizes the BLS (Boneh-Lynn-Shacham) HLAs [81].

3.5.3 PB-PMDP Procedural Steps

- ***Key Generation***. As earlier, $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a bilinear map and g is a generator of $\mathbb{G}_2$. The data owner runs the KeyGen algorithm to generate a private key $x \in \mathbb{Z}_p$ and a public key $y = g^x \in \mathbb{G}_2$ along with s elements $(u_1, u_2, \ldots, u_s) \in_R \mathbb{G}_1$.

- ***Generation of Distinct Copies***. The data owner runs the CopyGen algorithm to create n differentiable copies $\widetilde{\mathbb{F}} = \{\widetilde{F}_i\}_{1 \leq i \leq n}$. The copy $\widetilde{F}_i$ is generated by concatenating a copy number i with the file F, then encrypting using E_K, *i.e.*, $\widetilde{F}_i = E_K(i||F)$. $\widetilde{F}_i$ is divided into blocks $\{\tilde{b}_{ij}\}_{1 \leq j \leq m}$, and the block $\tilde{b}_{ij}$ is further fragmented into s sectors $\{\tilde{b}_{ij1}, \tilde{b}_{ij2}, \ldots, \tilde{b}_{ijs}\}$, *i.e.*, the copy $\widetilde{F}_i = \{\tilde{b}_{ijk}\}_{\substack{1 \leq j \leq m \\ 1 \leq k \leq s}}$, where each sector $\tilde{b}_{ijk} \in \mathbb{Z}_p$ for some large prime p.

  The authorized users need to keep only a single secret key K. Later, when an authorized user receives a file copy from the CSP, he decrypts the copy and removes the index from the copy header to reconstruct the plain form of the received file copy.

- ***Generation of Tags***. Given the distinct file copies $\widetilde{\mathbb{F}} = \{\widetilde{F}_i\}$, where $\widetilde{F}_i = \{\tilde{b}_{ijk}\}$, the data owner runs the TagGen algorithm to generate a tag σ_{ij} for each block $\tilde{b}_{ij}$ as $\sigma_{ij} = (\mathcal{H}(ID_F||j).\prod_{k=1}^{s} u_k^{\tilde{b}_{ijk}})^x \in \mathbb{G}_1$ $(i : 1 \rightarrow n,\ j : 1 \rightarrow m,\ k : 1 \rightarrow s)$. In the tag computation, $ID_F = Filename||n||m||u_1||\ldots||u_s$ is a unique fingerprint for each file F comprising the file name, the number of copies for this file, the number of blocks per copy, and the random values $\{u_k\}_{1\leq k\leq s}$. Embedding the ID_F into the block tag σ_{ij} prevents the CSP from cheating by using blocks from different files (*file swapping attack*).

In order to reduce storage overhead on cloud servers and lower communication cost, the data owner generates an aggregated tag σ_j for the blocks at the same indices in each copy $\widetilde{F}_i$ as $\sigma_j = \prod_{i=1}^{n}\sigma_{ij} \in \mathbb{G}_1$. Hence, instead of storing mn tags, the proposed PB-PMDP scheme requires the CSP to store only m tags for the files copies $\widetilde{\mathbb{F}}$. Let us denote the set of aggregated tags as $\Phi = \{\sigma_j\}_{1\leq j\leq m}$. The data owner sends $\{\widetilde{\mathbb{F}}, \Phi, ID_F\}$ to the CSP, and deletes the copies and the tags from its local storage.

- ***Challenge***. For challenging the CSP and validating the integrity of all copies, the verifier sends c (# of blocks to be challenged) and two fresh keys at each challenge: a PRP(π) key k_1 and a PRF(ψ) key k_2. *Both* the verifier and the CSP use π keyed with k_1 and the ψ keyed with k_2 to generate a set $Q = \{(j, r_j)\}$ of c pairs of random indices and random values, where $\{j\} = \pi_{k_1}(l)_{1\leq l\leq c}$ and $\{r_j\} = \psi_{k_2}(l)_{1\leq l\leq c}$.

- ***Response***. The CSP runs the Prove algorithm to generate a set $Q = \{(j, r_j)\}$ of random indices and values, and provide an evidence that the CSP is still correctly possessing the n copies. The CSP responds with a proof $\mathbb{P} = \{\sigma, \mu\}$, where

$$\sigma = \prod_{(j,r_j)\in Q}\sigma_j^{r_j} \in \mathbb{G}_1,\ \ \mu = \{\mu_{ik}\}_{\substack{1\leq i\leq n,\\ 1\leq k\leq s}}\ \text{and}\ \mu_{ik} = \sum_{(j,r_j)\in Q} r_j\cdot\tilde{b}_{ijk} \in \mathbb{Z}_p.$$

- ***Verify Response***. Upon receiving the proof $\mathbb{P} = \{\sigma, \mu\}$ from the CSP, the verifier runs the Verify algorithm to check the following verification equation:

$$\hat{e}(\sigma, g) \overset{?}{=} \hat{e}([\prod_{(j,r_j)\in Q}\mathcal{H}(ID_F||j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n}\mu_{ik}}, y). \tag{3.1}$$

In equation (3.1), the term $\sum_{i=1}^{n} \mu_{ik}$ is linear in n, while the term $[\cdot]^n$ costs one more exponentiation for any value of n. If the verification equation passes, the Verify algorithm returns 1, otherwise 0. The correctness of verification equation (3.1) can be shown as follows:

$$\begin{aligned}
\hat{e}(\sigma, g) &= \hat{e}(\prod_{(j,r_j)\in Q} \sigma_j^{r_j}, g) \\
&= \hat{e}(\prod_{(j,r_j)\in Q} [\prod_{i=1}^{n} \sigma_{ij}]^{r_j}, g) \\
&= \hat{e}(\prod_{(j,r_j)\in Q} [\prod_{i=1}^{n} (\mathcal{H}(ID_F||j) \cdot \prod_{k=1}^{s} u_k^{\tilde{b}_{ijk}})^x]^{r_j}, g) \\
&= \hat{e}(\prod_{(j,r_j)\in Q} [\prod_{i=1}^{n} \mathcal{H}(ID_F||j) \cdot \prod_{i=1}^{n}\prod_{k=1}^{s} u_k^{\tilde{b}_{ijk}}]^{r_j}, y) \\
&= \hat{e}(\prod_{(j,r_j)\in Q} \prod_{i=1}^{n} \mathcal{H}(ID_F||j)^{r_j} \cdot \prod_{(j,r_j)\in Q} \prod_{i=1}^{n}\prod_{k=1}^{s} u_k^{r_j \cdot \tilde{b}_{ijk}}, y) \\
&= \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n} \sum_{(j,r_j)\in Q} r_j \cdot \tilde{b}_{ijk}}, y) \\
&= \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n} \mu_{ik}}, y).
\end{aligned}$$

Remark 2. The proposed PB-PMDP scheme supports public verifiability where anyone, who knows the owner's public key but is not necessarily the data owner, can send a challenge vector to the CSP and verify the response. Public verifiability can resolve disputes that may arise between the data owner and the CSP regarding data integrity. If such a dispute occurs, a trusted third party auditor (TPA) can determine whether the data integrity is maintained or not. Since the owner's public key is only needed to perform the verification step, the owner is not required to reveal his secret key to the TPA. The PB-PMDP scheme is presented in Figure 3.3.

Setup

- File $F = \{b_1, b_2, \ldots, b_m\}$.
- $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a bilinear map, g is a generator for $\mathbb{G}_2$.
- $x \in \mathbb{Z}_p$ is a private key.
- $y = g^x \in \mathbb{G}_2$ along with $(u_1, u_2, \ldots, u_s) \in_R \mathbb{G}_1$ form a public key.

Data Owner

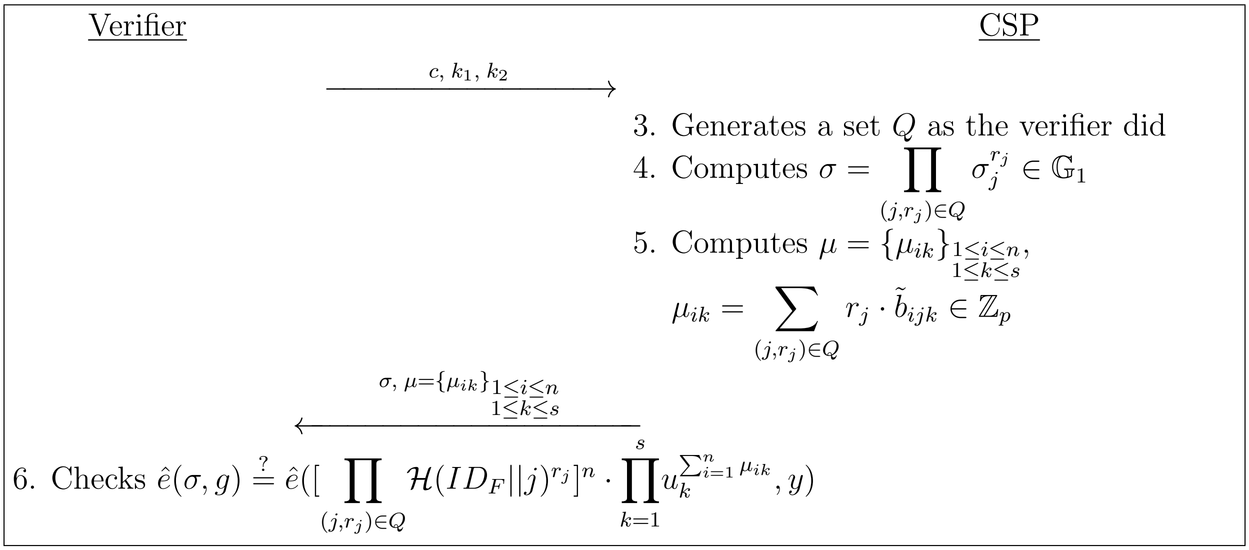
- Creates distinct file copies $\widetilde{\mathbb{F}} = \{\widetilde{F}_i\}_{1 \leq i \leq n}$, where $\widetilde{F}_i = E_K(i||F)_{1 \leq i \leq n}$. Each copy $\widetilde{F}_i$ is an ordered collection of blocks fragmented into sectors, *i.e.*, $\widetilde{F}_i = \{\tilde{b}_{ijk}\}_{\substack{1 \leq j \leq m \\ 1 \leq k \leq s}}$, where $\tilde{b}_{ijk} \in \mathbb{Z}_p$.
- Calculates the block tag $\sigma_{ij} = (\mathcal{H}(ID_F||j).\prod_{k=1}^{s} u_k^{\tilde{b}_{ijk}})^x \in \mathbb{G}_1$.
- Computes a set of aggregated tags $\Phi = \{\sigma_j\}_{1 \leq j \leq m}$ for the blocks at the same indices in each copy $\widetilde{F}_i$, where $\sigma_j = \prod_{i=1}^{n} \sigma_{ij} \in \mathbb{G}_1$.
- Sends $\{\widetilde{\mathbb{F}}, \Phi, ID_F\}$ to the CSP and deletes the copies and the tags from its local storage.

Challenge Response

Verifier CSP

1. Picks c (# of blocks to be challenged) and two fresh keys k_1 and k_2
2. Generates a set $Q = \{(j, r_j)\}$, $\{j\} = \pi_{k_1}(l)_{1 \leq l \leq c}$ and $\{r_j\} = \psi_{k_2}(l)_{1 \leq l \leq c}$

$\Longrightarrow$ ***continue***



Verifier

CSP

$\xrightarrow{c,\, k_1,\, k_2}$

3. Generates a set Q as the verifier did
4. Computes $\sigma = \prod_{(j,r_j)\in Q} \sigma_j^{r_j} \in \mathbb{G}_1$
5. Computes $\mu = \{\mu_{ik}\}_{\substack{1\leq i\leq n,\\ 1\leq k\leq s}}$

$\mu_{ik} = \sum_{(j,r_j)\in Q} r_j \cdot \tilde{b}_{ijk} \in \mathbb{Z}_p$

$\xleftarrow{\sigma,\ \mu=\{\mu_{ik}\}_{\substack{1\leq i\leq n\\ 1\leq k\leq s}}}$

6. Checks $\hat{e}(\sigma, g) \overset{?}{=} \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n}\mu_{ik}}, y)$

Figure 3.3: The proposed PB-PMDP scheme.

3.5.4 Reducing the Communication Cost

One can attempt to change the PB-PMDP scheme to reduce the communication cost by a factor of n by permitting the CSP to compute and send $\mu = \{\hat{\mu}_k\}_{1\leq k\leq s}$, where $\hat{\mu}_k = \sum_{i=1}^{n} \mu_{ik}$. However, this modification enables the CSP to simply cheat the verifier as follows:

$$\hat{\mu}_k = \sum_{i=1}^{n} \mu_{ik} = \sum_{i=1}^{n} \sum_{(j,r_j)\in Q} r_j \cdot \tilde{b}_{ijk} = \sum_{(j,r_j)\in Q} r_j \cdot \sum_{i=1}^{n} \tilde{b}_{ijk}.$$

Thus, the CSP can just keep the sectors summation $\sum_{i=1}^{n} \tilde{b}_{ijk}$ not the sectors themselves. Moreover, the CSP can corrupt the block sectors and the summation is still valid. Therefore, the proposed scheme requires the CSP to send $\mu = \{\mu_{ik}\}_{\substack{1\leq i\leq n\\ 1\leq k\leq s}}$, and the summation $\sum_{i=1}^{n} \mu_{ik}$ is done on the verifier side.

A slightly modified version of the PB-PMDP scheme can reduce the communication cost by a factor of s during the response phase by allowing the CSP to compute and send $\mu = \{\mu_i\}_{1\leq i\leq n}$ instead of $\mu = \{\mu_{ik}\}_{\substack{1\leq i\leq n\\ 1\leq k\leq s}}$. In this version, a file copy $\widetilde{F}_i$ is divided into blocks, but the blocks are not fragmented into sectors, *i.e.*, a copy $\widetilde{F}_i = \{\tilde{b}_{ij}\}_{1 \leq j \leq m}$, where $\tilde{b}_{ij} \in \mathbb{Z}_p$. A tag σ_{ij} is generated for each block $\tilde{b}_{ij}$: $\sigma_{ij} = (\mathcal{H}(ID_F||j).u^{\tilde{b}_{ij}})^x \in \mathbb{G}_1$, where u is a generator for $\mathbb{G}_1$. Tags are aggregated into a set $\Phi = \{\sigma_j\}_{1 \leq j \leq m}$, where $\sigma_j = \prod_{i=1}^{n} \sigma_{ij}$. In this scenario, the CSP responds with $\sigma = \prod_{(j,r_j) \in Q} \sigma_j^{r_j} \in \mathbb{G}_1$ and $\mu = \{\mu_i\}_{1 \leq i \leq n}$, where $\mu_i = \sum_{(j,r_j) \in Q} r_j \cdot \tilde{b}_{ij} \in \mathbb{Z}_p$. The verification equation (3.1) will be modified to $\hat{e}(\sigma, g) \overset{?}{=} \hat{e}([\prod_{(j,r_j) \in Q} \mathcal{H}(ID_F||j)^{r_j}]^n \cdot u^{\sum_{i=1}^{n} \mu_i}, y)$.

This reduced communication cost will be at the expense of increased storage overhead on the CSP side, where each block $\tilde{b}_{ij} \in \mathbb{Z}_p$ will be accompanied with a tag $\sigma_{ij} \in \mathbb{G}_1$ of equal length. If the block size is greater than $|p|$ (the bit length of the prime p), the CSP can simply cheat by storing $\tilde{b}_{ij}$ mod p instead of the whole block $\tilde{b}_{ij}$. Therefore, with this slightly modified version, to store n copies each of size $|F|$ bits, the total storage over the CSP will be $(n+1)|F|$ bits (using tag aggregation approach). The storage overhead equals the size of a complete file copy. The more storage space is used over the CSP side, the more fees the customers are charged (pay-as-you-go pricing model).

3.6 Security Analysis

Here we present the security analysis for the PB-PMDP scheme. First, in the proposed scheme, we utilize PRP (π) and PRF (ψ) to compress the challenge, and thus reducing the communication cost. Instead of sending the set Q of c pairs of random indices and values to the CSP, the verifier sends only two keys k_1 and k_2 (over secure communication). Using π and ψ in this manner is proved to be secure [10].

For the *correctness* security requirement, we have previously shown the correctness of equation (3.1). For the *soundness* security requirement, we will show that if a polynomial-time adversary $\mathcal{A}$ can win the data possession game (with non-negligible probability) with a challenger $\mathcal{C}$, then $\mathcal{A}$ is actually storing the n data copies $\widetilde{\mathbb{F}}$ in an uncorrupted state. For an adversary $\mathcal{A}$ to cheat the verifier, he has to respond with a malicious proof $\mathbb{P}' \neq \mathbb{P}$ and $\mathsf{Verify}(pk, \mathbb{P}')$ returns 1.

The soundness of the PB-PMDP scheme is based on the *unforgeability* of the used

HLAs, which depends on the security of the computational Diffie-Hellman (CDH) and the discrete logarithm (DL) problems.

Definitions.

1. **CDH problem:** given $g,\ g^x,\ h \in \mathbb{G}$ for some group $\mathbb{G}$ and $x \in \mathbb{Z}_p$, compute h^x
2. **DL problem:** given $g,\ h \in \mathbb{G}$ for some group $\mathbb{G}$, find x such that $h = g^x$.

The following theorem proves the unforgeability of the HLAs used in the proposed PB-PMDP scheme. Our approach to prove the theorem is by investigating all possible combinations of malicious CSP responses $\langle\{\sigma', \mu'\}, \{\sigma, \mu'\}, \{\sigma', \mu\}\rangle$, and checking whether any of these combinations can pass the verification equation (3.1).

Theorem 1. *Assuming the hardness of both the CDH and the DL problems in bilinear groups, the verifier of the proposed PB-PMDP scheme accepts a response to a challenge vector only if a correctly computed proof* $\mathbb{P} = \{\sigma,\ \mu\}$*, where* $\mu = \{\mu_{ik}\}_{\substack{1\le i\le n\\1\le k\le s}}$ *is sent from the CSP.*

Proof. We prove the theorem by contradiction. The goal of an adversary $\mathcal{A}$ (malicious CSP) is to generate a response that is not correctly computed and pass the verification process done by a challenger $\mathcal{C}$ (verifier). Let $\mathbb{P}' = \{\sigma',\ \mu'\}$ be $\mathcal{A}$'s response, where $\mu' = \{\mu'_{ik}\}_{\substack{1\le i\le n\\1\le k\le s}}$. Let $\mathbb{P} = \{\sigma,\ \mu\}$ be the expected response from an honest CSP, where $\sigma = \prod_{(j,r_j)\in Q} \sigma_j^{r_j}$, $\mu = \{\mu_{ik}\}_{\substack{1\le i\le n\\1\le k\le s}}$, and $\mu_{ik} = \sum_{(j,r_j)\in Q} r_j \cdot \tilde{b}_{ijk}$.

According to the correctness of PB-PMDP scheme, the expected proof $\mathbb{P} = \{\sigma,\ \mu\}$ satisfies the verification equation, *i.e.*,

$$\hat{e}(\sigma, g) = \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^n \mu_{ik}}, y).$$

Assume that $\sigma' \neq \sigma$, and σ' passes the verification equation, then we have

$$\hat{e}(\sigma', g) = \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^n \mu'_{ik}}, y).$$

Obviously, if $\mu'_{ik} = \mu_{ik}\ \forall(i,k)$, it follows from the above verification equations that $\sigma' = \sigma$, which contradicts our assumption. Let us define $\Delta\mu_{ik} = \mu'_{ik} - \mu_{ik}$ $(1 \leq i \leq n, 1 \leq k \leq s)$. It must be the case that at least one of $\{\Delta\mu_{ik}\}_{\substack{1\leq i\leq n\\ 1\leq k\leq s}}$ is nonzero. Dividing the verification equation for the malicious response by the verification equation for the expected response, we obtain

$$\begin{aligned}
\hat{e}(\sigma' \cdot \sigma^{-1}, g) &= \hat{e}(\prod_{k=1}^{s} u_k^{\sum_{i=1}^{n} \Delta\mu_{ik}}, y) \\
\hat{e}(\sigma' \cdot \sigma^{-1}, g) &= \hat{e}(\prod_{k=1}^{s} u_k^{x\cdot\sum_{i=1}^{n} \Delta\mu_{ik}}, g) \\
\sigma' \cdot \sigma^{-1} &= \prod_{k=1}^{s} u_k^{x\cdot\sum_{i=1}^{n} \Delta\mu_{ik}}.
\end{aligned}$$

We set $u_k = g^{\alpha_k} h^{\beta_k}$ for $\alpha_k, \beta_k \in \mathbb{Z}_p$, and thus

$$\begin{aligned}
\sigma' \cdot \sigma^{-1} &= \prod_{k=1}^{s} (g^{\alpha_k} h^{\beta_k})^{x\cdot\sum_{i=1}^{n} \Delta\mu_{ik}} \\
\sigma' \cdot \sigma^{-1} &= \prod_{k=1}^{s} (y^{\alpha_k} h^{x.\beta_k})^{\sum_{i=1}^{n} \Delta\mu_{ik}} \\
\sigma' \cdot \sigma^{-1} &= y^{\sum_{k=1}^{s} \alpha_k\cdot\sum_{i=1}^{n} \Delta\mu_{ik}} \,.\, h^{x\cdot\sum_{k=1}^{s} \beta_k\cdot\sum_{i=1}^{n} \Delta\mu_{ik}} \\
h^x &= (\sigma' \cdot \sigma^{-1} \cdot y^{-\sum_{k=1}^{s} \alpha_k\cdot\sum_{i=1}^{n} \Delta\mu_{ik}})^{\frac{1}{\sum_{k=1}^{s} \beta_k\cdot\sum_{i=1}^{n} \Delta\mu_{ik}}}.
\end{aligned}$$

Hence, we have found a solution to the CDH problem unless evaluating the exponent causes a division by zero. However, we noted that not all of $\{\Delta\mu_{ik}\}_{\substack{1\leq i\leq n\\ 1\leq k\leq s}}$ can be zero and the probability that $\beta_k = 0$ is $\frac{1}{p}$, which is negligible. Therefore, if $\sigma' \neq \sigma$, we can use the adversary $\mathcal{A}$ to break the CDH problem, and thus we guarantee that σ' must be equal to σ.

It is only the values $\mu' = \{\mu'_{ik}\}_{\substack{1\leq i\leq n\\ 1\leq k\leq s}}$ and $\mu = \{\mu_{ik}\}_{\substack{1\leq i\leq n\\ 1\leq k\leq s}}$ that can differ. Assume that the adversary $\mathcal{A}$ responds with $\sigma' = \sigma$ and $\mu' \neq \mu$. Now we have

$$\hat{e}(\sigma, g) = \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n} \mu_{ik}}, y) = \hat{e}(\sigma', g) =$$
$$\hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n} \mu'_{ik}}, y),$$

from which we conclude that

$$\hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n} \mu_{ik}}, y) = \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n} \mu'_{ik}}, y).$$

Thus,

$$\begin{aligned} 1 &= \hat{e}(\prod_{k=1}^{s} u_k^{\sum_{i=1}^{n} \Delta\mu_{ik}}, y) \\ 1 &= \prod_{k=1}^{s} (g^{\alpha_k} h^{\beta_k})^{\sum_{i=1}^{n} \Delta\mu_{ik}} \\ 1 &= g^{\sum_{k=1}^{s} \alpha_k \cdot \sum_{i=1}^{n} \Delta\mu_{ik}} \cdot h^{\sum_{k=1}^{s} \beta_k \cdot \sum_{i=1}^{n} \Delta\mu_{ik}} \\ h &= g^{-\frac{\sum_{k=1}^{s} \alpha_k \cdot \sum_{i=1}^{n} \Delta\mu_{ik}}{\sum_{k=1}^{s} \beta_k \cdot \sum_{i=1}^{n} \Delta\mu_{ik}}}. \end{aligned}$$

Now, we have found a solution to the DL problem unless evaluating the exponent causes a division by zero. However, the probability that $\beta_k = 0$ is $\frac{1}{p}$, which is negligible. Therefore, if there is at least one difference between $\{\mu'_{ik}\}_{\substack{1\leq i\leq n \\ 1\leq k\leq s}}$ and $\{\mu_{ik}\}_{\substack{1\leq i\leq n \\ 1\leq k\leq s}}$, we can use the adversary $\mathcal{A}$ to break the DL problem. As a result, we guarantee that $\{\mu'_{ik}\}$ must be equal to $\{\mu_{ik}\}$ $\forall(i, k)$. □

Data Extraction. We have shown that if a polynomial-time adversary $\mathcal{A}$ can win the data possession game (with non-negligible probability) with a challenger $\mathcal{C}$, then $\mathcal{A}$ is actually storing the data in an uncorrupted state. For the purpose of data extraction, the challenger $\mathcal{C}$ interacts with $\mathcal{A}$ to extract data blocks. Suppose that $\mathcal{C}$ challenges c blocks, namely the blocks with indices $\{j_1, j_2, \ldots, j_c\}$, then $\mathcal{A}$ responds with a proof $\mathbb{P}$ that contains $\sigma = \sigma_{j_1}^{r_{j_1}} \cdot \sigma_{j_2}^{r_{j_2}} \ldots \sigma_{j_c}^{r_{j_c}}$ and $\mu = \{\mu_{ik}\}_{\substack{1\leq i\leq n \\ 1\leq k\leq s}}$, where $\mu_{ik} = r_{j_1} \cdot \tilde{b}_{ij_1k} + r_{j_2} \cdot \tilde{b}_{ij_2k} + \cdots + r_{j_c} \cdot \tilde{b}_{ij_ck}$. The challenger $\mathcal{C}$ can extract the actual data blocks $\{\tilde{b}_{ijk}\}$ in polynomially-many interactions with $\mathcal{A}$. If the challenge-response phase has been repeated c times (each time we challenge c blocks), then there will be c proofs $\{\mathbb{P}^1, \mathbb{P}^2, \ldots, \mathbb{P}^c\}$. Thus, a system of linear equations can be constructed as follows.

$$\begin{aligned}
\mu_{11}^1 &= r_{j_1}^1 \cdot \tilde{b}_{1j_1 1} + r_{j_2}^1 \cdot \tilde{b}_{1j_2 1} + \cdots + r_{j_c}^1 \cdot \tilde{b}_{1j_c 1} \\
&\vdots \\
\mu_{11}^c &= r_{j_1}^c \cdot \tilde{b}_{1j_1 1} + r_{j_2}^c \cdot \tilde{b}_{1j_2 1} + \cdots + r_{j_c}^c \cdot \tilde{b}_{1j_c 1} \\
&\vdots \\
\mu_{ns}^c &= r_{j_1}^c \cdot \tilde{b}_{nj_1 s} + r_{j_2}^c \cdot \tilde{b}_{nj_2 s} + \cdots + r_{j_c}^c \cdot \tilde{b}_{nj_c s}
\end{aligned}$$

Solving this system of linear equations yields the data blocks $\{\tilde{b}_{ijk}\}$.

Finally, the PB-PMDP scheme is secure against file swapping attack. The file identifier ID_F is embedded into the block tag, and thus the CSP cannot use blocks from different files and pass the auditing procedures even if the owner uses the same secret key x with all his files.

3.7 Performance Analysis

In this section, we evaluate the performance of the presented schemes: MR-PDP and PB-PMDP. The file F used in our performance analysis is of size 64MB divided in blocks of 4KB. Without loss of generality, we assume that the desired security level is 80-bit. Thus, we utilize an elliptic curve defined over Galois field $GF(p)$ with $|p| = 160$ bits (a point on this curve can be represented by 161 bits using compressed representation [14]), and the size of the RSA modulus N is 1024 bits.

Similar to [80, 11, 54], the computation cost for the MR-PDP and PB-PMDP is estimated in terms of used cryptographic operations, which are notated in Table 3.1. $\mathbb{G}$ indicates a group of points over a suitable elliptic curve in the bilinear pairing, and QR_N is a set of quadratic residues modulo N.

Table 3.1: Notation of cryptographic operations.

Notation	Description	Notation	Description
$\mathcal{H}_{\mathbb{G}}$	Hashing to $\mathbb{G}$	$\mathcal{H}_{QR_N}$	Hashing to QR_N
$\mathcal{E}_{\mathbb{G}}$	Exponentiation in $\mathbb{G}$	$\mathcal{E}_{\mathbb{Z}_N}$	Exponentiation in $\mathbb{Z}_N$
$\mathcal{M}_{\mathbb{G}}$	Multiplication in $\mathbb{G}$	$\mathcal{M}_{\mathbb{Z}}$	Multiplication in $\mathbb{Z}$
$\mathcal{M}_{\mathbb{Z}_p}$	Multiplication in $\mathbb{Z}_p$	$\mathcal{D}_{\mathbb{Z}}$	Division in $\mathbb{Z}$
$\mathcal{A}_{\mathbb{Z}_p}$	Addition in $\mathbb{Z}_p$	$\mathcal{A}_{\mathbb{Z}}$	Addition in $\mathbb{Z}$
$\mathcal{P}$	Bilinear pairing	E_K	Encryption using K
$\mathcal{R}$	Random-number generation		

To perform a fair comparison between the PB-PMDP and the MR-PDP [37], we assume two small modifications to the model presented in [37]. First, we assume that the indices of the blocks being challenged are the same across all copies (*this assumption is an optimization for the verification computations of the MR-PDP*). Second, for the CSP to prove the possession of the blocks (not just only their sum), each block being challenged is multiplied by a random value. The second modification makes the S-PDP version of [8] to be the base of the MR-PDP scheme.

Let n, m, and s denote the number of copies, the number of blocks per copy, and the number of sectors per block, respectively. Let c denote the number of blocks to be challenged, and $|F|$ denote the size of the file copy. Let the keys used with π and ψ be of size 128 bits. Table 3.2 presents a theoretical analysis for the setup, storage, communication, and computation costs of the two schemes.

3.7.1 Comments

Sytem Setup. As it can be seen in Table 3.2, the cost of generating data copies in the proposed PB-PMDP scheme is much less than that of the MR-PDP scheme. On the other

Table 3.2: Storage, communication, and computation costs for MR-PDP and PB-PMDP schemes.

Costs		MR-PDP[37]	PB-MPDP
System Setup	Copies Generation	$E_K + nm\,\mathcal{R}$ $+ nm\,\mathcal{A}_{\mathbb{Z}}$	$n\,E_K$
	Tags Generation	$2m\,\mathcal{E}_{\mathbb{Z}_N} + m\,\mathcal{M}_{\mathbb{Z}}$ $+ m\,\mathcal{H}_{QR_N}$	$(s+1)nm\,\mathcal{E}_{\mathbb{G}} + nm\,\mathcal{H}_{\mathbb{G}}$ $+ (ns+n-1)m\,\mathcal{M}_{\mathbb{G}}$
Storage	File Copies	$n\lvert F\rvert$	$n\lvert F\rvert$
	CSP Overhead	$1024m$ bits	$161m$ bits
Communication	Challenge	$1280 + \log_2(c)$ bits	$256 + \log_2(c)$ bits
	Response	$1024(n+1)$ bits †	$161 + 160ns$ bits
Computation	Proof	$(c+n)\,\mathcal{E}_{\mathbb{Z}_N} + (cn+c-1)\,\mathcal{M}_{\mathbb{Z}}$ $+ (c-1)n\,\mathcal{A}_{\mathbb{Z}}$	$c\,\mathcal{E}_{\mathbb{G}} + (c-1)\,\mathcal{M}_{\mathbb{G}} + csn\,\mathcal{M}_{\mathbb{Z}_p}$ $+ (c-1)sn\,\mathcal{A}_{\mathbb{Z}_p}$
	Verification	$(2n+c+1)\,\mathcal{E}_{\mathbb{Z}_N} + c\,\mathcal{H}_{QR_N} + \mathcal{D}_{\mathbb{Z}}$ $+ (cn+c+n-1)\,\mathcal{M}_{\mathbb{Z}} + (c-1)n\,\mathcal{A}_{\mathbb{Z}}$	$2\mathcal{P} + (c+s+1)\,\mathcal{E}_{\mathbb{G}} + c\,\mathcal{H}_{\mathbb{G}}$ $+ (c+s-1)\,\mathcal{M}_{\mathbb{G}} + (n-1)s\,\mathcal{A}_{\mathbb{Z}_p}$

† There is an optimization for this response to be $1024 + 160n$ bits using hashing.

hand, Curtmola *et al.* [37] efficiently reduce the computation cost of generating the block tags. This is due to the fact that the tags are generated from the encrypted version of the file before masking with some unique randomness to generate the differentiable copies. In general, the impact of setup computations on the overall system performance may be insignificant; setup is done only once during the life time of the data storage system, which may be for tens of years.

Storage overhead. Storage overhead is the additional space used to store necessary information other than the outsourced file copies $\widetilde{\mathbb{F}}$. Both schemes require $n|F|$ bits to store $\widetilde{\mathbb{F}}$, while the storage overhead for the PB-PMDP scheme is much less than that of the MR-PDP model. The overheads on the CSP are 2MB and 0.31MB for the MR-PDP and PB-PMDP schemes, respectively (about 84% reduction). Reducing the storage overhead on the CSP side is economically a key feature to lower the fees paid by the customers.

Communication cost. The communication cost of the MR-PDP scheme is less than that of PB-PMDP. For 20 copies of F, the communication costs for the MR-PDP and PB-PMDP schemes are about 2.8KB and 80KB, respectively. However, for small s ($\ll n$), the PB-PMDP will have less communication cost.

Computation cost. As observed from Table 3.2, the cost expression of the proof for the MR-PDP scheme has three terms linear in the number of copies n, while the PB-PMDP scheme has two terms linear in n. Moreover, there are three terms linear in n in the verification cost expression for the MR-PDP scheme, while the PB-PMDP scheme contains only *one* term linear in n in the corresponding expression. These terms affect the total computation time when dealing with a large number of copies in practical applications. We note that since the cost of an addition is negligibly smaller than those of pairing and exponentiation, the verification time in the proposed PB-PMDP scheme is practically not affected by the value of n.

3.8 Implementation and Experimental Evaluation

3.8.1 Implementation

We have implemented the MR-PDP and PB-PMDP schemes on top of Amazon Elastic Compute Cloud (Amazon EC2) [5] and Amazon Simple Storage Service (Amazon S3) [6] cloud platforms. Amazon EC2 is a web service that enables customers to lunch and manage Linux/Unix and Windows server instances (virtual servers) in Amazon's data centers. Customers can automatically scale up and down the number of EC2 instances according to their demands. Moreover, customers can upgrade and downgrade a specific EC2 instance to fit current requirements. Amazon S3 is storage for the internet. It provides a simple web services interface that can be used to store and retrieve almost unlimited amount of data. Customers are allowed to choose the geographic locations where Amazon S3 will store the data.

Our implementation of the presented schemes consists of three modules: `OModule` (owner module), `CModule` (CSP module), and `VModule` (verifier module). `OModule`, which runs on the owner side, is a library that includes KeyGen, CopyGen, and TagGen algorithms. `CModule` is a library that runs on Amazon EC2 and includes Prove algorithm. `VModule` is a library to be run at the verifier side and includes the Verify algorithm.

In the experiments, we do not consider the system pre-processing time to prepare the different file copies and generate the tags set. Moreover, the time to access the file blocks is not considered in the implementation, as the state-of-the-art hard drive technology allows as much as 1MB to be read in just few nanoseconds [80]. Hence, the total access time is unlikely to have substantial impact on the overall system performance.

Implementation settings. In our implementation we use a "large" Amazon EC2 instance to run `CModule`. This instance type provides total memory of size 7.5GB and 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each). One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0 - 1.2GHz 2007 Opteron or 2007 Xeon processor [4]. The `OModule` and `VModule` are executed on a desktop computer with Intel(R) Xeon(R) 2GHz processor and 3GB RAM running Windows XP. We outsource copies of a data file of size 64MB to Amazon S3. Algorithms (encryption, pairing, hashing, etc.) are implemented using MIRACL library version 5.4.2. In the experiments, we utilize the MNT curve [68] defined over prime field $GF(p)$ with $|p| = 160$ bits and embedding degree $= 6$ (the MNT curve with these parameters is provided by the MIRACL library).

3.8.2 Experimental Evaluation

Timing measurements. The proposed PB-PMDP scheme is based on pairing and elliptic curve cryptography, while the MR-PDP scheme is based on RSA. To estimate the timing measurements for the cryptographic operations used in the implementations, we run the MIRACL library on the the used desktop computer. Table 3.3 presents the measured times (in milliseconds), where each reported measurement is an average of thousands of runs.

Table 3.3 shows three measurements for $\mathcal{E}_{\mathbb{Z}_N}$: 68.92 ms, 2.15 ms, and 0.32 ms. The reason is that the exponent part differs during the implementation of the MR-PDP scheme. For example, the data owner performs $\acute{g}^{\texttt{EXP}} \bmod N$, where `EXP` is the exponent part of size 4KB (32768 bits). The owner does this operation during the tag generations and the verification phase. Utilizing the Fermat-Euler theorem [59], the owner can reduce the

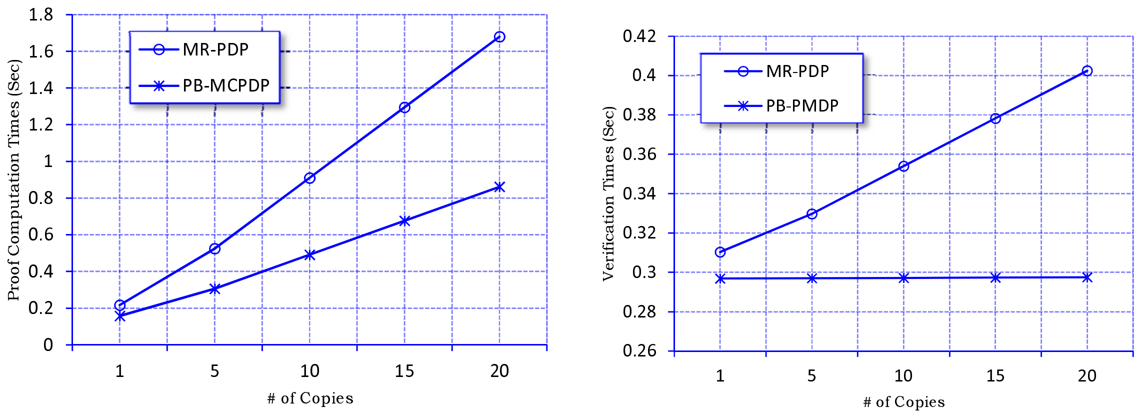
Table 3.3: Timing measurements for the cryptographic operations

Operation	Time (ms)	Operation	Time (ms)
$\mathcal{H}_{\mathbb{G}}$	0.22	$\mathcal{H}_{QR_N}$	0.34
$\mathcal{E}_{\mathbb{G}}$	0.27	$\mathcal{E}_{\mathbb{Z}_N}$	68.92/2.15/0.32
$\mathcal{M}_{\mathbb{G}}$	0.01	$\mathcal{M}_{\mathbb{Z}}$	0.02/0.004/0.0009
$\mathcal{M}_{\mathbb{Z}_p}$	0.00025	$\mathcal{D}_{\mathbb{Z}}$	0.00022
$\mathcal{A}_{\mathbb{Z}_p}$	0.00017	$\mathcal{A}_{\mathbb{Z}}$	0.009/0.00029
$\mathcal{P}$	4.6		

exponent part, where $\acute{g}^{\texttt{EXP}} \equiv \acute{g}^{\texttt{EXP} \bmod \phi(N)} \bmod N$ and $\phi(N) = (\acute{p} - 1)(\acute{q} - 1)$ is the Euler's totient function. On the other hand, the CSP cannot use this trick because $\phi(N)$ is not known in public. Therefore, $\mathcal{E}_{\mathbb{Z}_N}$ needs 68.92 ms and 2.15 ms at the CSP and the owner, respectively. Besides, a random value of size 160 bits is used in our slight modification to the MR-PDP scheme to prove the possession of the data blocks not only their sum. Thus, $\mathcal{E}_{\mathbb{Z}_N}$ needs 0.32 ms when the exponent part is 160 bits. Similar scenarios arise for $\mathcal{M}_{\mathbb{Z}}$ and $\mathcal{A}_{\mathbb{Z}}$ operations. $\mathcal{M}_{\mathbb{Z}}$ needs 0.02 ms for 32768-bits × 160-bits, 0.004 ms for 1024-bits × 1024-bits, and 0.0009 ms for 1024-bits × 160-bits. $\mathcal{A}_{\mathbb{Z}}$ needs 0.009 ms for 32768-bits + 32768-bits, and 0.00029 ms for 1024-bits + 1024-bits.

Experimental results. We compare the presented MR-PDP and PB-PMDP schemes in terms of both the proof computation times and the verification times. It has been reported in [8] that if the remote server is missing a fraction of the data, then the number of blocks that needs to be checked in order to detect server misbehavior with high probability is constant independent of the total number of file blocks. For example, if the server deletes 1% of the data file, the verifier needs only to check for $c = 460$-randomly chosen blocks of the file so as to detect this misbehavior with probability larger than 99%. Therefore, in our experiments, we use $c = 460$ to achieve a high probability of assurance.

For different number of copies, Figure 3.4a presents the proof computation times (in seconds). The timing curve of the proposed PB-PMDP scheme is less than that of the MR-PDP scheme. For 20 copies, the proof computation times for the MR-PDP and the PB-PMDP schemes are 1.68 and 0.86 seconds, respectively (about 49% reduction).



(a) CSP computation times (sec)

(b) Verifier computation times (sec)

Figure 3.4: Computation costs of the MR-PDP and PB-PMDP.

Figure 3.4b presents the verification times (in seconds). For 20 copies, the verification times for the MR-PDP and the PB-PMDP schemes are 0.40 and 0.29 seconds, respectively (about 27% reduction).

More importantly, the verification timing curve of the PB-PMDP scheme is *almost* unchanged for the range of number of copies considered in our experiments. This is due to the fact that although the term $(n-1)s\,\mathcal{A}_{\mathbb{Z}_p}$ in the verification cost of the PB-PMDP scheme is linear in n (Table 3.2), in our experiments its numerical value is quite small compared to those of the other terms in the cost expression. This feature makes the PB-PMDP scheme computationally cost-effective and more efficient when verifying a large number of file copies.

3.9 Identifying Corrupted Copies

Here we show how the proposed PB-PMDP scheme can be slightly modified to identify the indices of corrupted copies. The proof $\mathbb{P} = \{\sigma, \mu\}$ generated by the CSP will be valid and will pass the verification equation (3.1) only if all copies are intact and consistent. Thus, when there is one or more corrupted copies, the whole auditing procedure fails. To handle this situation and identify the corrupted copies, a slightly modified version of the PB-PMDP scheme can be used. In this version, the data owner generates a tag σ_{ij} for each block $\tilde{b}_{ij}$, but does not aggregate the tags for the blocks at the same indices in different copies, *i.e.*, $\Phi = \{\sigma_{ij}\}_{\substack{1 \le i \le n \\ 1 \le j \le m}}$. During the response phase, the CSP computes $\mu = \{\mu_{ik}\}_{\substack{1 \le i \le n \\ 1 \le k \le s}}$ as before, but $\sigma = \prod_{(j,r_j) \in Q} [\prod_{i=1}^{n} \sigma_{ij}]^{r_j} \in \mathbb{G}_1$. Upon receiving the proof $\mathbb{P} = \{\sigma, \mu\}$, the verifier first validates $\mathbb{P}$ using equation (3.1). If the verification fails, the verifier asks the CSP to send $\sigma = \{\sigma_i\}_{1 \le i \le n}$, where $\sigma_i = \prod_{(j,r_j) \in Q} \sigma_{ij}^{r_j}$. Thus, the verifier has two lists $\sigma\mathsf{List} = \{\sigma_i\}_{1 \le i \le n}$ and $\mu\mathsf{List} = \{\mu_{ik}\}_{\substack{1 \le i \le n \\ 1 \le k \le s}}$ ($\mu\mathsf{List}$ is a two dimensional list).

Utilizing a recursive divide-and-conquer approach (binary search) [46], the verifier can identify the indices of corrupted copies. Specifically, $\sigma\mathsf{List}$ and $\mu\mathsf{List}$ are divided into halves: $\sigma\mathsf{List} \rightarrow$ ($\sigma\mathsf{Left}$:$\sigma\mathsf{Right}$), and $\mu\mathsf{List} \rightarrow$ ($\mu\mathsf{Left}$:$\mu\mathsf{Right}$). The verification equation (3.1) is applied recursively on $\sigma\mathsf{Left}$ with $\mu\mathsf{Left}$ and $\sigma\mathsf{Right}$ with $\mu\mathsf{Right}$. Note that the individual tags in $\sigma\mathsf{Left}$ or $\sigma\mathsf{Right}$ are aggregated via multiplication to generate one σ that is used during the recursive application of equation (3.1). The procedural steps of identifying the indices of corrupted copies are indicated in Algorithm 1.

The binary search algorithm takes four parameters: $\sigma\mathsf{List}$, $\mu\mathsf{List}$, start that indicates the start index of the currently working lists, and end to indicate the last index of the working lists. The initial call to the search algorithm takes ($\sigma\mathsf{List}$, $\mu\mathsf{List}$, 1, n). The invalid indices are stored in $\mathsf{invalidList}$, which is a global data structure.

This slight modification to identify the corrupted copies will be associated with some extra storage overhead on the cloud servers, where the CSP has to store mn tags for the file copies $\widetilde{\mathbb{F}}$ (m tags in the original version). Moreover, the challenge-response phase may be done in two rounds if the initial round to verify all copies fails.

We have performed experiments to show the effect of identifying the corrupted copies on the verification time. We generate 100 copies (using same file/parameters from Section 3.8), which are verified in 0.3 seconds when all copies are accurate. A percentage – ranging from 1% to 20% – of the file copies is *randomly* corrupt. Figure 3.5 shows the verification time (in seconds) with different corrupted percentages. The verification time is about 3.87

Algorithm 1: BS(σList, μList, start, end)

```
begin
    $len \longleftarrow$ (end$-$start)$+1$          /* List length */
    if $len$ = $1$ then
        $\sigma \longleftarrow \sigma$List[start]
        $\{\mu_k\}_{1\le k\le s} \longleftarrow \mu$List[start][$k$]
        $\hat{e}(\sigma, g) \overset{?}{=} \hat{e}(\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||j)^{r_j} \cdot \prod_{k=1}^{s} u_k^{\mu_k}, y)$
        if NOT verified then
            invalidList.Add(start)
        end
    else
        $\sigma \longleftarrow \prod_{i=1}^{len} \sigma$List[start$+i-1$]
        $\{\mu_{ik}\}_{\substack{1\le i\le len\\ 1\le k\le s}} \longleftarrow \mu$List[start$+i-1$][$k$]
        $\hat{e}(\sigma, g) \overset{?}{=} \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||j)^{r_j}]^{len} \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{len} \mu_{ik}}, y)$
        if NOT verified then
            /* work with the left and right halves of $\sigma$List and $\mu$List */
            mid$\longleftarrow \lfloor$(start$+$end)$/2\rfloor$          /* List middle */
            BS($\sigma$List, $\mu$List, start, mid)       /* Left part */
            BS($\sigma$List, $\mu$List, mid$+1$, end)   /* Right part */
        end
    end
end
```

seconds when 1% of the copies are invalid. As observed from Figure 3.5, when the percentages of corrupted copies are up to 15% of the total copies, the performance of using the binary search algorithm in the verification is more efficient than individual verification for each copy. It takes about 0.29 seconds to verify one copy, and thus individual verifications of 100 copies requires $100\times0.29 = 29$ seconds.

In short, the proposed PB-PMDP scheme can be slightly modified to support the feature of identifying the corrupted copies at the cost of some extra storage, communication, and computation overheads. For the CSP to remain in business and maintain a good

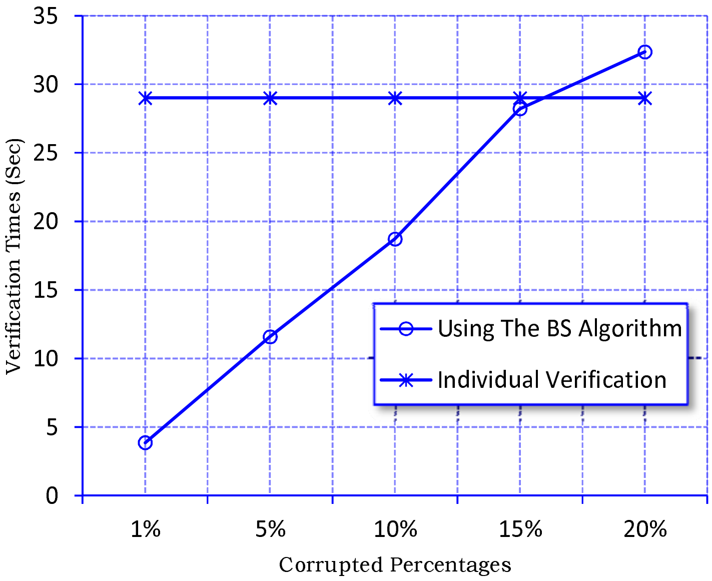


Figure 3.5: Verification times with different percentages of corrupted copies.

reputation, invalid responses to verifier's challenges are sent in very rare situations, and thus the original version of the proposed scheme is used in most of the time.

3.10 Summary

In this chapter, we have studied the problem of creating multiple copies of a data file and verifying those copies stored on cloud servers. We have proposed a pairing-based provable multi-copy data possession (PB-PMDP) scheme, which supports outsourcing of multiple data copies to untrusted CSP. The interaction between the authorized users and the CSP is considered in our scheme, where the authorized users can seamlessly access a data copy received from the CSP using a single secret key shared with the data owner. Moreover, the BP-PMDP scheme supports public verifiability, allows unlimited number of auditing, and provides *possession-free* verification where the verifier has the ability to verify the data integrity even though he neither possesses nor retrieves the file blocks from the server.

Our security analysis has shown that the proposed PB-PMDP scheme is provably secure against colluding servers. Through theoretical analysis, experimental results, and comparison with the MR-PDP scheme, we have explained the improved performance of the proposed scheme. The verification time of PB-PMDP is practically independent of the number of file copies, which makes the scheme computationally cost-effective and more efficient when verifying a large number of file copies.

A slight modification can be done on the proposed scheme to support the feature of identifying the indices of corrupted copies of data. The corrupted copy can be reconstructed even from a complete damage using duplicated copies on other servers.

Chapter 4

Provable Multi-Copy Dynamic Data Possession

In this chapter, we direct our study to the the dynamic behavior of multiple data copies outsourced to cloud servers, and describe the scheme we proposed in [16, 19]. Section 4.1 highlights the motivation of this work. Our system model and assumptions are presented in Section 4.2. The proposed scheme to verify the integrity of multiple dynamic data copies stored by cloud servers is elaborated in Section 4.4. Section 4.5 presents an extension to provable possession models for single-copy dynamic data to work in the setting of multiple copies of dynamic data. Section 4.6 contains the security analysis of the proposed scheme. The performance analysis is shown in Section 4.7. Section 4.8 presents the implementation and experimental results. Section 4.9 discusses how to identify the corrupted copy among the outsourced data copies. The chapter is summarized in Section 4.10.

4.1 Introduction

Increasingly more and more organizations are opting for outsourcing data to remote cloud service providers (CSPs). This is primarily to reduce the maintenance cost and the burden of large local data storage. Customers can rent the CSP's storage infrastructure to store and retrieve almost unlimited amount of data by paying fees metered in GB/month.

Replicating data on multiple servers across multiple data centers achieves a higher level of scalability, availability, and durability. The more copies the CSP is asked to store, the more fees the customers are charged. Moreover, the remotely stored data can be not only accessed by authorized users (*i.e.*, those who have the right to access the owner's file), but also updated and scaled by the data owner. Therefore, customers need to have a strong guarantee that the CSP is storing all data copies that are agreed upon in the service contract, and all these copies are consistent with the most recent modifications issued by the customers.

In this chapter, we propose a map-based provable multi-copy *dynamic* data possession (MB-PMDDP) scheme that achieves three main goals: (i) it provides an evidence to the customers that the CSP is not cheating by storing fewer copies, (ii) it supports outsourcing of dynamic data, *i.e.*, it supports *block-level* operations such as block modification, insertion, deletion, and append, and (iii) it allows authorized users to seamlessly access the file copies stored by the CSP. We show the security of the proposed scheme against colluding servers. We also give a comparative analysis of the proposed MB-PMDDP scheme with a reference model obtained by extending existing provable possession of dynamic *single-copy* schemes. The theoretical analysis is validated through experimental results. Additionally, we utilize similar ideas to that applied in the previous chapter to identify corrupted copies by slightly modifying the proposed MB-PMDDP scheme. To the best of our knowledge, there was no provable data possession (PDP) scheme for *multiple* copies of *dynamic* data in the open literature.

4.2 Our System and Assumptions

System components. The cloud computing storage model considered in this chapter is similar to that of Figure 3.2 (Chapter 3). It consists of three main components: a data owner, a CSP, and authorized users.

Outsourcing, updating, and accessing. The data owner has a file F divided into m blocks, and requests the CSP to store n copies $\{\widetilde{F}_1, \widetilde{F}_2, \dots, \widetilde{F}_n\}$ of F on different servers

– to prevent simultaneous failure of all copies. Replication strategy depends on the importance of outsourced data; more copies are needed for critical data that cannot easily be reproduced, and to achieve a higher level of scalability. For this critical data, multiple copies are distributed on multiple servers, while needless reproducible data are stored at reduced levels of redundancy. The data owner has to pay according to the used storage space on the CSP side, where the pricing model is related to the number of data copies.

For data confidentiality, the owner encrypts his data before outsourcing to the CSP. After outsourcing all n copies of the file, the owner may interact with the CSP to perform block-level operations on all copies. These operations includes modify, insert, append, and delete specific blocks of the outsourced data copies.

An authorized user of the outsourced data sends a data-access request to the CSP and receives a file copy in an encrypted form that can be decrypted using a secret key shared with the owner. According to the load balancing mechanism used by the CSP to organize the work of the servers, the data-access request is directed to the server with the lowest congestion, and thus the authorized user is not aware of which copy has been received.

We assume that the interaction between the owner and the authorized users to authenticate their identities and share the secret key has already been completed, and it is not considered in this work. Throughout this chapter, the terms cloud server and cloud service provider are used interchangeably.

Threat model. The integrity of customers' data in the cloud may be at risk due to the following reasons. First, the CSP – whose goal is likely to make a profit and maintain a reputation – has an incentive to hide data loss (due to hardware failure, management errors, various attacks) or reclaim storage by discarding data that has not been or is rarely accessed. Second, to save the computational resources, the CSP may totally ignore the data-update requests issued by the owner, or not execute them on all copies leading to inconsistency between the file copies. Third, a dishonest CSP may store fewer copies than what has been agreed upon in the service contact with the data owner, and try to convince the owner that all copies are correctly stored intact. The goal of the proposed scheme is to detect (with *high probability*) the CSP misbehavior by validating the number and integrity of file copies.

Underlying algorithms. The proposed scheme consists of seven polynomial time algorithms: KeyGen, $\mathsf{CopyGen}$, TagGen, $\mathsf{PrepareUpdate}$, $\mathsf{ExecUpdate}$, Prove, and Verify.

- $(pk, sk) \leftarrow \mathsf{KeyGen}()$. This algorithm is run by the data owner to generate a public key pk and a private key sk. The private key sk is kept secret by the owner, while pk is publicly known.
- $\widetilde{\mathbb{F}} \leftarrow \mathsf{CopyGen}(CN_i, F)_{1\leq i\leq n}$. This algorithm is run by the data owner. It takes as input a copy number CN_i and a file F, and generates n copies $\widetilde{\mathbb{F}} = \{\widetilde{F}_i\}_{1\leq i\leq n}$. The owner sends the copies $\widetilde{\mathbb{F}}$ to the CSP to be stored on cloud servers.
- $\Phi \leftarrow \mathsf{TagGen}(sk, \widetilde{\mathbb{F}})$. This algorithm is run by the data owner. It takes as input the private key sk and the file copies $\widetilde{\mathbb{F}}$, and outputs tags/authenticators set Φ, which is an ordered collection of tags for the data blocks. The owner sends Φ to the CSP to be stored along with the copies $\widetilde{\mathbb{F}}$.
- $(\mathcal{D}', \; \mathit{UpdateReq}) \leftarrow \mathsf{PrepareUpdate}(\mathcal{D}, \; \mathit{UpdateInfo})$. This algorithm is run by the data owner to update the outsourced file copies stored by the remote CSP. The input parameters are a previous metadata $\mathcal{D}$ stored on the owner side, and some information *UpdateInfo* about the dynamic operation to be performed on a specific block. The outputs of this algorithm are a modified metadata $\mathcal{D}'$ and an update request *UpdateReq*. This request may contain a modified version of a previously stored block, a new block to be inserted, or a delete command to delete a specific block from the file copies. *UpdateReq* also contains updated (or new) tags for modified (or inserted/appended) blocks, and it is sent from the data owner to the CSP in order to perform the requested update.
- $(\widetilde{\mathbb{F}}', \; \Phi') \leftarrow \mathsf{ExecUpdate}(\widetilde{\mathbb{F}}, \; \Phi, \; \mathit{UpdateReq})$. This algorithm is run by the CSP, where the input parameters are the file copies $\widetilde{\mathbb{F}}$, the tags set Φ, and the request *UpdateReq*. It outputs an updated version of the file copies $\widetilde{\mathbb{F}}'$ along with an updated tags set Φ'.

- $\mathbb{P} \leftarrow \mathsf{Prove}(\widetilde{\mathbb{F}}, \Phi, chal)$. This algorithm is run by the CSP. It takes as input the file copies $\widetilde{\mathbb{F}}$, the tags set Φ, and a challenge *chal* (sent from a verifier). It returns a proof $\mathbb{P}$ which guarantees that the CSP is actually storing n copies and all these copies are intact, updated, and consistent.

- $\{1, 0\} \leftarrow \mathsf{Verify}(pk, \mathbb{P}, \mathcal{D})$. This algorithm is run by a verifier (original owner or any other trusted auditor). It takes as input the public key pk, the proof $\mathbb{P}$ returned from the CSP, and the most recent metadata $\mathcal{D}$. The output is 1 if the integrity of all file copies is correctly verified or 0 otherwise.

4.3 Security Model

As we have indicated in Chapter 3, two security requirements can be defined [81]: correctness and soundness. The former means that the verifier accepts valid server responses, and the latter indicates that any cheating server that passes the verification process is actually storing the owner's data intact.

The security of the proposed scheme can be stated using a "game" that captures the data possession property [8, 42, 81]. The data possession game between an adversary $\mathcal{A}$ (acts as a malicious CSP) and a challenger $\mathcal{C}$ (acts as a verifier) consists of the following:

- Setup. $\mathcal{C}$ runs the KeyGen algorithm to generate a key pair (pk, sk), and sends pk to $\mathcal{A}$.

- Interact. $\mathcal{A}$ interacts with $\mathcal{C}$ to get the file copies and the verification tags set Φ. $\mathcal{A}$ adaptively selects a file F and sends it to $\mathcal{C}$. $\mathcal{C}$ divides the file into m blocks, runs the two algorithms $\mathsf{CopyGen}$ and TagGen to create n distinct copies $\widetilde{\mathbb{F}}$ along with the tags set Φ, and returns both $\widetilde{\mathbb{F}}$ and Φ to $\mathcal{A}$.

  Moreover, $\mathcal{A}$ can interact with $\mathcal{C}$ to perform dynamic operations on $\widetilde{\mathbb{F}}$. $\mathcal{A}$ specifies a block to be updated, inserted, or deleted, and sends the block to $\mathcal{C}$. $\mathcal{C}$ runs the $\mathsf{PrepareUpdate}$ algorithm, sends the *UpdateReq* to $\mathcal{A}$, and updates the local metadata

$\mathcal{D}$. $\mathcal{A}$ can further request challenges $\{chal_i\}_{1\leq i\leq L}$ for some parameter $L \geq 1$ of $\mathcal{A}$'s choice, and return proofs $\{\mathbb{P}_i\}_{1\leq i\leq L}$ to $\mathcal{C}$. $\mathcal{C}$ runs the Verify algorithm and provides the verification results to $\mathcal{A}$. The INTERACT step between $\mathcal{A}$ and $\mathcal{C}$ can be repeated polynomially-many times.

- CHALLENGE. $\mathcal{A}$ decides on a file F previously used during the INTERACT step, requests a challenge $chal$ from $\mathcal{C}$, and generates a proof $\mathbb{P} \leftarrow \mathsf{Prove}(\widetilde{\mathbb{F}}', \Phi, chal)$, where $\widetilde{\mathbb{F}}'$ is $\widetilde{\mathbb{F}}$ except that at least one of its file copies (or a portion of it) is missing or tampered with. Upon receiving the proof $\mathbb{P}$, $\mathcal{C}$ runs the Verify algorithm and if $\mathsf{Verify}(pk, \mathbb{P}, \mathcal{D})$ returns 1, then $\mathcal{A}$ has won the game. Note that $\mathcal{D}$ is the latest metadata held by $\mathcal{C}$ corresponding to the file F. The CHALLENGE step can be repeated polynomially-many times for the purpose of data extraction.

The proposed scheme is secure if the probability that any polynomial-time adversary $\mathcal{A}$ wins the game is negligible. In other words, if a polynomial-time adversary $\mathcal{A}$ can win the game with non-negligible probability, then there exists a polynomial time extractor that can repeatedly execute the CHALLENGE step until it extracts the blocks of data copies.

In addition, a remote checking scheme for dynamic data must be secure against the following types of attacks.

- **File swapping attack**. In this type of attacks, the remote server tries to prove the possession of the data using blocks from different files.
- **Replay attack**. The remote server does not perform the data modification requests issued by the owner, and sends stale date as a response to a challenge vector.

4.4 Proposed MB-PMDDP Scheme

4.4.1 Overview and Rationale

As mentioned in Chapter 3, generating unique differentiable copies of the data file is the core to design a multi-copy provable data possession scheme. Identical data copies enable the CSP to simply deceive the owner by storing only one copy and pretending that it stores multiple copies. Utilizing the *diffusion* feature of any secure encryption model, the proposed scheme can generate distinct copies. The diffusion property ensures that there will be an unpredictable complete change in the ciphertext, if there is a single bit change in the plaintext [35]. Generating the distinct copies based on the diffusion feature enables authorized users to seamlessly decrypt and access a file copy received from the CSP. The received copy is decrypted using a single secret key (shared with the data owner), and it is not necessarily to recognize which copy has been received.

In this chapter, we propose a MB-PMDDP scheme allowing the data owner to update and scale the blocks of the file copies outsourced to cloud servers which may be untrusted. Validating such dynamic data copies requires the knowledge of the block versions to ensure that the data blocks in all copies are consistent with the most recent modifications issued by the owner. Moreover, the verifier should be aware of the block indices to guarantee that the CSP has inserted or added the new blocks at the requested positions in all copies. To this end, the proposed MB-PMDDP scheme is based on using a small data structure (metadata), which we call a map-version table.

4.4.2 Map-Version Table

The map-version table (MVT) is a small *dynamic* data structure stored on the verifier side to validate the integrity and consistency of all file copies outsourced to the CSP. The MVT consists of three columns: serial number ($\mathcal{SN}$), block number ($\mathcal{BN}$), and block version ($\mathcal{BV}$). The $\mathcal{SN}$ is an indexing to the file blocks. It indicates the *physical* position of a block in a data file. The $\mathcal{BN}$ is a counter used to make a *logical* numbering/indexing to the file blocks. Thus, the relation between $\mathcal{BN}$ and $\mathcal{SN}$ can be viewed as a mapping between the logical number $\mathcal{BN}$ and the physical position $\mathcal{SN}$. The $\mathcal{BV}$ indicates the current version of file blocks. When a data file is initially created the $\mathcal{BV}$ of each block is 1. If a specific block is being updated, its $\mathcal{BV}$ is incremented by 1.

Remark 1. It is important to note that the verifier keeps only <u>*one*</u> table for unlimited number of file copies, *i.e.*, the storage requirement on the verifier side does not depend on the number of file copies on cloud servers. For n copies of a data file of size $|F|$, the storage requirement on the CSP side is $O(n|F|)$, while the verifier's overhead is $O(m)$ for all file copies (m is the number of file blocks).

Remark 2. The MVT is implemented as a linked list to simplify the insertion and deletion of table entries. For actual implementation, the $\mathcal{SN}$ is not needed to be stored in the table; $\mathcal{SN}$ is considered to be the entry/table index, *i.e.*, each table entry contains just two integers $\mathcal{BN}$ and $\mathcal{BV}$ (8 bytes). Thus, the total table size is $8m$ bytes for all file copies. We further note that although the table size is linear to the file size, in practice the former would be smaller by *several orders of magnitude*. For example, outsourcing unlimited number of file copies of a 1GB-file with 16KB block size requires a verifier to keep MVT of only 512KB (less than 0.05% of the file size). More details on the MVT and how it works will be explained later.

4.4.3 Notations

- F is a data file to be outsourced, and is composed of a sequence of m blocks, *i.e.*, $F = \{b_1, b_2, \ldots, b_m\}$.
- $\pi_{key}(\cdot)$ is a pseudo-random permutation (PRP): $key \times \{0,1\}^{\log_2(m)} \rightarrow \{0,1\}^{\log_2(m)}$. [1]
- $\psi_{key}(\cdot)$ is a pseudo-random function (PRF): $key \times \{0,1\}^* \rightarrow \mathbb{Z}_p$.
- **Bilinear Map/Pairing**. Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups of prime order p. Let $\bar{g}$ and g be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. A bilinear pairing is a map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ with the following properties indicated in sub-section 3.5.2 of Chapter 3: *Bilinear*, *Non-degenerate*, and *Computable*.
- $\mathcal{H}(\cdot)$ is a map-to-point hash function : $\{0,1\}^* \rightarrow \mathbb{G}_1$.
- h is a cryptographic hash function, *e.g.*, SHA-2.

[1] The number of file blocks (m) will be changed due to dynamic operations on the file. We use HMAC-SHA-1 with 160-bit output to allow up to 2^{160} blocks in the file.

– E_K is an encryption algorithm with strong *diffusion* property, *e.g.*, AES (Advanced Encryption Standard) [38].

4.4.4 MB-PMDDP Procedural Steps

- ***Key Generation***. Let $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear map and g is a generator of $\mathbb{G}_2$. The data owner runs the KeyGen algorithm to generate a private key $x \in \mathbb{Z}_p$ and a public key $y = g^x \in \mathbb{G}_2$ along with $(u_1, u_2, \ldots, u_s) \in_R \mathbb{G}_1$.

- ***Generation of Distinct Copies***. For a file $F = \{b_j\}_{1 \leq j \leq m}$, the owner runs the CopyGen algorithm to create n differentiable copies $\widetilde{\mathbb{F}} = \{\widetilde{F}_i\}_{1 \leq i \leq n}$, where a copy $\widetilde{F}_i = \{\tilde{b}_{ij}\}_{1 \leq j \leq m}$. The block $\tilde{b}_{ij}$ is generated by concatenating a copy number i with the block b_j, then encrypting using an encryption scheme E_K, *i.e.*, $\tilde{b}_{ij} = E_K(i||b_j)$. The encrypted block $\tilde{b}_{ij}$ is fragmented into s sectors $\{\tilde{b}_{ij1}, \tilde{b}_{ij2}, \ldots, \tilde{b}_{ijs}\}$, *i.e.*, the copy $\widetilde{F}_i = \{\tilde{b}_{ijk}\}_{\substack{1 \leq j \leq m \\ 1 \leq k \leq s}}$, where each sector $\tilde{b}_{ijk} \in \mathbb{Z}_p$ for some large prime p.

  The authorized users need only to keep a single secret key K. Later, when an authorized user receives a file copy from the CSP, he decrypts the copy blocks, removes the copy index from the blocks header, and then recombines the decrypted blocks to reconstruct the plain form of the received file copy.

- ***Generation of Tags***. Given the distinct file copies $\widetilde{\mathbb{F}} = \{\widetilde{F}_i\}$, where $\widetilde{F}_i = \{\tilde{b}_{ijk}\}$, the data owner runs the TagGen algorithm to generate a tag σ_{ij} for each block $\tilde{b}_{ij}$ as $\sigma_{ij} = (\mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j) \cdot \prod_{k=1}^{s} u_k^{\tilde{b}_{ijk}})^x \in \mathbb{G}_1$ $(i : 1 \to n,\ j : 1 \to m,\ k : 1 \to s)$. In the tag computation, $\mathcal{BN}_j$ is the *logical* number of the block at *physical* position j, $\mathcal{BV}_j$ is the current version of that block, and $ID_F = Filename||n||u_1|| \ldots ||u_s$ is a unique fingerprint for each file F comprising the file name, the number of copies for this file, and the random values $\{u_k\}_{1 \leq k \leq s}$. We assume that ID_F is signed with some owner's signing secret key (different than x), and the CSP verifies this signature during different scheme operations to validate the owner's identity. Embedding the ID_F into the block tag σ_{ij} prevents the CSP from cheating by using blocks from different files (*file swapping attack*).

In order to reduce storage overhead on cloud servers and lower communication cost, the data owner generates an aggregated tag σ_j for the blocks at the same indices in each copy $\widetilde{F}_i$ as $\sigma_j = \prod_{i=1}^{n} \sigma_{ij} \in \mathbb{G}_1$. Hence, instead of storing mn tags, the proposed MB-PMDDP scheme requires the CSP to store only m tags for the files copies $\widetilde{\mathbb{F}}$. Let us denote the set of aggregated tags as $\Phi = \{\sigma_j\}_{1 \leq j \leq m}$. The data owner sends $\{\widetilde{\mathbb{F}}, \Phi, ID_F\}$ to the CSP, and deletes the copies and the tags from its local storage. The MVT is stored on the local storage of the owner (or any trusted verifier).

■ ***Dynamic Operations on the Data Copies***. The dynamic operations in the proposed MB-PMDDP scheme are performed at the block level via a request in the general form $\langle ID_F, BlockOp, j, \{b_i^*\}_{1 \leq i \leq n}, \sigma_j^* \rangle$, where ID_F is the file identifier and *BlockOp* corresponds to block modification (denoted by BM), block insertion (denoted by BI), or block deletion (denoted by BD). The parameter j indicates the index of the block to be updated, $\{b_i^*\}_{1 \leq i \leq n}$ are the new block values for all copies, and σ_j^* is the new aggregated tag for the new blocks.

♦ **Modification**. For a file $F = \{b_1, b_2, \ldots, b_m\}$, suppose the owner wants to modify a block b_j with a block b'_j for all file copies $\widetilde{\mathbb{F}}$. The owner runs the PrepareUpdate algorithm to do the following:

1. Updates $\mathcal{BV}_j = \mathcal{BV}_j + 1$ in the MVT
2. Creates n distinct blocks $\{\tilde{b}'_{ij}\}_{1 \leq i \leq n}$, where $\tilde{b}'_{ij} = E_K(i || b'_j)$ is fragmented into s sectors $\{\tilde{b}'_{ij1}, \tilde{b}'_{ij2}, \ldots, \tilde{b}'_{ijs}\}$
3. Creates a new tag σ'_{ij} for each block $\tilde{b}'_{ij}$ as $\sigma'_{ij} = (\mathcal{H}(ID_F || \mathcal{BN}_j || \mathcal{BV}_j) \cdot \prod_{k=1}^{s} u_k^{\tilde{b}'_{ijk}})^x \in \mathbb{G}_1$, then generates an aggregated tag $\sigma'_j = \prod_{i=1}^{n} \sigma'_{ij} \in \mathbb{G}_1$
4. Sends a modify request $\langle ID_F, \mathsf{BM}, j, \{\tilde{b}'_{ij}\}_{1 \leq i \leq n}, \sigma'_j \rangle$ to the CSP

Upon receiving the modify request, the CSP runs the ExecUpdate algorithm to do the following:

1. Replaces the block $\tilde{b}_{ij}$ with $\tilde{b}'_{ij} \forall i$, and constructs updated file copies $\widetilde{\mathbb{F}}' = \{\widetilde{F}'_i\}_{1 \leq i \leq n}$

2. Replaces σ_j with σ'_j in the set Φ, and outputs $\Phi' = \{\sigma_1, \sigma_2, \dots, \sigma'_j, \dots, \sigma_m\}$

♦ **Insertion**. In the block insertion operation, the owner wants to insert a new block $\hat{b}$ *after* position j in a file $F = \{b_1, b_2, \dots, b_m\}$, *i.e.*, the newly constructed file is $F' = \{b_1, b_2, \dots, b_j, \hat{b}, \dots, b_{m+1}\}$, where $b_{j+1} = \hat{b}$. In the proposed MB-PMDDP scheme, the *physical* block index $\mathcal{SN}$ is not included in the block tag. Thus, the insertion operation can be performed without recomputing the tags of all blocks that have been shifted after inserting the new block. Embedding the physical index in the tag results in unacceptable computation overhead, especially for large data files. To perform the insertion of a new block $\hat{b}$ after position j in all file copies $\widetilde{\mathbb{F}}$, the owner runs the PrepareUpdate algorithm to do the following:

1. Constructs a new table entry $\langle \mathcal{SN}, \mathcal{BN}, \mathcal{BV} \rangle = \langle j{+}1, (Max\{\mathcal{BN}_j\}_{1 \le j \le m}){+}1, 1\rangle$, and inserts this entry in the MVT after position j
2. Creates n distinct blocks $\{\hat{b}_i\}_{1 \le i \le n}$, where $\hat{b}_i = E_K(i||\hat{b})$ is fragmented into s sectors $\{\hat{b}_{i1}, \hat{b}_{i2}, \dots, \hat{b}_{is}\}$
3. Creates a new tag $\hat{\sigma}_i$ for each block $\hat{b}_i$ as $\hat{\sigma}_i = (\mathcal{H}(ID_F||\mathcal{BN}_{j+1}||\mathcal{BV}_{j+1}) \cdot \prod_{k=1}^{s} u_k^{\hat{b}_{ik}})^x \in \mathbb{G}_1$, then generates an aggregated tag $\hat{\sigma} = \prod_{i=1}^{n} \hat{\sigma}_i \in \mathbb{G}_1$. Note that $\mathcal{BN}_{j+1}$ is the logical number of the new block with current version $\mathcal{BV}_{j+1} = 1$
4. Sends an insert request $\langle ID_F, \mathsf{BI}, j, \{\hat{b}_i\}_{1 \le i \le n}, \hat{\sigma}\rangle$ to the CSP

Upon receiving the insert request, the CSP runs the ExecUpdate algorithm to do the following:

1. Inserts the block $\hat{b}_i$ after position j in the file copy $\widetilde{F}_i$ $\forall i$, and constructs a new version of the file copies $\widetilde{\mathbb{F}}' = \{\widetilde{F}'_i\}_{1 \le i \le n}$
2. Inserts $\hat{\sigma}$ after position j in the set Φ, and outputs $\Phi' = \{\sigma_1, \dots, \sigma_j, \hat{\sigma}, \dots, \sigma_{m+1}\}$, *i.e.*, $\sigma_{j+1} = \hat{\sigma}$

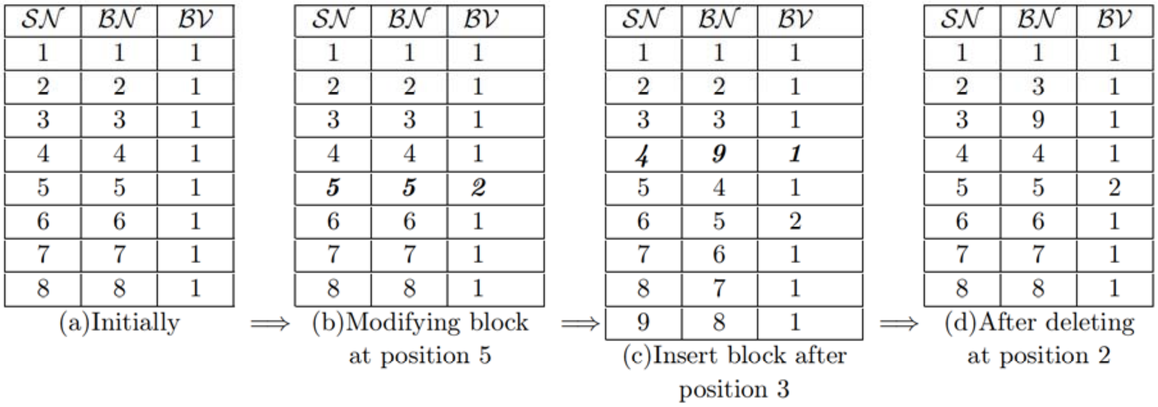
Remark 3. To prevent the CSP from cheating and using less storage, the modified or inserted blocks for the outsourced copies cannot be identical. To this end, the proposed scheme in this work leaves the control of creating such distinct blocks in the owner hand. This illustrates the linear relation between the work done by the owner during dynamic operations and the number of copies. The proposed scheme assumes that the CSP stores the outsourced copies on *different* servers to avoid simultaneous failure and achieve a higher level of availability. Therefore, even if the CSP is honest to perform part of the owner work, this is unlikely to significantly reduce the communication overhead since the distinct blocks are sent to different servers for updating the copies. The experimental results show that the computation overhead on the owner side due to dynamic block operations is practical.

- **Append**. Block append operation means adding a new block at the end of the outsourced data. It can simply be implemented via insert operation after the last block of the data file.

- **Deletion**. Block deletion operation is the opposite of the insertion operation. When one block is deleted all subsequent blocks are moved one step forward. To delete a specific data block at position j from all copies, the owner deletes the entry at position j from the MVT and sends a delete request $\langle ID_F, \mathsf{BD}, j, null, null\rangle$ to the CSP. Upon receiving this request, the CSP runs the ExecUpdate algorithm to do the following:

  1. Deletes the blocks $\{\tilde{b}_{ij}\}_{1\leq i\leq n}$, and outputs a new version of the file copies $\widetilde{\mathbb{F}}' = \{\widetilde{F}_i'\}_{1\leq i\leq n}$
  2. Deletes σ_j from Φ and outputs $\Phi' = \{\sigma_1, \sigma_2, \ldots, \sigma_{j-1}, \sigma_{j+1}, \ldots, \sigma_{m-1}\}$

Figure 4.1 shows the changes in the MVT due to dynamic operations on the copies $\widetilde{\mathbb{F}}$ of a file $F = \{b_j\}_{1\leq j\leq 8}$. When the copies are initially created (Figure 4.1a), $\mathcal{SN}_j = \mathcal{BN}_j$ and $\mathcal{BV}_j = 1$: $1 \leq j \leq 8$. Figure 4.1b shows that $\mathcal{BV}_5$ is incremented by 1 for updating the block at position 5 for all copies. To insert a new block after position 3 in $\widetilde{\mathbb{F}}$, Figure 4.1c shows that a new entry $\langle 4, 9, 1\rangle$ is inserted in the MVT after $\mathcal{SN}_3$, where 4 is the physical position of the newly inserted block, 9 is the new logical block number computed by incrementing the maximum of all previous logical block numbers, and 1 is the version of the new block. Deleting a block at position 2 from all copies requires deleting the table entry at $\mathcal{SN}_2$ and shifting all subsequent entries one position up (Figure 4.1d). Note that during all dynamic operations, the $\mathcal{SN}$ indicates the actual physical positions of the data blocks in the file copies $\widetilde{\mathbb{F}}$.



$\mathcal{SN}$	$\mathcal{BN}$	$\mathcal{BV}$
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1
6	6	1
7	7	1
8	8	1

(a)Initially ⟹

$\mathcal{SN}$	$\mathcal{BN}$	$\mathcal{BV}$
1	1	1
2	2	1
3	3	1
4	4	1
5	**5**	**2**
6	6	1
7	7	1
8	8	1

(b)Modifying block at position 5 ⟹

$\mathcal{SN}$	$\mathcal{BN}$	$\mathcal{BV}$
1	1	1
2	2	1
3	3	1
4	**9**	**1**
5	4	1
6	5	2
7	6	1
8	7	1
9	8	1

(c)Insert block after position 3 ⟹

$\mathcal{SN}$	$\mathcal{BN}$	$\mathcal{BV}$
1	1	1
2	3	1
3	9	1
4	4	1
5	5	2
6	6	1
7	7	1
8	8	1

(d)After deleting at position 2

Figure 4.1: Changes in the MVT due to different dynamic operations on copies of a file $F = \{b_j\}_{1\leq j\leq 8}$.

- ■ ***Challenge***. For challenging the CSP and validating the integrity and consistency of all copies, the verifier sends c (# of blocks to be challenged) and two fresh keys at each challenge: a PRP(π) key k_1 and a PRF(ψ) key k_2. *Both* the verifier and the CSP use π keyed with k_1 and the ψ keyed with k_2 to generate a set $Q = \{(j, r_j)\}$ of c pairs of random indices and random values, where $\{j\} = \pi_{k_1}(l)_{1\leq l\leq c}$ and $\{r_j\} = \psi_{k_2}(l)_{1\leq l\leq c}$. The set of random indices $\{j\}$ is the physical positions (serial numbers $\mathcal{SN}$) of the blocks to be challenged.

- ■ ***Response***. The CSP runs the Prove algorithm to generate a set $Q = \{(j, r_j)\}$ of random indices and values, and provide an evidence that the CSP is still correctly possessing the n copies in an updated and consistent state. The CSP responds with a proof $\mathbb{P} = \{\sigma, \mu\}$, where

$$\sigma = \prod_{(j,r_j)\in Q} \sigma_j^{r_j} \in \mathbb{G}_1, \quad \mu_{ik} = \sum_{(j,r_j)\in Q} r_j \cdot \tilde{b}_{ijk} \in \mathbb{Z}_p, \quad \text{and } \mu = \{\mu_{ik}\}_{\substack{1\leq i\leq n\\ 1\leq k\leq s}}.$$

- ***Verify Response***. Upon receiving the proof $\mathbb{P} = \{\sigma, \mu\}$ from the CSP, the verifier runs the Verify algorithm to check the following verification equation:

$$\hat{e}(\sigma, g) \stackrel{?}{=} \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n}\mu_{ik}}, y) \tag{4.1}$$

The verifier utilizes the set of random indices $\{j\}$ (generated from π) and the MVT to get the logical block number $\mathcal{BN}_j$ and the block version $\mathcal{BV}_j$ of each block being challenged. If the verification equation passes, the Verify algorithm returns 1, otherwise 0. The correctness of the verification equation (4.1) can be shown as follows:

$$\begin{aligned}
\hat{e}(\sigma, g) &= \hat{e}(\prod_{(j,r_j)\in Q} \sigma_j^{r_j}, g) = \hat{e}(\prod_{(j,r_j)\in Q} [\prod_{i=1}^{n} \sigma_{ij}]^{r_j}, g) \\
&= \hat{e}(\prod_{(j,r_j)\in Q} [\prod_{i=1}^{n} (\mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j) \cdot \prod_{k=1}^{s} u_k^{\tilde{b}_{ijk}})^x]^{r_j}, g) \\
&= \hat{e}(\prod_{(j,r_j)\in Q} [\prod_{i=1}^{n} (\mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j) \cdot \prod_{k=1}^{s} u_k^{\tilde{b}_{ijk}})]^{r_j}, y) \\
&= \hat{e}(\prod_{(j,r_j)\in Q} \prod_{i=1}^{n} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j)^{r_j} \cdot \prod_{(j,r_j)\in Q} \prod_{i=1}^{n} \prod_{k=1}^{s} u_k^{r_j \cdot \tilde{b}_{ijk}}, y) \\
&= \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n} \sum_{(j,r_j)\in Q} r_j \cdot \tilde{b}_{ijk}}, y) \\
&= \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n} \mu_{ik}}, y)
\end{aligned}$$

One can attempt to slightly modify the MB-PMDDP scheme to reduce the communication overhead by a factor of n via allowing the CSP to compute and send $\mu = \{\hat{\mu}_k\}_{1\leq k\leq s}$, where $\hat{\mu}_k = \sum_{i=1}^{n} \mu_{ik}$. However, this modification enables the CSP to simply cheat the verifier as follows:

$$\hat{\mu}_k = \sum_{i=1}^{n} \mu_{ik} = \sum_{i=1}^{n} \sum_{(j,r_j)\in Q} r_j \cdot \tilde{b}_{ijk} = \sum_{(j,r_j)\in Q} r_j \cdot \sum_{i=1}^{n} \tilde{b}_{ijk}$$

Thus, the CSP can just keep the sectors summation $\sum_{i=1}^{n} \tilde{b}_{ijk}$ not the sectors themselves. Moreover, the CSP can corrupt the block sectors and the summation is still valid. Therefore, we require the CSP send $\mu = \{\mu_{ik}\}_{\substack{1\le i\le n\\ 1\le k\le s}}$, and the summation $\sum_{i=1}^{n} \mu_{ik}$ is done on the verifier side. The challenge response protocol in the MB-PMDDP scheme is summarized in Figure 4.2.

Verifier | **CSP**

1. Generates a PRP key k_1 and a PRF key k_2.
2. Determines c (# of blocks in the challenge vector)
3. Generates a set $Q = \{(j, r_j)\}$,
   $\{j\} = \pi_{k_1}(l)_{1\le l\le c}$,
   and $\{r_j\} = \psi_{k_2}(l)_{1\le l\le c}$

$$\xrightarrow{c,\, k_1,\, k_2}$$

4. Uses k_1 and k_2 to generate the set Q
5. Computes $\sigma = \prod_{(j,r_j)\in Q} \sigma_j^{r_j} \in \mathbb{G}_1$
6. Computes $\mu = \{\mu_{ik}\}_{\substack{1\le i\le n\\ 1\le k\le s}}$, $\mu_{ik} = \sum_{(j,r_j)\in Q} r_j \cdot \tilde{b}_{ijk} \in \mathbb{Z}_p$

$$\xleftarrow{\sigma,\, \mu=\{\mu_{ik}\}_{\substack{1\le i\le n\\ 1\le k\le s}}}$$

7. Checks $\hat{e}(\sigma, g) \stackrel{?}{=} \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n} \mu_{ik}}, y)$

Figure 4.2: Challenge response protocol in the MB-PMDDP scheme.

Remark 4. The proposed MB-PMDDP scheme supports public verifiability where anyone, who knows the owner's public key but is not necessarily the data owner, can send a challenge vector to the CSP and verify the response. Public verifiability can resolve disputes that may occur between the data owner and the CSP regarding data integrity. If such a dispute occurs, a trusted third party auditor (TPA) can determine whether the data integrity is maintained or not. Since the owner's public key is only needed to perform the verification step, the owner is not required to reveal his secret key to the TPA.

4.5 Extending Dynamic Single-Copy PDP schemes

It is possible to obtain a provable multi-copy dynamic data possession scheme by extending existing PDP models for single-copy dynamic data. Such PDP schemes selected for extension must meet the following conditions: (i) support of *full* dynamic operations (modify, insert, append, and delete), (ii) support of public verifiability, (iii) based on pairing cryptography in creating block tags (homomorphic authenticators); and (iv) block tags are outsourced along with data blocks to the CSP (*i.e.*, tags are not stored on the local storage of the data owner). Meeting these conditions allows us to construct a PDP reference model that has similar features to the proposed MB-PMDDP scheme. Therefore, we can establish a fair comparison between the two schemes and evaluate the performance of our proposed approach.

Below we drive a scheme by extending PDP models, which are based on authenticated data structures, *e.g.*, [42] and [87]. Using Merkle hash trees (MHTs) [67], we construct a scheme labelled as TB-PMDDP (tree-based provable multi-copy dynamic data possession), but it can also be designed using authenticated skip lists [42] or other authenticated data structures. The TB-PMDDP is used as a reference model for comparing the proposed MB-PMDDP scheme.

4.5.1 Merkle Hash Tree

An MHT [67] is a binary tree structure used to efficiently verify the integrity of the data. The MHT is a tree of hashes where the leaves of the tree are the hashes of the data blocks. Figure 4.3 shows an example of an MHT used for verifying the integrity of a file F consisting of 8 blocks (h denotes a cryptographic hash function, *e.g.*, SHA-2).

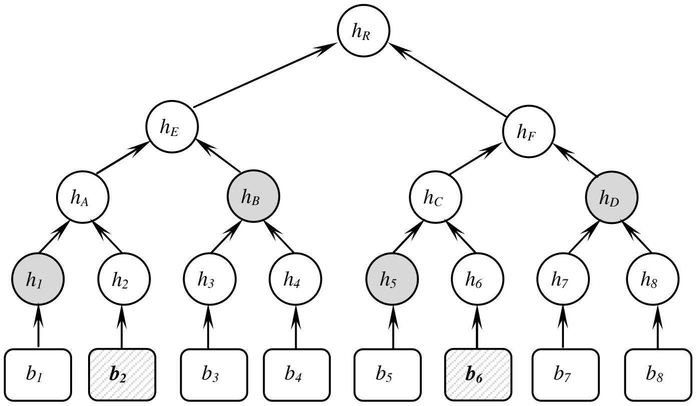


Figure 4.3: Merkle hash tree.

The hash $h_j = h(b_j)$ $(1 \leq j \leq 8)$. At upper levels, $h_A = h(h_1||h_2)$, $h_B = h(h_3||h_4)$, and so on. Finally, $h_R = h(h_E||h_F)$ is the hash of the root node that is used to authenticate the integrity of all data blocks. The data blocks $\{b_1, b_2, \ldots, b_8\}$ are stored on a remote server, and only the authentic value h_R is stored locally on the verifier side. For example, if the verifier requests to check the integrity of the blocks b_2 and b_6, the server will send these two blocks along with the authentication paths $\mathbb{A}_2 = \{h_1, h_B\}$ and $\mathbb{A}_6 = \{h_5, h_D\}$ that are used to reconstruct the root of the MHT. $\mathbb{A}_j$ – the authentication path of b_j – is a set of node siblings (grey-shaded circles) on the path from h_j to the root of the MHT. The verifier uses the received blocks and the authentication paths to recompute the root in the following manner. The verifier constructs $h_2 = h(b_2)$, $h_6 = h(b_6)$, $h_A = h(h_1||h_2)$, $h_C = h(h_5||h_6)$, $h_E = h(h_A||h_B)$, $h_F = h(h_C||h_D)$, and $h_R = h(h_E||h_F)$. After computing h_R, it is compared with the authentic value stored locally on the verifier side.

The MHT is commonly used to authenticate the *values* of the data blocks. In the dynamic behavior of outsourced data, we need to authenticate both the *values* and the *positions* of the data blocks, *i.e.*, we need an assurance that a specific value is stored at a specific leaf node. For example, if a data owner requires to insert a new block after position j, the verifier needs to make sure that the server has inserted the new block at the requested position. To validate the positions of the blocks, the leaf nodes of the MHT are treated in a specific sequence, *e.g.*, left-to-right sequence [62]. So, the hash of any internal node = h(left child || right child), *e.g.*, $h_A = h(h_1||h_2) \neq h(h_2||h_1)$. Besides, the authentication path $\mathbb{A}_j$ is viewed as an *ordered* set, and thus any leaf node is uniquely specified by following the used sequence of constructing the root of the MHT.

4.5.2 Directory MHT for File Copies

In the TB-PMDDP scheme an MHT is constructed for each file copy, and then the roots of the individual trees are used to build a hash tree which we call a directory MHT. The key idea is to make the root node of each copy's MHT as a leaf node in a directory MHT used to authenticate the integrity of all file copies in a hierarchical manner. The directory tree is depicted in Figure 4.4. The verifier can keep only one hash value (metadata) $\mathcal{M} = h(ID_F||h_{DR})$, where ID_F is a unique file identifier for a file F, and h_{DR} is the authenticated directory root value that can be used to periodically check the integrity of all file copies.

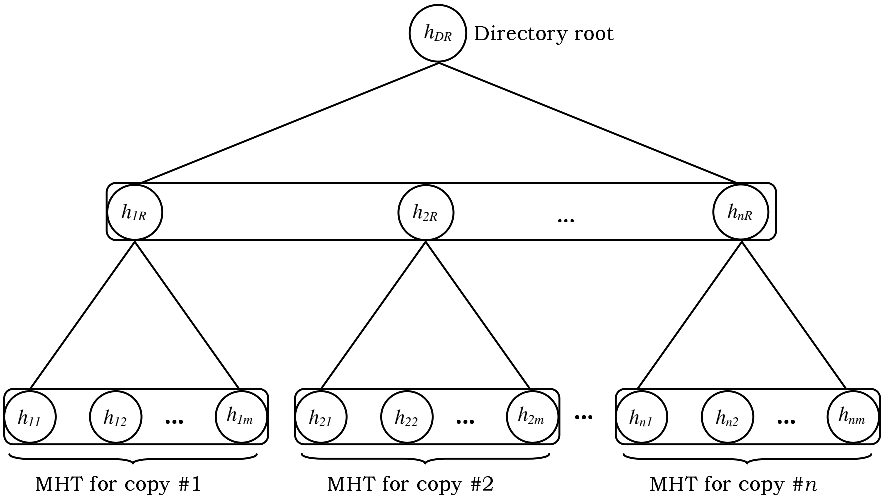


Figure 4.4: Directory tree.

4.5.3 TB-PMDDP Procedural Steps

- ***Key Generation***. The same as in the MB-PMDDP scheme.

- ***Generation of Distinct Copies***. The same as in the MB-PMDDP scheme.

- ***Generation of Tags and Trees***. Given the distinct file copies $\widetilde{\mathbb{F}} = \{\widetilde{F}_i\}$, where $\widetilde{F}_i = \{\tilde{b}_{ijk}\}$, the data owner runs the TagGen algorithm to create a tag σ_{ij} for each block $\tilde{b}_{ij}$ as $\sigma_{ij} = (\mathcal{H}(\tilde{b}_{ij}) \cdot \prod_{k=1}^{s} u_k^{\tilde{b}_{ijk}})^x \in \mathbb{G}_1$ $(i : 1 \to n,\ j : 1 \to m,\ k : 1 \to s)$. Similar to the MB-PMDDP scheme, the data owner reduces the storage overhead on the CSP side by generating an aggregated tag σ_j for the blocks at the same indices in each copy $\widetilde{F}_i$ as $\sigma_j = \prod_{i=1}^{n} \sigma_{ij} \in \mathbb{G}_1$. Let us denote the set of aggregated tags as $\Phi = \{\sigma_j\}_{1 \le j \le m}$.

  The data owner then generates an MHT for each file copy $\widetilde{F}_i$. The leaf nodes of each tree are the ordered set $\{h(\mathcal{H}(\tilde{b}_{ij}))\}$, *i.e.*, the leaf nodes of the MHT are the cryptographic hashes of $\mathcal{H}(\tilde{b}_{ij})$, and the root of the tree is denoted as h_{iR}. Using the roots $\{h_{iR}\}_{1 \le i \le n}$, the data owner generates a directory MHT in which the leaf nodes are $\{h_{iR}\}_{1 \le i \le n}$, and the directory root is denoted as h_{DR}. Note that the MHTs are constructed using a specific sequence, *e.g.*, left-to-right sequence to authenticate both the value and the position of $\mathcal{H}(\tilde{b}_{ij})$. The owner computes a metadata $\mathcal{M} = h(ID_F || h_{DR})$, where $ID_F = Filename||n||u_1|| \ldots ||u_s$ is a unique identifier for each owner's file F. The data owner sends $\langle \widetilde{\mathbb{F}}, \Phi, ID_F, \{\text{MHT}_i\}_{1 \le i \le n} \rangle$ to the CSP and deletes the copies, the tags, and the trees from its local storage. The metadata $\mathcal{M}$ is stored on the local storage of the owner (or any trusted verifier).

- ***Dynamic Operations on the Data Copies***. The dynamic operations in the TB-PMDDP scheme are performed at the block level – as in the MB-PMDDP scheme – via a request in the general form $\langle ID_F, BlockOp, j, \{b_i^*\}_{1 \le i \le n}, \sigma_j^* \rangle$. ID_F is the file identifier, *BlockOp* is BM for modification, BI for insertion, or BD for deletion, j indicates the index of the block to be updated, $\{b_i^*\}_{1 \le i \le n}$ are the new block values for all copies, and σ_j^* is the new aggregated tag for the new blocks.

- **Modification**. For a file $F = \{b_1, b_2, \dots, b_m\}$, suppose the owner wants to modify a block b_j with a block b'_j for all file copies $\widetilde{\mathbb{F}}$. The owner runs the $\mathsf{PrepareUpdate}$ algorithm to do the following:

  1. Creates n distinct blocks $\{\tilde{b}'_{ij}\}_{1\leq i\leq n}$, where $\tilde{b}'_{ij} = E_K(i||b'_j)$ is fragmented into s sectors $\{\tilde{b}'_{ij1}, \tilde{b}'_{ij2}, \dots, \tilde{b}'_{ijs}\}$
  2. Creates a new tag σ'_{ij} for each block $\tilde{b}'_{ij}$ as $\sigma'_{ij} = (\mathcal{H}(\tilde{b}'_{ij}) \cdot \prod_{k=1}^{s} u_k^{\tilde{b}'_{ijk}})^x \in \mathbb{G}_1$, then generates an aggregated tag $\sigma'_j = \prod_{i=1}^{n} \sigma'_{ij} \in \mathbb{G}_1$
  3. Sends a modify request $\langle ID_F, \mathsf{BM}, j, \{\tilde{b}'_{ij}\}_{1\leq i\leq n}, \sigma'_j\rangle$ to the CSP

  Upon receiving the modify request, the CSP runs the $\mathsf{ExecUpdate}$ algorithm to do the following:

  1. Replaces the block $\tilde{b}_{ij}$ with $\tilde{b}'_{ij}\forall i$, and constructs updated file copies $\widetilde{\mathbb{F}}' = \{\widetilde{F}'_i\}_{1\leq i\leq n}$
  2. Replaces $h(\mathcal{H}(\tilde{b}_{ij}))$ with $h(\mathcal{H}(\tilde{b}'_{ij}))$ in the leaf nodes of each copy's MHT, and accordingly updates the MHTs
  3. Calculates the authentication paths $\langle \mathbb{A}_{ij}\rangle_{1\leq i\leq n}$ of the updated blocks at position j in all copies. $\mathbb{A}_{ij}$ is an *ordered* set of node siblings on the path from the leaf node $h(\mathcal{H}(\tilde{b}'_{ij}))$ to the root of the MHT of copy i
  4. Replaces σ_j with σ'_j in the aggregated tags set Φ, and outputs $\Phi' = \{\sigma_1, \sigma_2, \dots, \sigma'_j, \dots, \sigma_m\}$
  5. Sends $\langle \mathbb{A}_{ij}\rangle_{1\leq i\leq n}$ to the owner

  Upon receiving $\langle \mathbb{A}_{ij}\rangle_{1\leq i\leq n}$ from the CSP, the owner uses these authentication paths and $\{\tilde{b}'_{ij}\}_{1\leq i\leq n}$ to generate a new directory root h'_{DR} and update the metadata $\mathcal{M}' = h(ID_F||h'_{DR})$.

- **Insertion**. Inserting a new block $\hat{b}$ *after* position j in a file $F = \{b_1, b_2, \dots, b_m\}$ constructs a new file $F' = \{b_1, b_2, \dots, b_j, \hat{b}, \dots, b_{m+1}\}$, where $b_{j+1} = \hat{b}$. The *physical* block index is not included in the block tag, and thus a new block can be inserted without recomputing the tags of all blocks that have been shifted after the insertion operation. MHTs are used to validate the positions of file blocks.

To perform the insertion of a new block $\hat{b}$ after position j in all file copies, the owner runs the $\mathsf{PrepareUpdate}$ algorithm to do the following:

1. Creates n distinct blocks $\{\hat{b}_i\}_{1 \leq i \leq n}$, where $\hat{b}_i = E_K(i||\hat{b})$ is fragmented into s sectors $\{\hat{b}_{i1}, \hat{b}_{i2}, \ldots, \hat{b}_{is}\}$
2. Creates a new tag $\hat{\sigma}_i$ for each block $\hat{b}_i$ as $\hat{\sigma}_i = (\mathcal{H}(\hat{b}_i) \cdot \prod_{k=1}^{s} u_k^{\hat{b}_{ik}})^x \in \mathbb{G}_1$, then generates an aggregated tag $\hat{\sigma} = \prod_{i=1}^{n} \hat{\sigma}_i \in \mathbb{G}_1$
3. Sends an insert request $\langle ID_F, \mathsf{BI}, j, \{\hat{b}_i\}_{1 \leq i \leq n}, \hat{\sigma} \rangle$ to the CSP

Upon receiving the insert request, the CSP runs the $\mathsf{ExecUpdate}$ algorithm to do the following:

1. Inserts the block $\hat{b}_i$ after position j in the file copy $\widetilde{F}_i$ $\forall i$, and then adds a leaf node $h(\mathcal{H}(\hat{b}_i))$ after the leaf node $h(\mathcal{H}(\tilde{b}_{ij}))$ for each copy's MHT. This leads to constructing a new version of the file copies $\widetilde{\mathbb{F}}' = \{\widetilde{F}'_i\}_{1 \leq i \leq n}$, and a new version of the MHTs
2. Calculates the authentication paths $\langle \widehat{\mathbb{A}}_i \rangle_{1 \leq i \leq n}$ of the newly inserted blocks $\{\hat{b}_i\}_{1 \leq i \leq n}$ in all copies. $\widehat{\mathbb{A}}_i$ is an ordered set of node siblings on the path from the leaf node $h(\mathcal{H}(\hat{b}_i))$ to the root of the MHT of copy i
3. Inserts $\hat{\sigma}$ after position j in the set Φ, and outputs $\Phi' = \{\sigma_1, \ldots, \sigma_j, \hat{\sigma}, \ldots, \sigma_{m+1}\}$, *i.e.*, $\sigma_{j+1} = \hat{\sigma}$
4. Sends $\langle \widehat{\mathbb{A}}_i \rangle_{1 \leq i \leq n}$ to the owner

Upon receiving $\langle \widehat{\mathbb{A}}_i \rangle_{1 \leq i \leq n}$ from the CSP, the owner uses these authentication paths and $\{\hat{b}_i\}_{1 \leq i \leq n}$ to generate a new directory root h'_{DR} and update the metadata $\mathcal{M}' = h(ID_F||h'_{DR})$.

♦ **Append**. It can simply be implemented via insert operation after the last block of the data file.

♦ **Deletion**. To delete a specific data block at position j from all copies, the owner sends a delete request $\langle ID_F, \mathsf{BD}, j, null, null \rangle$ to the CSP. Upon receiving this request, the CSP runs the $\mathsf{ExecUpdate}$ algorithm to do the following:

1. Calculates the authentication paths $\langle \mathbb{A}_{ij} \rangle_{1 \le i \le n}$ of the blocks at position j (blocks to be deleted) in all copies. $\mathbb{A}_{ij}$ is an *ordered* set of node siblings on the path from the leaf node $h(\mathcal{H}(\tilde{b}_{ij}))$ to the root of the MHT of copy i
2. Deletes the existing blocks $\{\tilde{b}_{ij}\}_{1 \le i \le n}$ and the leaf nodes $\{h(\mathcal{H}(\tilde{b}_{ij}))\}_{1 \le i \le n}$, and outputs new file copies $\widetilde{\mathbb{F}}'$ along with a new version of the MHTs
3. Deletes σ_j from Φ, and outputs $\Phi' = \{\sigma_1, \sigma_2 \ldots, \sigma_{j-1}, \sigma_{j+1} \ldots, \sigma_{m-1}\}$
4. Sends $\langle \mathbb{A}_{ij} \rangle_{1 \le i \le n}$ to the owner

The owner uses the authentication information received from the CSP to generate a new directory root h'_{DR} and update the metadata $\mathcal{M}' = h(ID_F || h'_{DR})$.

Remark 5. Appendix A contains examples that demonstrate how the dynamic operations performed on the outsourced file copies affect the MHTs on the CSP side. Moreover, these examples show how the owner uses the information received from the CSP to generate the new directory root and update the metadata $\mathcal{M}$.

- ***Challenge***. The same as in the MB-PMDDP scheme.

- ***Response***. The CSP runs the Prove algorithm to generate a set $Q = \{(j, r_j)\}$ of random indices and values, and provide an evidence that the CSP is still correctly possessing the n copies in an updated and consistent state. The CSP responds with a proof $\mathbb{P} = \langle \sigma, \mu, \{\mathcal{H}(\tilde{b}_{ij})\}_{\substack{1 \le i \le n \\ (j,*) \in Q}}, \langle \mathbb{A}_{ij} \rangle_{\substack{1 \le i \le n \\ (j,*) \in Q}} \rangle$, where

$$\sigma = \prod_{(j,r_j) \in Q} \sigma_j^{r_j} \in \mathbb{G}_1 \ , \qquad \mu_{ik} = \sum_{(j,r_j) \in Q} r_j \cdot \tilde{b}_{ijk} \in \mathbb{Z}_p, \qquad \text{and } \mu = \{\mu_{ik}\}_{\substack{1 \le i \le n \\ 1 \le k \le s}}.$$

$\langle \mathbb{A}_{ij} \rangle_{\substack{1 \le i \le n \\ (j,*) \in Q}}$ are the authentication paths of $\{\mathcal{H}(\tilde{b}_{ij})\}_{\substack{1 \le i \le n \\ (j,*) \in Q}}$.

- ***Verify Response***. Upon receiving the proof $\mathbb{P} = \langle \sigma, \mu, \{\mathcal{H}(\tilde{b}_{ij})\}_{\substack{1 \le i \le n \\ (j,*) \in Q}}, \langle \mathbb{A}_{ij} \rangle_{\substack{1 \le i \le n \\ (j,*) \in Q}} \rangle$ from the CSP, the verifier runs the Verify algorithm to do the following:

  1. Constructs the directory root h_{DR} using $\{\mathcal{H}(\tilde{b}_{ij})\}_{\substack{1 \le i \le n \\ (j,*) \in Q}}$ and $\langle \mathbb{A}_{ij} \rangle_{\substack{1 \le i \le n \\ (j,*) \in Q}}$

2. Generates a value $\mathcal{V} = h(ID_F||h_{DR})$, and checks $\mathcal{V} \stackrel{?}{=} \mathcal{M}$, where $\mathcal{M}$ is the authenticated-most-recent metadata stored on the verifier side. If the checking fails, returns 0, otherwise the Verify algorithm checks the following verification equation:
$$\hat{e}(\sigma, g) \stackrel{?}{=} \hat{e}(\prod_{i=1}^{n} \prod_{(j,r_j)\in Q} \mathcal{H}(\tilde{b}_{ij})^{r_j} \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n} \mu_{ik}}, y) \tag{4.2}$$

If the verification equation passes, the verifier accepts the response, otherwise rejects. The correctness of equation (4.2) can be illustrated as follows:

$$\begin{aligned}
\hat{e}(\sigma, g) &= \hat{e}(\prod_{(j,r_j)\in Q} \sigma_j^{r_j}, g) = \hat{e}(\prod_{(j,r_j)\in Q} [\prod_{i=1}^{n} \sigma_{ij}]^{r_j}, g) \\
&= \hat{e}(\prod_{(j,r_j)\in Q} [\prod_{i=1}^{n} (\mathcal{H}(\tilde{b}_{ij}) \cdot \prod_{k=1}^{s} u_k^{\tilde{b}_{ijk}})^{x}]^{r_j}, g) \\
&= \hat{e}(\prod_{(j,r_j)\in Q} [\prod_{i=1}^{n} (\mathcal{H}(\tilde{b}_{ij}) \cdot \prod_{k=1}^{s} u_k^{\tilde{b}_{ijk}})]^{r_j}, y) \\
&= \hat{e}(\prod_{(j,r_j)\in Q} \prod_{i=1}^{n} \mathcal{H}(\tilde{b}_{ij})^{r_j} \cdot \prod_{(j,r_j)\in Q} \prod_{i=1}^{n} \prod_{k=1}^{s} u_k^{r_j \cdot \tilde{b}_{ijk}}, y) \\
&= \hat{e}(\prod_{i=1}^{n} \prod_{(j,r_j)\in Q} \mathcal{H}(\tilde{b}_{ij})^{r_j} \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n} \sum_{(j,r_j)\in Q} r_j \cdot \tilde{b}_{ijk}}, y) \\
&= \hat{e}(\prod_{i=1}^{n} \prod_{(j,r_j)\in Q} \mathcal{H}(\tilde{b}_{ij})^{r_j} \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n} \mu_{ik}}, y)
\end{aligned}$$

For the verification purpose of the TB-PMDDP scheme, it is not sufficient to check only $\mathcal{V} \stackrel{?}{=} \mathcal{M}$. The directory root h_{DR} is reconstructed using hash values sent from the CSP. Thus, if the scheme counts only on verifying $\mathcal{V} \stackrel{?}{=} \mathcal{M}$, the CSP can simply cheat by storing the hashes of the outsourced data blocks not the blocks themselves. The scheme needs to verify equation (4.2) that guarantees the storage of the actual data. The challenge response protocol is summarized in Figure 4.5.

Verifier **CSP**

1. Picks c (# of blocks to be challenged), and two fresh keys k_1 and k_2
2. Generates a set $Q = \{(j, r_j)\}$: $\{j\} = \pi_{k_1}(l)_{1 \le l \le c}$ and $\{r_j\} = \psi_{k_2}(l)_{1 \le l \le c}$

$$\xrightarrow{c,\, k_1,\, k_2}$$

3. Generates a set Q as the verifier did
4. Computes $\sigma = \prod_{(j,r_j) \in Q} \sigma_j^{r_j} \in \mathbb{G}_1$
5. Computes $\mu = \{\mu_{ik}\}_{\substack{1 \le i \le n,\\ 1 \le k \le s}}$ $\mu_{ik} = \sum_{(j,r_j) \in Q} r_j \cdot \tilde{b}_{ijk} \in \mathbb{Z}_p$
6. Calculates $\{\mathcal{H}(\tilde{b}_{ij})\}_{\substack{1 \le i \le n\\ (j,*) \in Q}}$ and $\langle \mathbb{A}_{ij} \rangle_{\substack{1 \le i \le n\\ (j,*) \in Q}}$

$$\xleftarrow{\langle \sigma, \mu, \{\mathcal{H}(\tilde{b}_{ij})\}_{\substack{1 \le i \le n\\ (j,*) \in Q}}, \langle \mathbb{A}_{ij} \rangle_{\substack{1 \le i \le n\\ (j,*) \in Q}} \rangle}$$

7. Constructs h_{DR} using $\{\mathcal{H}(\tilde{b}_{ij})\}_{\substack{1 \le i \le n\\ (j,*) \in Q}}$ and $\langle \mathbb{A}_{ij} \rangle_{\substack{1 \le i \le n\\ (j,*) \in Q}}$
8. Computes $\mathcal{V} = h(ID_F || h_{DR})$, and checks $\mathcal{V} \overset{?}{=} \mathcal{M}$ (if fails returns 0).
9. Checks $\hat{e}(\sigma, g) \overset{?}{=} \hat{e}(\prod_{i=1}^{n} \prod_{(j,r_j) \in Q} \mathcal{H}(\tilde{b}_{ij})^{r_j} \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n} \mu_{ik}}, y)$

Figure 4.5: The challenge response protocol in the TB-PMDDP scheme.

4.6 Security Analysis

In this section, we present the security analysis for the MB-PMDDP scheme. The security proof for the TB-PMDDP is quite similar and is not presented here. For the TB-PMDDP scheme, we should however note that the verification step of the metadata $\mathcal{M}$ will fail unless the CSP sends the correct information $\{\mathcal{H}(\tilde{b}_{ij})\}_{\substack{1 \le i \le n\\ (j,*) \in Q}}$ along with the accurate authentication paths $\langle \mathbb{A}_{ij} \rangle_{\substack{1 \le i \le n\\ (j,*) \in Q}}$ for the blocks being challenged. This is due to the collision resistance property of the cryptographic hash function h and the used sequence to reconstruct the directory root h_{DR}.

The MB-PMDDP scheme utilizes PRP (π) and PRF (ψ) to compress the challenge, and thus reducing the communication cost. Instead of sending the set Q of c pairs of random indices and values to the CSP, the verifier sends only two keys k_1 and k_2 (over secure communication). Using π and ψ in this manner is proved to be secure [10].

For the correctness security requirement, we have previously shown the correctness of equation (4.1). For the soundness security requirement, we will show that if a polynomial-time adversary $\mathcal{A}$ can win the data possession game (with non-negligible probability) with a challenger $\mathcal{C}$, then $\mathcal{A}$ is actually storing the n data copies $\widetilde{\mathbb{F}}$ in an updated and consistent state. For an adversary $\mathcal{A}$ to cheat the verifier, he has to respond with a malicious proof $\mathbb{P}' \neq \mathbb{P}$ and $\mathsf{Verify}(pk, \mathbb{P}')$ returns 1.

The soundness of the MB-PMDDP scheme is based on the *unforgeability* of the used HLAs (homomorphic linear authenticators), which depends on the security of the computational Diffie-Hellman (CDH) and the discrete logarithm (DL) problems (refer to Section 3.6 for the definitions of CDH and DL problems).

The following theorem proves the unforgeability of the HLAs used in the proposed MB-PMDDP scheme. Our approach to prove the theorem is by investigating all possible combinations of malicious CSP responses $\langle\{\sigma', \mu'\}, \{\sigma, \mu'\}, \{\sigma', \mu\}\rangle$, and checking whether any of these combinations can pass the verification equation (3.1).

Theorem 1. *Assuming the hardness of both the CDH and the DL problems in bilinear groups, the verifier of the proposed MB-PMDDP scheme accepts a response to a challenge vector only if a correctly computed proof $\mathbb{P} = \{\sigma,\ \mu\}$, where $\mu = \{\mu_{ik}\}_{\substack{1\le i\le n\\ 1\le k\le s}}$ is sent from the CSP.*

Proof. We prove the theorem by contradiction. The goal of an adversary $\mathcal{A}$ (malicious CSP) is to generate a response that is not correctly computed and pass the verification process done by a challenger $\mathcal{C}$. Let $\mathbb{P}' = \{\sigma',\ \mu'\}$ be the $\mathcal{A}$'s response, where $\mu' = \{\mu'_{ik}\}_{\substack{1\le i\le n\\ 1\le k\le s}}$. Let $\mathbb{P} = \{\sigma,\ \mu\}$ be the expected response from an honest CSP, where $\sigma = \prod_{(j,r_j)\in Q} \sigma_j^{r_j}$, $\mu = \{\mu_{ik}\}_{\substack{1\le i\le n\\ 1\le k\le s}}$, and $\mu_{ik} = \sum_{(j,r_j)\in Q} r_j \cdot \tilde{b}_{ijk}$.

According to the correctness of MB-PMDDP scheme, the expected proof $\mathbb{P} = \{\sigma,\ \mu\}$

satisfies the verification equation, *i.e.*,

$$\hat{e}(\sigma, g) = \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n}\mu_{ik}}, y)$$

Assume that $\sigma' \neq \sigma$, and σ' passes the verification equation, then we have

$$\hat{e}(\sigma', g) = \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{n}\mu'_{ik}}, y)$$

Obviously, if $\mu'_{ik} = \mu_{ik}\ \forall(i,k)$, it follows from the above verification equations that $\sigma' = \sigma$ which contradicts our assumption. Let us define $\Delta\mu_{ik} = \mu'_{ik} - \mu_{ik}$ $(1 \leq i \leq n, 1 \leq k \leq s)$. It must be the case that at least one of $\{\Delta\mu_{ik}\}_{\substack{1\leq i\leq n\\1\leq k\leq s}}$ is nonzero. Dividing the verification equation for the malicious response by the verification equation for the expected response, we obtain

$$\begin{aligned}
\hat{e}(\sigma' \cdot \sigma^{-1}, g) &= \hat{e}(\prod_{k=1}^{s} u_k^{\sum_{i=1}^{n}\Delta\mu_{ik}}, y)\\
\hat{e}(\sigma' \cdot \sigma^{-1}, g) &= \hat{e}(\prod_{k=1}^{s} u_k^{x\cdot\sum_{i=1}^{n}\Delta\mu_{ik}}, g)\\
\sigma' \cdot \sigma^{-1} &= \prod_{k=1}^{s} u_k^{x\cdot\sum_{i=1}^{n}\Delta\mu_{ik}}.
\end{aligned}$$

We set $u_k = g^{\alpha_k}h^{\beta_k}$ for $\alpha_k, \beta_k \in \mathbb{Z}_p$, and thus

$$\begin{aligned}
\sigma' \cdot \sigma^{-1} &= \prod_{k=1}^{s}(g^{\alpha_k}h^{\beta_k})^{x\cdot\sum_{i=1}^{n}\Delta\mu_{ik}}\\
\sigma' \cdot \sigma^{-1} &= \prod_{k=1}^{s}(y^{\alpha_k}h^{x.\beta_k})^{\sum_{i=1}^{n}\Delta\mu_{ik}}\\
\sigma' \cdot \sigma^{-1} &= y^{\sum_{k=1}^{s}\alpha_k\cdot\sum_{i=1}^{n}\Delta\mu_{ik}} \,.\, h^{x\cdot\sum_{k=1}^{s}\beta_k\cdot\sum_{i=1}^{n}\Delta\mu_{ik}}\\
h^x &= (\sigma' \cdot \sigma^{-1} \cdot y^{-\sum_{k=1}^{s}\alpha_k\cdot\sum_{i=1}^{n}\Delta\mu_{ik}})^{\frac{1}{\sum_{k=1}^{s}\beta_k\cdot\sum_{i=1}^{n}\Delta\mu_{ik}}}.
\end{aligned}$$

Hence, we have found a solution to the CDH problem unless evaluating the exponent causes a division by zero. However, we noted that not all of $\{\Delta\mu_{ik}\}_{\substack{1\leq i\leq n\\1\leq k\leq s}}$ can be zero and the probability that $\beta_k = 0$ is $\frac{1}{p}$, which is negligible. Therefore, if $\sigma' \neq \sigma$, we can use the adversary $\mathcal{A}$ to break the CDH problem, and thus we guarantee that σ' must be equal to σ.

It is only the values $\mu' = \{\mu'_{ik}\}_{\substack{1\leq i\leq n\\1\leq k\leq s}}$ and $\mu = \{\mu_{ik}\}_{\substack{1\leq i\leq n\\1\leq k\leq s}}$ that can differ. Assume that the adversary $\mathcal{A}$ responds with $\sigma' = \sigma$ and $\mu' \neq \mu$. Now we have

$$\hat{e}(\sigma, g) = \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^n \mu_{ik}}, y) = \hat{e}(\sigma', g) =$$
$$\hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^n \mu'_{ik}}, y)$$

from which we conclude that

$$\hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^n \mu_{ik}}, y) =$$
$$\hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j)^{r_j}]^n \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^n \mu'_{ik}}, y).$$

Thus,

$$\begin{aligned}
1 &= \hat{e}(\prod_{k=1}^{s} u_k^{\sum_{i=1}^n \Delta\mu_{ik}}, y)\\
1 &= \prod_{k=1}^{s} (g^{\alpha_k} h^{\beta_k})^{\sum_{i=1}^n \Delta\mu_{ik}}\\
1 &= g^{\sum_{k=1}^s \alpha_k \cdot \sum_{i=1}^n \Delta\mu_{ik}} \cdot h^{\sum_{k=1}^s \beta_k \cdot \sum_{i=1}^n \Delta\mu_{ik}}\\
h &= g^{-\frac{\sum_{k=1}^s \alpha_k \cdot \sum_{i=1}^n \Delta\mu_{ik}}{\sum_{k=1}^s \beta_k \cdot \sum_{i=1}^n \Delta\mu_{ik}}}.
\end{aligned}$$

Now, we have found a solution to the DL problem unless evaluating the exponent causes a division by zero. However, the probability that $\beta_k = 0$ is $\frac{1}{p}$, which is negligible. Therefore, if there is at least one difference between $\{\mu'_{ik}\}_{\substack{1\leq i\leq n\\1\leq k\leq s}}$ and $\{\mu_{ik}\}_{\substack{1\leq i\leq n\\1\leq k\leq s}}$, we can use the adversary $\mathcal{A}$ to break the DL problem. As a result, we guarantee that $\{\mu'_{ik}\}$ must be equal to $\{\mu_{ik}\}\ \forall(i,k)$. $\square$

Data Extraction. We have shown that if a polynomial-time adversary $\mathcal{A}$ can win the data possession game (with non-negligible probability) with a challenger $\mathcal{C}$, then $\mathcal{A}$ is actually storing the data in an uncorrupted state. For the purpose of data extraction, the challenger $\mathcal{C}$ interacts with $\mathcal{A}$ to extract data blocks. Suppose that $\mathcal{C}$ challenges c blocks, namely the blocks with indices $\{j_1, j_2, \ldots, j_c\}$, then $\mathcal{A}$ responds with a proof $\mathbb{P}$ that contains $\sigma = \sigma_{j_1}^{r_{j_1}} \cdot \sigma_{j_2}^{r_{j_2}} \ldots \sigma_{j_c}^{r_{j_c}}$ and $\mu = \{\mu_{ik}\}_{\substack{1\leq i\leq n\\ 1\leq k\leq s}}$, where $\mu_{ik} = r_{j_1} \cdot \tilde{b}_{ij_1k} + r_{j_2} \cdot \tilde{b}_{ij_2k} + \cdots + r_{j_c} \cdot \tilde{b}_{ij_ck}$. The challenger $\mathcal{C}$ can extract the actual data blocks $\{\tilde{b}_{ijk}\}$ in polynomially-many interactions with $\mathcal{A}$. If the challenge-response phase has been repeated c times (each time we challenge c blocks), then there will be c proofs $\{\mathbb{P}^1, \mathbb{P}^2, \ldots, \mathbb{P}^c\}$. Thus, a system of linear equations can be constructed as follows.

$$\begin{aligned}
\mu_{11}^1 &= r_{j_1}^1 \cdot \tilde{b}_{1j_11} + r_{j_2}^1 \cdot \tilde{b}_{1j_21} + \cdots + r_{j_c}^1 \cdot \tilde{b}_{1j_c1} \\
&\vdots \\
\mu_{11}^c &= r_{j_1}^c \cdot \tilde{b}_{1j_11} + r_{j_2}^c \cdot \tilde{b}_{1j_21} + \cdots + r_{j_c}^c \cdot \tilde{b}_{1j_c1} \\
&\vdots \\
\mu_{ns}^c &= r_{j_1}^c \cdot \tilde{b}_{nj_1s} + r_{j_2}^c \cdot \tilde{b}_{nj_2s} + \cdots + r_{j_c}^c \cdot \tilde{b}_{nj_cs}
\end{aligned}$$

Solving this system of linear equations yields the data blocks $\{\tilde{b}_{ijk}\}$.

Finally, the MB-PMDDP scheme is secure against the following two attacks:

- **File swapping attack**. The file identifier ID_F is embedded into the block tag, and thus the CSP cannot use blocks from different files and pass the auditing procedures even if the owner uses the same secret key x with all his files.
- **Replay attack**. If the CSP does not honestly perform the data modification requests

issued by the owner, and sends old blocks and old tags during the challenge-response protocol, it will be detected. The block version $\mathcal{BV}_j$ – incremented in the MVT with each modify request – is embedded into the block tag as a countermeasure against such replay attacks.

4.7 Performance Analysis

Here we evaluate the performance of the presented schemes: MB-PMDDP and TB-PMDDP. The file F used in our performance analysis is of size 64MB with 4KB block size. Without loss of generality, we assume that the desired security level is 128-bit. Thus, we utilize an elliptic curve defined over Galois field $GF(p)$ with $|p| = 256$ bits (a point on this curve can be represented by 257 bits using compressed representation [14]), and a cryptographic hash of size 256 bits (*e.g.*, SHA-256).

The computation cost for the two schemes is estimated in terms of the used cryptographic operations, which are notated in Table 4.1. $\mathbb{G}$ indicates a group of points over a suitable elliptic curve in the bilinear pairing.

Table 4.1: Notation of cryptographic operations

Notation	Description	Notation	Description
h_{SHA}	Cryptographic hashing	$\mathcal{M}_{\mathbb{Z}_p}$	Multiplication in $\mathbb{Z}_p$
$\mathcal{H}_{\mathbb{G}}$	Hashing to $\mathbb{G}$	$\mathcal{A}_{\mathbb{Z}_p}$	Addition in $\mathbb{Z}_p$
$\mathcal{E}_{\mathbb{G}}$	Exponentiation in $\mathbb{G}$	$\mathcal{P}$	Bilinear pairing
$\mathcal{M}_{\mathbb{G}}$	Multiplication in $\mathbb{G}$	E_K	Encryption under a key K

Let n, m, and s denote the number of copies, the number of blocks per copy, and the number of sectors per block, respectively. Let c denotes the number of blocks to be challenged, and $|F|$ denotes the size of the file copy. Let the keys used with the PRP and the PRF be of size 128 bits. Table 4.2 presents a theoretical analysis for the setup, storage, communication, computation, and dynamic operations costs of the two schemes: MB-PMDDP and TB-PMDDP.

Table 4.2: Performance of the MB-PMDDP and TB-PMDDP schemes.

Costs		MB-PMDDP	TB-PMDDP
System Setup	Tag Generation	$(s+1)nm\,\mathcal{E}_{\mathbb{G}}+nm\,\mathcal{H}_{\mathbb{G}}$ $+(sn+n-1)m\,\mathcal{M}_{\mathbb{G}}$	$(s+1)nm\,\mathcal{E}_{\mathbb{G}}+nm\,\mathcal{H}_{\mathbb{G}}$ $+(sn+n-1)m\,\mathcal{M}_{\mathbb{G}}$
	Metadata Generation	—	$2nm\,h_{SHA}$
Storage	File Copies	$n\|F\|$	$n\|F\|$
	CSP Overhead	$257m$ bits	$257m$ + $(512m$ - $256)n$ bits
	Verifier Overhead	$64m$ bits	256 bits
Communication	Challenge	256 + $\log_2(c)$ *bits*	256 + $\log_2(c)$ *bits*
	Response	257 + $256sn$ bits	257 + $256sn$ + $(256\log_2(m) + 257)cn$ bits ‡
Computation	Proof	$c\,\mathcal{E}_{\mathbb{G}}+(c-1)\,\mathcal{M}_{\mathbb{G}}+csn\,\mathcal{M}_{\mathbb{Z}_p}$ $+\ (c-1)sn\,\mathcal{A}_{\mathbb{Z}_p}$	$c\,\mathcal{E}_{\mathbb{G}}+(c-1)\,\mathcal{M}_{\mathbb{G}}+csn\,\mathcal{M}_{\mathbb{Z}_p}$ $+\ (c-1)sn\,\mathcal{A}_{\mathbb{Z}_p}+cn\,\mathcal{H}_{\mathbb{G}}$
	Verification	$2\mathcal{P}+(c+s+1)\,\mathcal{E}_{\mathbb{G}}+c\,\mathcal{H}_{\mathbb{G}}$ $+(c+s-1)\,\mathcal{M}_{\mathbb{G}}+s(n-1)\mathcal{A}_{\mathbb{Z}_p}$	$(c\log_2(m)+2)n\,h_{SHA}$ ‡ $+\ 2\mathcal{P}+(c+s)\,\mathcal{E}_{\mathbb{G}}$ $+\ (cn+s-1)\,\mathcal{M}_{\mathbb{G}}+s(n-1)\,\mathcal{A}_{\mathbb{Z}_p}$
Dynamic Operations	Communication	“Request”	“Request” + $O(n\log_2(m))$
	Owner Computation (Modify/Insert/Append)	$n\,E_K+(s+1)n\,\mathcal{E}_{\mathbb{G}}+n\,\mathcal{H}_{\mathbb{G}}$ $+\ (sn+n-1)\,\mathcal{M}_{\mathbb{G}}$	$n\,E_K+(s+1)n\,\mathcal{E}_{\mathbb{G}}+n\,\mathcal{H}_{\mathbb{G}}$ $+\ (sn+n-1)\,\mathcal{M}_{\mathbb{G}}$
	State Update	—	$n\,\mathcal{H}_{\mathbb{G}}+(2n\log_2(m)+3n)\,h_{SHA}$

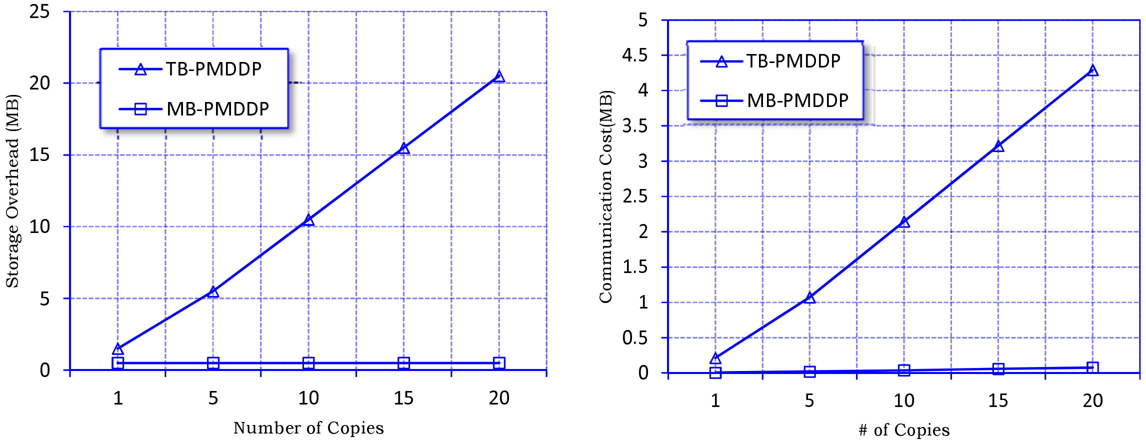
‡ $\log_2(m)$ is the upper bound of the authentication path length when c >1.

4.7.1 Comments

Sytem Setup. Table 4.2 shows that the setup cost of the MB-PMDDP scheme is less than that of the TB-PMDDP scheme. The TB-PMDDP scheme takes some extra cryptographic hash operations to prepare the MHTs for the file copies to generate the metadata $\mathcal{M}$.

Storage overhead. It is the additional space used to store some information other than the outsourced file copies $\widetilde{\mathbb{F}}$ ($n|F|$ bits are used to store $\widetilde{\mathbb{F}}$). The storage overhead on the CSP for the MB-PMDDP scheme is much less than that of the TB-PMDDP model. Both schemes need some additional space to store the aggregated block tags $\Phi = \{\sigma_j\}_{1\leq j\leq m}$, where σ_j is a group element that can be represented by 257 bits. Besides Φ, the TB-PMDDP scheme needs to store an MHT for each file copy which costs additional storage space on the cloud servers. The MHTs can be computed on the fly during the operations of the TB-PMDDP scheme. This slight modification can reduce the storage overhead on the remote servers, but it will negatively affect the overall system performance. The MHTs are needed through each dynamic operation of the file blocks and through the verification phase of the system. Thus, being not explicitly stored on the CSP can influence the system performance. For different number of copies of the file F, Figure 4.6a presents the CSP storage overhead (in MB) of the two schemes. An important feature of the MB-PMDDP scheme is that the CSP storage overhead is independent of the number of copies n, while it is linear in n for the TB-PMDDP scheme. As shown in Figure 4.6a, for 20 copies of the file F the overheads on the CSP are 0.50MB and 20.50MB for the MB-PMDDP and TB-PMDDP schemes, respectively (about 97% reduction). Reducing the storage overhead is economically a key feature to reduce the fees paid by the customers.

Regarding the verifier storage overhead, the MB-PMDDP scheme keeps an MVT on the verifier side compared with $\mathcal{M}$ (one hash value) for the TB-PMDDP. It is important to note there is only *one* table for all file copies, which mitigates the storage overhead on the verifier side. An entry of the MVT is of size 8 bytes (two integers), and the total number of entries equals to the number of file blocks. During implementation the $\mathcal{SN}$ is not needed to be stored in the table; $\mathcal{SN}$ is considered to be the entry/table index (the MVT is implemented as a linked list). The size of the MVT for the file F is only 128KB



(a) CSP storage overhead (MB)

(b) Response communication cost (MB)

Figure 4.6: CSP storage overhead and communication cost of the MB-PMDDP and TB-PMDDP schemes.

for unlimited number of copies.

Communication cost. From Table 4.2, the communication cost of the MB-PMDDP scheme is much less than that of the TB-PMDDP scheme. During the response phase, the map-based scheme sends one element σ (257 bits) and $\mu = \{\mu_{ik}\}_{\substack{1\leq i\leq n\\ 1\leq k\leq s}}$, where μ_{ik} is represented by 256 bits. On the other hand, the tree-based approach sends $\langle \sigma, \mu, \{\mathcal{H}(\tilde{b}_{ij})\}_{\substack{1\leq i\leq n\\ (j,*)\in Q}}, \langle \mathbb{A}_{ij}\rangle_{\substack{1\leq i\leq n\\ (j,*)\in Q}}\rangle$, where each $\mathcal{H}(\tilde{b}_{ij})$ is represented by 257 bits, and $\mathbb{A}_{ij}$ is an authentication path of length $O(\log_2 m)$. Each node along $\mathbb{A}_{ij}$ is a cryptographic hash of size 256 bits. For different number of copies of the file F, the communication cost (in MB) during the *response* phase of the two schemes is depicted in Figure 4.6b. The response of the MB-PMDDP scheme for 20 copies of F is 0.078MB, while it is 4.29MB for the TB-PMDDP scheme (about 98% reduction). The challenge for both schemes is about 34 bytes.

Computation cost. The computation cost of the two schemes is estimated in terms of the cryptographic operations (see Table 4.1) needed to generate the proof $\mathbb{P}$ and check the verification equation that validates $\mathbb{P}$. As observed from Table 4.2, the cost expression of the proof for the MB-PMDDP scheme has two terms linear in the number of copies n, while the TB-PMDDP scheme has three terms linear in n. Moreover, the MB-PMDDP scheme contains only one term linear in n in the verification cost expression, while there are three terms linear in n in the verification cost expression for the TB-PMDDP scheme. These terms affect the total computation time when dealing with a large number of copies in practical applications.

Dynamic operations cost. Table 4.2 also presents the cost of dynamic operations for both schemes. The communication cost of the MB-PMDDP scheme due to dynamic operations is less than that of the TB-PMDDP scheme for the owner sends a request $\langle ID_F, \mathit{BlockOp}, j, \{b_i^*\}_{1\leq i\leq n}, \sigma_j^*\rangle$ to the CSP and receives no information back. During the dynamic operations of the TB-PMDDP scheme, the owner sends a request to the CSP and receives the authentication paths which are of order $O(n\log_2(m))$. The authentication paths for updating 20 copies of $F \approx 8.75$KB.

The owner in both schemes uses $n\,E_K$ operations to create the distinct blocks $\{b_i^*\}_{1\leq i\leq n}$, and $(s+1)n\,\mathcal{E}_{\mathbb{G}}\;+\;n\,\mathcal{H}_{\mathbb{G}}+\;(sn+n\text{-}1)\,\mathcal{M}_{\mathbb{G}}$ to generate the aggregated tag σ_j^* (the delete operation does not require this computations). For the MB-PMDDP scheme, the owner updates the state (the map-version table) without usage of cryptographic operations (add, remove, or modify a table entry). On the other hand, updating the state (MHTs on the CSP and $\mathcal{M}$ on the owner) of the TB-PMDDP scheme costs $n\,\mathcal{H}_{\mathbb{G}} + (2n\log_2(m)+3n)\,h_{SHA}$ to update the MHTs of the file copies according to the required dynamic operations, and regenerate the new directory root that constructs a new $\mathcal{M}$. The experimental results show that updating the state of the TB-PMDDP scheme has insignificant effect on the total computation time of the dynamic operations.

4.8 Implementation and Experimental Evaluation

4.8.1 Implementation

We have implemented the proposed MB-PMDDP scheme and the TB-PMDDP reference model on top of Amazon Elastic Compute Cloud (Amazon EC2) [5] and Amazon Simple

Storage Service (Amazon S3) [6] cloud platforms. Through Amazon EC2 customers can lunch and manage Linux/Unix/Windows server instances (virtual servers) in Amazon's infrastructure. The number of EC2 instances can be automatically scaled up and down according to customers' needs. Amazon S3 is a web storage service to store and retrieve almost unlimited amount of data. Moreover, it enables customers to specify geographic locations for storing their data.

Our implementation of the presented schemes consists of three modules: `OModule` (owner module), `CModule` (CSP module), and `VModule` (verifier module). `OModule`, which runs on the owner side, is a library that includes KeyGen, CopyGen, TagGen, and PrepareUpdate algorithms. `CModule` is a library that runs on Amazon EC2 and includes ExecuteUpdate and Prove algorithms. `VModule` is a library to be run at the verifier side and includes the Verify algorithm.

In the experiments, we do not consider the system pre-processing time to prepare the different file copies and generate the tags set. This pre-processing is done only once during the life time of the system which may be for tens of years. Moreover, in the implementation we do not consider the time to access the file blocks, as the state-of-the-art hard drive technology allows as much as 1MB to be read in just few nanoseconds [80]. Hence, the total access time is unlikely to have substantial impact on the overall system performance.

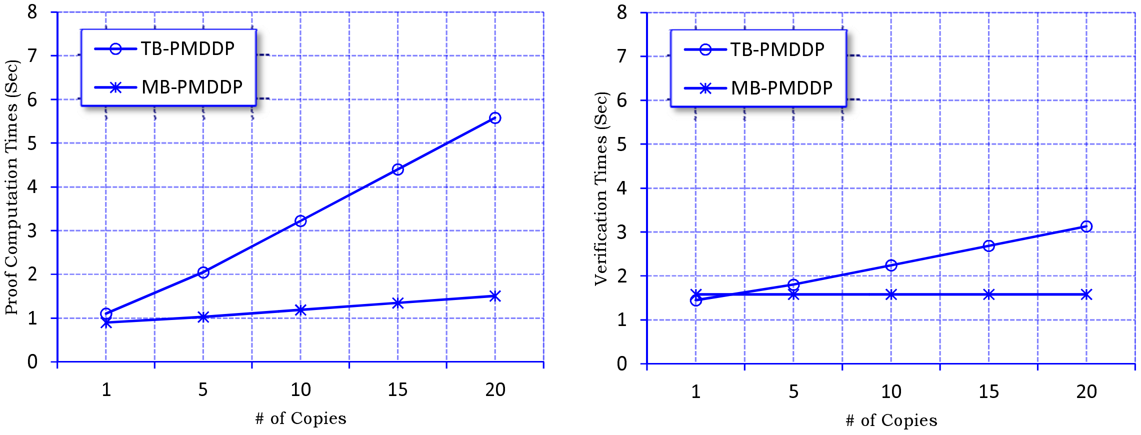
Implementation settings. A "large" Amazon EC2 instance is used to run `CModule`. Through this instance, a customers gets total memory of size 7.5GB and 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each). One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0 - 1.2GHz 2007 Opteron or 2007 Xeon processor [4]. The `OModule` and `VModule` are executed on a desktop computer with Intel(R) Xeon(R) 2GHz processor and 3GB RAM running Windows XP. We outsource copies of a data file of size 64MB to Amazon S3. Algorithms (encryption, pairing, hashing, etc.) are implemented using MIRACL library version 5.4.2. For 128-bit security level, the elliptic curve group we work on has a 256-bit group order. In the experiments, we utilize the Barreto-Naehrig(BN)[13] curve defined over prime field $GF(p)$ with $|p| = 256$ bits and embedding degree = 12 (the BN curve with these parameters is provided by the MIRACL library).

4.8.2 Experimental Evaluation

We compare the presented two schemes from various perspectives: proof computation times, verification times, and cost of dynamic operations. It has been reported in [8] that if the remote server is missing a fraction of the data, then the number of blocks that needs to be checked in order to detect server misbehavior with high probability is constant independent of the total number of file blocks. For example, if the server deletes 1% of the data file, the verifier only needs to check for $c = 460$-randomly chosen blocks of the file so as to detect this misbehavior with probability larger than 99%. Therefore, in our experiments, we use $c = 460$ to achieve a high probability of assurance.

Proof computation time. For different number of copies, Figure 4.7a presents the proof computation times (in seconds) to provide an evidence that the file copies are actually stored on the cloud servers in an updated, uncorrupted, and consistent state. The timing curve of the MB-PMDDP scheme is much less than that of the TB-PMDDP scheme. For 20 copies, the proof computation times for the MB-PMDDP and the TB-PMDDP schemes are 1.51 and 5.58 seconds, respectively (about 73% reduction in the computation time). As observed from Figure 4.7a, the timing curve of the TB-PMDDP scheme grows with increasing number of copies at a rate higher than that of the MB-PMDDP scheme. That is because the proof cost expression of the TB-PMDDP scheme contains more terms which are linear in the number of copies n (Table 4.2).

Verification time. Figure 4.7b presents the verification times (in seconds) to check the responses/proofs received from the CSP. The MB-PMDDP scheme has verification times less than that of the TB-PMDDP scheme. For 20 copies, the verification times for the MB-PMDDP and the TB-PMDDP schemes are 1.58 and 3.13 seconds, respectively (about 49% reduction in the verification time). The verification timing curve of the MB-PMDDP scheme is *almost* constant. There is a very small increase in the verification time with increasing number of copies. This is due to the fact that although the term $s(n-1)\mathcal{A}_{\mathbb{Z}_p}$ in the verification cost of the MB-PMDDP scheme is linear in n (Table 4.2), in our experiments its numerical value is quite small compared to those of the other terms in the cost expression. This feature makes the the MB-PMDDP scheme computationally



(a) CSP computation times (sec)

(b) Verifier computation times (sec)

Figure 4.7: Computation costs of the MB-PMDDP and TB-PMDDP schemes.

cost-effective and more efficient when verifying a large number of file copies.

Dynamic operations cost. For different number of copies, Table 4.3 presents the computation times (in seconds) on the owner side of the two schemes due to dynamic operations on a single block. The owner computation times for both schemes are *approximately* equal. The slight increase of the TB-PMDDP scheme is due to some additional hash operations required to regenerate a new directory root that constructs a new $\mathcal{M}$ (Table 4.2). As noted, the computation overhead on the owner side is practical. It takes about 5 seconds to modify/insert/append a block of size 4KB on 20 copies ($<$ 1 minute for 200 copies). In the experiments, we use only *one* desktop computer to accomplish the organization (data owner) work. In practice during updating the outsourced copies, the owner may choose to split the work among a few devices inside the organization or use a single device with a *multi-core* processor which is becoming prevalent these days, and thus the computation time on the owner side is significantly reduced in many applications.

Table 4.3: Owner computation times (sec) due to dynamic operations on a single block.

# of Copies	1	5	10	15	20
MB-PMDDP	0.261	1.304	2.608	3.913	5.217
TB-PMDDP	0.261	1.305	2.610	3.916	5.221

4.9 Identifying Corrupted Copies

Here, we utilize similar ideas to that applied in the previous chapter to identify which copies have been corrupted by slightly modifying the proposed MB-PMDDP scheme. Only if *all* copies stored by the CSP are intact and consistent, the generated proof $\mathbb{P} = \{\sigma, \mu\}$ will be valid and will pass the verification equation (4.1). Thus, the overall system integrity check fails when there is one or more corrupted copies. To address this issue and recognize the corrupted copies, a slight modification can be applied to the MB-PMDDP scheme. Through this modification, block tags are not aggregated, *i.e.*, $\Phi = \{\sigma_{ij}\}_{\substack{1\leq i\leq n \\ 1\leq j\leq m}}$. As a response to a challenge sent from the verifier, the CSP computes $\mu = \{\mu_{ik}\}_{\substack{1\leq i\leq n \\ 1\leq k\leq s}}$ as before, but $\sigma = \prod_{(j,r_j)\in Q}[\prod_{i=1}^{n}\sigma_{ij}]^{r_j} \in \mathbb{G}_1$. Upon receiving the proof $\mathbb{P} = \{\sigma, \mu\}$, the verifier first validates $\mathbb{P}$ using equation (4.1). In case of checking failure, the CSP sends $\sigma = \{\sigma_i\}_{1\leq i\leq n}$, where $\sigma_i = \prod_{(j,r_j)\in Q}\sigma_{ij}^{r_j}$. Thus, the verifier has two lists $\sigma\mathsf{List} = \{\sigma_i\}_{1\leq i\leq n}$ and $\mu\mathsf{List} = \{\mu_{ik}\}_{\substack{1\leq i\leq n \\ 1\leq k\leq s}}$ ($\mu\mathsf{List}$ is a two dimensional list).

The verifier can utilize a recursive divide-and-conquer approach (binary search) [46] to identify the indices of corrupted copies. The two lists $\sigma\mathsf{List}$ and $\mu\mathsf{List}$ are divided into halves: $\sigma\mathsf{List}\rightarrow$ ($\sigma\mathsf{Left}$:$\sigma\mathsf{Right}$), and $\mu\mathsf{List}\rightarrow$ ($\mu\mathsf{Left}$:$\mu\mathsf{Right}$). The verifier applies equation (4.1) recursively on $\sigma\mathsf{Left}$ with $\mu\mathsf{Left}$ and $\sigma\mathsf{Right}$ with $\mu\mathsf{Right}$. To generate one σ that is used during the recursive application of equation (4.1), individual tags in $\sigma\mathsf{Left}$ or $\sigma\mathsf{Right}$ are aggregated via multiplication. The procedural steps of identifying the indices of corrupted copies are indicated in Algorithm 2.

The input parameters of the BS (binary search) algorithm are $\sigma\mathsf{List}$, $\mu\mathsf{List}$, start (indicates the start index of the currently working lists), and end (indicates the last index of the currently working lists). Initially, the BS algorithm is called with ($\sigma\mathsf{List}$, $\mu\mathsf{List}$, 1, n).

Algorithm 2: BS(σList, μList, start, end)

```
begin
    $len \longleftarrow$ (end$-$start)$+1$            /* The list length */
    if $len$ = 1 then
        $\sigma \longleftarrow \sigma$List[start]      $\{\mu_k\}_{1\le k\le s} \longleftarrow \mu$List[start][$k$]
        $\hat{e}(\sigma, g) \overset{?}{=} \hat{e}(\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j)^{r_j} \cdot \prod_{k=1}^{s} u_k^{\mu_k}, y)$
        if NOT verified then
            invalidList.Add(start)
        end
    else
        $\sigma \longleftarrow \prod_{i=1}^{len} \sigma$List[start$+i-1$]
        $\{\mu_{ik}\}_{\substack{1\le i\le len\\ 1\le k\le s}} \longleftarrow \mu$List[start$+i-1$][$k$]
        $\hat{e}(\sigma, g) \overset{?}{=} \hat{e}([\prod_{(j,r_j)\in Q} \mathcal{H}(ID_F||\mathcal{BN}_j||\mathcal{BV}_j)^{r_j}]^{len} \cdot \prod_{k=1}^{s} u_k^{\sum_{i=1}^{len}\mu_{ik}}, y)$
        if NOT verified then
            /* work with the left and right halves of $\sigma$List and $\mu$List */
            mid$\longleftarrow \lfloor$(start+end)/2$\rfloor$          /* List middle */
            BS($\sigma$List, $\mu$List, start, mid)      /* Left part */
            BS($\sigma$List, $\mu$List, mid+1, end)  /* Right part */
        end
    end
end
```

A global data structure invalidList is used to store the indices of corrupted copies.

This slight modification to the proposed MB-PMDDP scheme will be associated with some extra storage overhead on the cloud servers. The CSP has to store mn tags for the file copies $\widetilde{\mathbb{F}}$ (m tags in the original version). Moreover, the challenge-response phase may be done in two rounds if the initial round to verify all copies fails.

We design experiments (using same file/parameters from Section 4.8) to show the effect of identifying the corrupted copies on the verification time. We generate 100 copies, which are verified in 1.584 seconds when all copies are accurate. A percentage – ranging from 1% to 20% – of the file copies is *randomly* corrupt. Figure 4.8 shows the verification time (in seconds) with different corrupted percentages. The verification time is about 20.58 seconds when 1% of the copies are invalid.

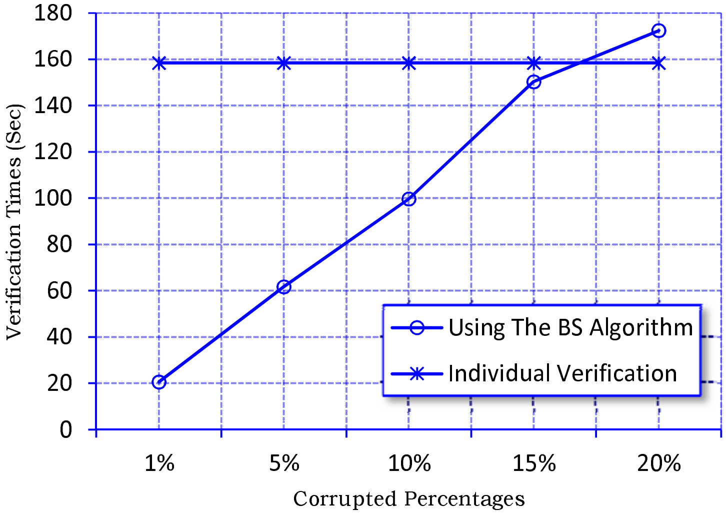


Figure 4.8: Verification times with different percentages of corrupted copies.

As observed from Figure 4.8, when the percentages of corrupted copies are up to 15% of the total copies, the performance of using the BS algorithm in the verification is more efficient than individual verification for each copy. It takes about 1.58 seconds to verify one copy, and thus individual verifications of 100 copies requires 100×1.58 = 158 seconds.

Shortly, a slight modification can be applied to the proposed scheme to support the feature of identifying the corrupted copies at the cost of some extra storage, communication, and computation overheads. It is crucial for the CSP – to remain in business and maintain a good reputation – to send valid responses to verifier's challenges. Invalid responses are sent in very rare situations, and thus the original version of the proposed scheme is used in most of the time.

Remark 6. To validate the integrity of outsourced data, the MB-PMDDP scheme relies on checking the relationship between the data blocks and their authentication tags. For the TB-PMDDP scheme, it relies on that relationship and the hash trees structure. Therefore, the divide-and-conquer approach used to identify corrupted copies cannot be directly applied to the TB-PMDDP scheme. Instead, a more complex method should be used to efficiently locate the corruption under a two-level hash tree.

4.10 Summary

Outsourcing data to remote servers has become a growing trend for many organizations to alleviate the burden of local data storage and maintenance. In this chapter, we have studied the problem of creating multiple copies of dynamic data file and verifying those copies stored on untrusted cloud servers.

We have proposed a new PDP scheme (referred to as MB-PMDDP), which supports outsourcing of multi-copy dynamic data, where the data owner is capable of not only archiving and accessing the data copies stored by the CSP, but also updating and scaling these copies on the remote servers. To the best of our knowledge, the proposed scheme is the first to address *multiple* copies of *dynamic* data. The interaction between the authorized users and the CSP is considered in our scheme, where the authorized users can seamlessly access a data copy received from the CSP using a single secret key shared with the data owner. Moreover, the proposed scheme supports public verifiability, enables arbitrary number of auditing, and allows *possession-free* verification where the verifier has the ability to verify the data integrity even though he neither possesses nor retrieves the file blocks from the server.

The MB-PMDDP scheme is proved to be secure against colluding servers, where only valid responses can be accepted from the CSP. Through performance analysis and experimental results, we have demonstrated that the proposed MB-PMDDP scheme outperforms the TB-PMDDP approach derived from a class of dynamic single-copy PDP models. The TB-PMDDP leads to high storage overhead on the remote servers and high computations on both the CSP and the verifier sides. The MB-PMDDP scheme significantly reduces the computation time during the challenge-response phase, which makes it more practical for applications where a large number of verifiers are connected to the CSP causing a huge computation overhead on the servers. Besides, it has lower storage overhead on the CSP, and thus reduces the fees paid by the cloud customers. The CSP's response and dynamic block operations of the map-based approach are done with less communication cost than that of the tree-based approach.

A slight modification can be done on the proposed scheme to support the feature of identifying the indices of corrupted copies. The corrupted data copy can be reconstructed even from a complete damage using duplicated copies on other servers.

Chapter 5

Dynamic Data and Mutual Trust

To complement our research, in this chapter we consider achieving mutual trust between data owners and cloud service providers (CSPs). We propose a cloud-based storage scheme [21, 17] that enables the data owner to utilize facilities offered by the CSP, and release concerns regarding confidentiality, integrity, and access control of the outsourced data. Meanwhile, a dishonest owner is not able to falsely accuse the CSP and claim data corruption over cloud servers to get illegal compensations. Section 5.1 highlights the motivation of this work. Section 5.2 contains some related concepts. Our system and assumptions are presented in Section 5.3. Some techniques pertaining to the design of our proposed scheme are reviewed in Section 5.4. The proposed scheme is elaborated in Section 5.5. Section 5.6 contains the security analysis of the proposed scheme. The performance analysis is shown in Section 5.7. Section 5.8 presents the implementation and experimental results. Section 5.9 discusses a slight modification to the proposed scheme to optimize the communication cost. The chapter is summarized in Section 5.10.

5.1 Introduction

Storage-as-a-Service (SaaS) offered by cloud service providers (CSPs) is a *paid* facility that enables organizations to outsource their sensitive data to be stored on remote servers.

Thus, SaaS reduces the maintenance cost and mitigates the burden of large local data storage at the organization's end. A data owner pays for a desired level of security and must get some compensation in case of any misbehavior committed by the CSP. On the other hand, the CSP needs a protection from any false accusation that may be claimed by the owner to get illegal compensations. This concern, if not properly handled, can cause the CSP to go out of business [73].

The material of this chapter addresses some important issues related to outsourcing the storage of data, namely *dynamic data*, *newness*, *mutual trust*, and *access control*. Dynamic scalability of data is one of the core design principles of data outsourcing for various applications. This means that the remotely stored data can be not only accessed by authorized users, but also updated and scaled by the owner. After updating, the authorized users should receive the latest version of the data (newness property), *i.e.*, a technique is required to detect whether the received data is stale. This issue is crucial for applications in which critical decisions are taken based on the received data. For example, in e-Health applications a physician may write a prescription based on a patient's medical history received from remote servers. If such medical data is not up-to-date, the given prescription may conflict with the patient's current circumstances causing severe health problems. Mutual trust between the data owner and the CSP is another imperative issue, which is addressed in this chapter. A mechanism is needed to determine the dishonest party, *i.e.*, misbehavior from any side should be detected and the responsible party is identified. Last but not least, the access control is considered, which allows the data owner to grant or revoke access rights to the outsourced data.

In this work, we propose a cloud-based storage scheme that allows the data owner to benefit from the facilities offered by the CSP and enables *indirect* mutual trust between them. The proposed scheme has four important features: (i) it allows the owner to outsource sensitive data to a CSP, and perform full block-level dynamic operations on the outsourced data, *i.e.*, block modification, insertion, deletion, and append, (ii) it ensures that authorized users (*i.e.*, those who have the right to access the owner's file) receive the latest version of the outsourced data, (iii) it enables indirect mutual trust between the owner and the CSP, and (iv) it allows the owner to grant or revoke access to the outsourced data. We discuss the security issues of the proposed scheme. Besides, we justify its performance through theoretical analysis and a prototype implementation on Amazon cloud platform to evaluate storage, communication, and computation overheads.

5.2 Related Concepts

Existing research close to our work can be found in the areas of integrity verification of outsourced data, cryptographic file systems in distributed networks, access control of outsourced data, and and non-repudiation protocols. The reader can refer to Chapter 2 for more details about verifying the integrity of data stored on remote servers using PDP (provable data possession) and POR (proof of retrievability) techniques.

5.2.1 Cryptographic File Systems

Kallahalla *et al.* [57] designed a cryptography-based file system called Plutus for secure sharing of data on untrusted servers. Some authorized users of the data have the privilege to read and write, while others can only read the data. In Plutus, a file-group represents a set of files with similar attributes, and each file-group is associated with a symmetric key called *file-lockbox* key. A data file is fragmented into blocks, where each block is encrypted with a unique symmetric key called a *file-block* key. The file-block key is further encrypted with the file-lockbox key of the file-group to which the data file belongs. If the data owner wants to share a file-group with a set of users, the file-lockbox key is just distributed to them. Plutus supports two operations on the file blocks: read and write/modify. Delete operation can be supported by overwriting an existing block with *null*.

Goh *et al.* [50] have presented SiRiUS, which is designed to be layered over existing file systems such as NFS (network file system) to provide end-to-end security. To enforce access control in SiRiUS, each data file (*d-file*) is attached with a metadata file (*md-file*) that contains an encrypted key block for each authorized user with some access rights (read or write). More specifically, the md-file represents the d-file's access control list (ACL). The d-file is encrypted using a file encryption key (FEK), and each entry in the ACL contains an encrypted version of the FEK under the public key of one authorized user. For large-scale sharing, the authors in [50] presented SiRiUS-NNL that uses NNL (Naor-Naor-Lotspiech) broadcast encryption algorithm [70] to encrypt the FEK of each file instead of encrypting using each authorized user's public key. SiRiUS supports two operations on the file blocks: read and write/modify. Other cryptographic file systems can be found in the literature, *e.g.*, [23, 31, 63]

5.2.2 Access Control for Outsourced Data

Commonly, traditional access control techniques assume the existence of the data owner and the storage servers in the same trust domain. This assumption, however, no longer holds when the data is outsourced to a remote CSP, which takes the full charge of the outsourced data management, and resides outside the trust domain of the data owner. A feasible solution can be presented to enable the owner to enforce access control of the data stored on a remote untrusted CSP. Through this solution, the data is encrypted under a certain key, which is shared only with the authorized users. The unauthorized users, including the CSP, are unable to access the data since they do not have the decryption key. This general solution has been widely incorporated into existing schemes [57, 50, 9, 40], which aim at providing data storage security on untrusted remote servers. Another class of solutions utilizes attribute-based encryption (ABE) to achieve fine-grained access control. ABE [53][1] is a public key cryptosystem for one-to-many communications that enables fine-grained sharing of encrypted data. The ABE associates the ciphertext with a set of attributes, and the private key with an access structure (policy). The ciphertext is decrypted if and only if the associated attributes satisfy the access structure of the private key. Access revocation in ABE-based systems is an issue since each attribute is conceivably shared by many users. Examples of ABE-based systems for achieving access control of remotely stored data are [90, 91, 71].

[1] The construction presented in [53] is called key-policy ABE (KP-ABE), which contrasts with another construction called ciphertext-policy ABE (CP-ABE) [22]. In the CP-ABE, an access structure is associated with the ciphertext, and a set of attributes is associated with the private key.

Based on proxy re-encryption [24], Ateniese *et al.* [9] have introduced a secure distributed storage protocol. In their protocol, a data owner encrypts the blocks with symmetric data keys, which are encrypted using a master public key. The owner keeps a master private key to decrypt the symmetric data keys. Using the master private key and the authorized user's public key, the owner generates proxy re-encryption keys. A semi-trusted server then uses the proxy re-encryption keys to translate a ciphertext into a form that can be decrypted only by granted users, and thus enforces access control for the data.

Vimercati *et al.* [40] have constructed a scheme for securing data on semi-trusted storage servers based on key derivation methods of [7]. In their scheme, a secret key is assigned to each authorized user, and data blocks are grouped based on users that can access these blocks. One key is used to encrypt all blocks in the same group. Moreover, the data owner generates public tokens to be used along with the user's secret key to derive decryption keys of specific blocks. The blocks and the tokens are sent to remote servers, which are not able to drive the decryption key of any block using just the public tokens. The approach in [40] allows the servers to conduct a second level of encryption (over-encryption) to enforce access control of the data. Repeated access grant and revocation may lead to a complicated hierarchy structure for key management [88].

The concept of over-encryption to enforce access control has also been used by Wang *et al.* [88]. In their scheme, the owner encrypts the data block-by-block, and constructs a binary tree of the block keys. The binary tree enables the owner to reduce the number of keys given to each user, where different keys in the tree can be generated from one common parent node. The remote storage server performs over-encryption to prevent revoked users from getting access to updated data blocks.

Popa *et al.* [73] have introduced a cryptographic cloud storage system called CloudProof that provides read and write data sharing and enforce access control. CloudProof has been designed to offer security guarantees in the service level agreements of cloud storage systems. It divides the security properties in four categories: confidentiality, integrity, read freshness, and write-serializability. CloudProof can provide these security properties using attestations (signed messages) and chain hash. Besides, it can detect and prove to a third party that any of these properties have been violated. Read freshness and write-serializability in CloudProof are guaranteed by periodic auditing in a *centralized* manner. The time is divided into epochs, which are time periods at the end of each the data owner performs the auditing process. The authorized users send the attestations – they receive from the CSP during the epoch – to the owner for auditing. CloudProof supports two operations on the file blocks: read and write/modify.

5.2.3 Non-Repudiation

Feng *et al.* [44] analyzed several existing cloud storage platforms, *e.g.*, Amazon S3, Microsoft Azure storage service, and Google secure data connector. They identified the problem of repudiation and presented a non-repudiation protocol for cloud data storage platforms to identify which party is dishonest: a data owner or a service provider. Their protocol is based on using *evidence* (extra information transmitted with the data to resolve repudiation when occurs) and trusted third party. The literature contains other work done on non-repudiation protocols; for example, see [45, 72, 78].

5.2.4 Discussion

Some aspects related to outsourcing data storage are beyond the setting of both PDP and POR, *e.g.*, enforcing access control, and ensuring the newness of data delivered to authorized users. Even in the case of dynamic PDP, a verifier can validate the correctness of data, but the server is still able to cheat and return stale data to authorized users after the auditing process is done. The schemes presented in [57, 50, 9, 40] have focused on access control and secure sharing of data on untrusted servers. The issues of full block-level dynamic operations (modify, insert, delete, and append), and achieving mutual trust between the data owners and the remote servers were outside the scope of those schemes. Although the authors of [88] have presented an efficient access control technique and handled full dynamic operations for the data over remote servers, data integrity, newness property, and mutual trust are not addressed. Authorized users in CloudProof [73] do not perform *immediate* checking for freshness of received data; the attestations are sent at the end of each epoch to the owner for completing the auditing task. Instantaneous validation of data freshness is crucial before taking any decisions based on the received data from the cloud. CloudProof guarantees write-serializability, which is outside the scope of our current work as we are focusing on owner-write-users-read applications. Performing dynamic operations at the block-level and enforcing access control for remote data were not considered in the non-repudiation protocol of [44].

5.3 Our System and Assumptions

System components and relations. The cloud computing storage model considered in this work consists of four main components as illustrated in Figure 5.1: (i) a data owner that can be an organization generating sensitive data to be stored in the cloud and made available for controlled external use; (ii) a CSP who manages cloud servers and provides paid storage space on its infrastructure to store the owner's files and make them available for authorized users; (iii) authorized users – a set of owner's clients who have the right to access the remote data; and (iv) a trusted third party (TTP), an entity who is trusted by all other system components, and has expertise and capabilities to detect and specify dishonest parties.

In Figure 5.1, the relations between different system components are represented by *double-sided* arrows, where solid and dashed arrows represent trust and distrust relations, respectively. For example, the data owner, the authorized users, and the CSP trust the TTP. On the other hand, the data owner and the authorized users have mutual distrust relations with the CSP. Thus, the TTP is used to enable *indirect* mutual trust between these three components. There is a direct trust relation between the data owner and the authorized users.

The storage model used in this work can be adopted by many practical applications. For example, e-Health applications can be envisioned by this model, where the patients' database that contains large and sensitive information can be stored on cloud servers. In these types of applications, a medical center can be considered as the data owner, physicians

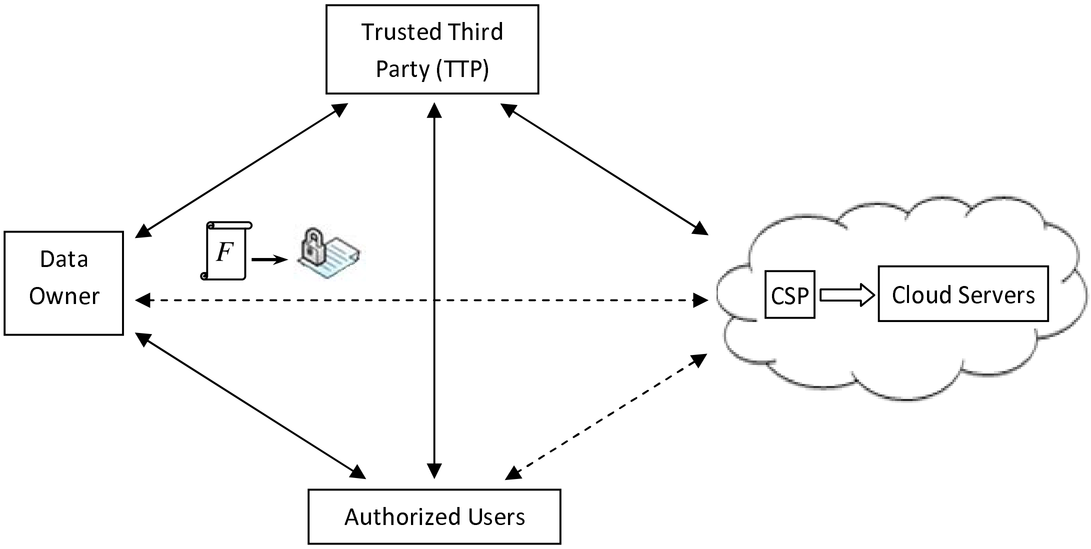


Figure 5.1: Cloud computing data storage system model.

as the authorized users who have the right to access the patients' medical history, and an independent-trusted organization as the TTP. Many other practical applications like financial, scientific, and educational applications can be viewed in similar settings.

Remark 1. The idea of using a third party auditor has been used before in outsourcing data storage systems, especially for customers with constrained computing resources and capabilities, *e.g.*, [82, 83, 54, 86]. The main focus of a third party auditor is to verify the data stored on remote servers, and give incentives to providers for improving their services. The proposed scheme in this work uses the TTP in a slightly different fashion. The auditing process of the data received from the CSP is done by the authorized users, and we resort to the TTP only to resolve disputes that may arise regarding data integrity or newness. Reducing the storage overhead on the CSP side is economically a key feature to lower the fees paid by the customers. Moreover, decreasing the overall computation cost in the system is another crucial aspect. To achieve these goals, a small part of the owner's work is delegated to the TTP.

Outsourcing, updating, and accessing. The data owner has a file F consisting of m

blocks to be outsourced to a CSP, where storage fees are pre-specified according to the used storage space. For confidentiality, the owner encrypts the data before sending to cloud servers. After data outsourcing, the owner can interact with the CSP to perform block-level operations on the file. These operations includes modify, insert, append, and delete specific blocks. In addition, the owner enforces access control by granting or revoking access rights to the outsourced data.

An authorized user sends a data-access request to the CSP, and receives the data file in an encrypted form that can be decrypted using a secret key generated by the authorized user (more details will be explained later). We assume that the interaction between the owner and the authorized users to authenticate their identities has already been completed, and it is not considered in this work. Moreover, all authorized users have the same privileges, *i.e.*, access hierarchy is outside the current scope.

The TTP is an independent entity, and thus has no incentive to collude with any party in the system. However, any possible leakage of data towards the TTP must be prevented to keep the outsourced data private. The TTP and the CSP are always online, while the owner is *intermittently* online. The authorized users are able to access the data file from the CSP even when the owner is offline. Throughout this chapter, the terms cloud server and cloud service provider are used interchangeably.

Threat model. The CSP is untrusted, and thus the confidentiality and integrity of data in the cloud may be at risk. For economic incentives and maintaining a reputation, the CSP may hide data loss (due to hardware failure, management errors, various attacks), or reclaim storage by discarding data that has not been or is rarely accessed. To save the computational resources, the CSP may totally ignore the data-update requests issued by the owner, or execute just a few of them. Hence, the CSP may return damaged or stale data for any access request from the authorized users. Furthermore, the CSP may not honor the access rights created by the owner, and permit unauthorized access for misuse of confidential data.

On the other hand, a data owner and authorized users may collude and falsely accuse the CSP to get a certain amount of reimbursement. They may dishonestly claim that data integrity over cloud servers has been violated, or the CSP has returned a stale file that does not match the most recent modifications issued by the owner.

Security requirements. *Confidentiality*: outsourced data must be protected from the TTP, the CSP, and users that are not granted access. *Integrity*: outsourced data is required to remain intact on cloud servers. The data owner and authorized users must be enabled to recognize data corruption over the CSP side. *Newness*: receiving the most recent version of the outsourced data file is an imperative requirement of cloud-based storage systems. There must be a detection mechanism if the CSP ignores any data-update requests issued by the owner. *Access control*: only authorized users are allowed to access the outsourced data. Revoked users can read unmodified data, however, they must not be able to read updated/new blocks. *CSP's defence*: the CSP must be safeguarded against false accusations that may be claimed by dishonest owner/users, and such a malicious behavior is required to be revealed.

Combining the confidentiality, integrity, newness, access control, and CSP's defence properties in the proposed scheme enables the mutual trust between the data owner and the CSP. Thus, the owner can benefit from the wide range of facilities offered by the CSP, and at the same time, the CSP can mitigate the concern of cheating customers.

5.4 System Preliminaries

5.4.1 Lazy Revocation

The proposed scheme in this work allows the data owner to revoke the right of some users for accessing the outsourced data. In lazy revocation, it is acceptable for revoked users to read (decrypt) *unmodified* data blocks. However, updated or new blocks must not be accessed by such revoked users. The notation of lazy revocation was first introduced in [49]. The idea is that allowing revoked users to read unchanged data blocks is not a significant loss in security. This is equivalent to accessing the blocks from cashed copies. Updated or new blocks following a revocation are encrypted under new keys. Lazy revocation trades re-encryption and data access cost for a degree of security. However, it causes fragmentation of encryption keys, *i.e.*, data blocks could have more than one key. Lazy revocation has been incorporated into many cryptographic systems [73, 88, 12, 75].

5.4.2 Key Rotation

Key rotation [57] is a technique in which a sequence of keys can be generated from an initial key and a master secret key. The sequence of keys has two main properties: (i) only the owner of the master secret key is able to generate the next key in the sequence from the current key, and (ii) any authorized user knowing a key in the sequence is able to generate all previous versions of that key. In other words, given the i-th key K_i in the sequence, it is computationally infeasible to compute keys $\{K_l\}$ for $l > i$ without having the master secret key, but it is easy to compute keys $\{K_j\}$ for $j < i$.

The first property enables the data owner to revoke access to the data by producing new keys in the sequence, which are used to encrypt updated/new blocks following a revocation (lazy revocation). It is intended to prevent a user revoked during the i-th time from getting access to data blocks encrypted during the l-th time for $l > i$.

The second property allows authorized users to maintain access to blocks that are encrypted under older versions of the current key. It enables the data owner to transfer only a single key K_i to authorized users for accessing all data blocks that are encrypted under keys $\{K_j\}_{j \leq i}$ (rather than transferring a potentially large set of keys $\{K_1, K_2, \dots, K_i\}$). Thus, the second property reduces the communication overhead on the owner side.

The proposed scheme in this work utilizes the key rotation technique [57]. Let $N = pq$ denote the RSA modulus ($p \,\&\, q$ are prime numbers), a public key $= (N, e)$, and a master secret key d. The key d is known only to the data owner, and $ed \equiv 1 \bmod (p-1)(q-1)$.

Whenever a user's access is revoked, the data owner generates a new key in the sequence (*rotating forward*). Let *ctr* indicate the index/version number of the current key in the keys sequence. The owner generates the next key by exponentiating K_{ctr} with the master secret key d: $K_{ctr+1} = K_{ctr}^d \bmod N$. Authorized users can recursively generate older versions of the current key by exponentiating with the public key component e: $K_{ctr-1} = K_{ctr}^{e} \bmod N$ (*rotating backward*). The RSA encryption is used as a pseudorandom number generator; it is unlikely that repeated encryption results in cycling, for otherwise, it can be used to factor the RSA modulus N [66].

5.4.3 Broadcast Encryption

Broadcast encryption (bENC) [26, 47] allows a broadcaster to encrypt a message for an arbitrary subset of a group of users. The users in the subset are only allowed to decrypt the message. However, even if all users outside the subset collude they cannot access the encrypted message. Such systems have the collusion resistance property, and are used in many practical applications including TV subscription services and DVD content protection. The proposed scheme in this work uses bENC [26] to enforce access control in outsourced data. The bENC [26] is composed of three algorithms: Setup, Encrypt, and Decrypt.

Setup. This algorithm takes as input the number of system users n. It defines a bilinear group $\mathbb{G}$ of prime order p with a generator g, a cyclic multiplicative group $\mathbb{G}_T$, and a bilinear map $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, which has the properties of bilinearity, computability, and non-degeneracy [65]. The algorithm picks a random $\alpha \in \mathbb{Z}_p$, computes $g_i = g^{(\alpha^i)} \in \mathbb{G}$ for $i = 1, 2, \ldots, n, n+2, \ldots, 2n$, and sets $v = g^{\gamma} \in \mathbb{G}$ for $\gamma \in_R \mathbb{Z}_p$. The outputs are a public key $PK = (g, g_1, \ldots, g_n, g_{n+2}, \ldots, g_{2n}, v) \in \mathbb{G}^{2n+1}$, and n private keys $\{d_i\}_{1 \leq i \leq n}$, where $d_i = g_i^{\gamma} \in \mathbb{G}$.

Encrypt. This algorithm takes as input a subset $S \subseteq \{1, 2, \ldots, n\}$, and a public key PK. It outputs a pair (Hdr, K), where Hdr is called the header (broadcast ciphertext), and K is a message encryption key. Hdr $= (C_0, C_1) \in \mathbb{G}^2$, where for $t \in_R \mathbb{Z}_p, C_0 = g^t$ and $C_1 = (v \cdot \prod_{j \in S} g_{n+1-j})^t$. The key $K = \hat{e}(g_{n+1}, g)^t$ is used to encrypt a message M (symmetric encryption) to be broadcast to the subset S.

Decrypt. This algorithm takes as input a subset $S \subseteq \{1, 2, \ldots, n\}$, a user-ID $i \in \{1, 2, \ldots, n\}$, the private key d_i for user i, the header Hdr $= (C_0, C_1)$, and the public key

PK. If $i \in S$, the algorithm outputs the key $K = \hat{e}(g_i, C_1)/\hat{e}(d_i \cdot \prod_{\substack{j \in S \\ j \neq i}} g_{n+1-j+i}, C_0)$, which can be used to decrypt the encrypted version of M.

In the above construction of the bENC [26], a private key contains only one element of $\mathbb{G}$, and the broadcast ciphertext (Hdr) consists of two elements of $\mathbb{G}$. On the other hand, the public key PK is comprised of $2n+1$ elements of $\mathbb{G}$. A second construction, which is a generalization of the first one was presented in [26] to trade the PK size for the Hdr size. The main idea is to run multiple parallel instances of the first construction, where each instance can broadcast to at most B users. Setting $B = \lfloor \sqrt{n} \rfloor$ results in a system with $O(\sqrt{n})$ elements of $\mathbb{G}$ for each of PK and Hdr. The private key is still just one element.

In this work, we utilize the second construction to achieve a balance between the sizes of PK and Hdr. For an organization (data owner) with 10^5 users, each of PK and Hdr contains only 317 elements of $\mathbb{G}$.

5.5 Proposed Cloud-Based Storage Scheme

5.5.1 Warmup Discussion

Before presenting our main scheme, we discuss a straightforward solution. Once the data has been outsourced to a remote CSP, which may not be trustworthy, the owner loses the direct control over the sensitive data. This lack of control raises the data owner's concerns about the integrity of data stored in the cloud. Conversely, a dishonest owner may falsely claim that the data stored in the cloud is corrupted to get some compensation. This mutual distrust between the data owner and the CSP, if not properly handled, may hinder the successful deployment of cloud architecture.

A straightforward solution to detect cheating from any side is through using authentication tags (digital signatures). For a file $F = \{b_j\}_{1 \leq j \leq m}$, the owner attaches a tag $\texttt{OWN}\sigma_j$ with each block before outsourcing. The tags are generated per block not per file to enable dynamic operations at the block level without retrieving the whole outsourced file. The owner sends $\{b_j, \mathtt{OWN}\sigma_j\}_{1 \leq j \leq m}$ to the CSP, where the tags $\{\mathtt{OWN}\sigma_j\}_{1 \leq j \leq m}$ are first verified. In case of failed verification, the CSP rejects to store the data blocks and asks the owner to re-send the correct tags. If the tags are valid, both the blocks and the tags are stored on the cloud servers. The tags $\{\mathtt{OWN}\sigma_j\}_{1 \leq j \leq m}$ achieve non-repudiation from the owner side. When an authorized user (or the owner) requests to retrieve the data file, the CSP sends $\{b_j, \mathtt{OWN}\sigma_j, \mathtt{CSP}\sigma_j\}_{1 \leq j \leq m}$, where $\mathtt{CSP}\sigma_j$ is the CSP's signature/tag on $b_j||\mathtt{OWN}\sigma_j$. The authorized user first verifies the tags $\{\mathtt{CSP}\sigma_j\}_{1 \leq j \leq m}$. In case of failed verification, the user asks the CSP to re-perform the transmission process. If $\{\mathtt{CSP}\sigma_j\}_{1 \leq j \leq m}$ are valid tags, the user then verifies the owner's tag $\mathtt{OWN}\sigma_j$ on the block $b_j \; \forall \; j$. If any tag $\mathtt{OWN}\sigma_j$ is not verified, this indicates the corruption of data over the cloud servers. The CSP cannot repudiate such corruption for the owner's tags $\{\mathtt{OWN}\sigma_j\}_{1 \leq j \leq m}$ are previously verified and stored by the CSP along with the data blocks. Since the CSP's signatures $\{\mathtt{CSP}\sigma_j\}_{1 \leq j \leq m}$ are attached with the received data, a dishonest owner cannot falsely accuse the CSP regarding data integrity.

Although the previous straightforward solution can detect cheating from either side, it cannot guarantee the newness property of the outsourced data; the CSP can replace the new blocks and tags with old versions without being detected (*replay attack*). The above solution increases the storage overhead – especially for large files in order of gigabytes – on the cloud servers as each outsourced block is attached with a tag. Moreover, there is an increased computation overhead on different system components; the data owner generates a signature for each block, the CSP performs a signature verification for each outsourced block, and the authorized user (or the owner) verifies two signatures for each received block from the cloud servers. Thus, for a file F containing m blocks, the straightforward solution requires $2m$ signature generations and $3m$ signature verifications, which may be computationally a challenging task for large data files. For example, if the outsourced file is of size 1GB with 4KB block size, the straightforward solution requires 2^{19} signature generations and 3×2^{18} signature verifications.

If the CSP receives the data blocks from a trusted entity (other than the owner), the block tags and the signature operations are not needed since the trusted entity has no incentive for repudiation or collusion. Therefore, delegating a small part of the owner's work to the TTP reduces both the storage and computation overheads. However, the outsourced data must be kept private and any possible leakage of data towards the TTP must be prevented.

5.5.2 Overview and Rationale

The proposed scheme in this work addresses important issues related to outsourcing data storage: dynamic data, newness, mutual trust, and access control. The owner is allowed to update and scale the outsourced data file. Validating such dynamic data and its newness property requires the knowledge of some metadata that reflects the most recent modifications issued by the owner. Moreover, it requires the awareness of block indices to guarantee that the CSP has inserted, added, or deleted the blocks at the requested positions. To this end, the proposed scheme is based on using *combined* hash values and a small data structure, which we call block status table (BST). The TTP establishes the mutual trust among different system components in an indirect way.

For enforcing access control of the outsourced data, the proposed scheme utilizes and combines three cryptographic techniques: bENC, lazy revocation, and key rotation. The bENC enables a data owner to encrypt some secret information to only authorized users allowing them to access the outsourced data file. Through lazy revocation, revoked users can read unmodified data blocks, while updated/new blocks are encrypted under new keys generated from the secret information broadcast to the authorized users. Using key rotation, the authorized users are able to access both updated/new blocks and unmodified ones that are encrypted under older versions of the current key.

5.5.3 Notations

- F is a data file to be outsourced, and is composed of a sequence of m blocks, *i.e.*, $F = \{b_1, b_2, \ldots, b_m\}$.
- h is a cryptographic hash function.

- DEK is a data encryption key.
- E_{DEK} is a symmetric encryption algorithm under DEK, *e.g.*, AES (advanced encryption standard) [38, 1].
- E^{-1}_{DEK} is a symmetric decryption algorithm under DEK.
- $\widetilde{F}$ is an encrypted version of the file blocks.
- FH_{TTP} is a combined hash value for $\widetilde{F}$, and is computed and stored by the TTP.
- TH_{TTP} is a combined hash value for the BST, and is computed and stored by the TTP.
- ctr is a counter kept by the data owner to indicate the version of the most recent key.
- $Rot = \langle ctr, \mathsf{bENC}(K_{ctr})\rangle$ is a rotator, where $\mathsf{bENC}(K_{ctr})$ is a broadcast encryption of the key K_{ctr}.
- $\oplus$ is an XOR operator.

5.5.4 Block Status Table

The block status table (BST) is a small *dynamic* data structure used to reconstruct and access file blocks outsourced to the CSP. The BST consists of three columns: serial number ($\mathcal{SN}$), block number ($\mathcal{BN}$), and key version ($\mathcal{KV}$). $\mathcal{SN}$ is an indexing to the file blocks. It indicates the physical position of each block in the data file. $\mathcal{BN}$ is a counter used to make a logical numbering/indexing to the file blocks. Thus, the relation between $\mathcal{BN}$ and $\mathcal{SN}$ can be viewed as a mapping between the logical number $\mathcal{BN}$ and the physical position $\mathcal{SN}$. $\mathcal{KV}$ indicates the version of the key that is used to encrypt each block in the file.

The BST is implemented as a linked list to simplify the insertion and deletion of table entries. During implementation, $\mathcal{SN}$ is not needed to be stored in the table; $\mathcal{SN}$ is considered to be the entry/table index. Thus, each table entry contains just two integers $\mathcal{BN}$ and $\mathcal{KV}$ (8 bytes), *i.e.*, the total table size is $8m$ bytes, where m is the number of file blocks.

When a data file is initially created, the owner initializes both *ctr* and $\mathcal{KV}$ of each block to 1. If block modification or insertion operations are to be performed following a revocation, *ctr* is incremented by 1 and $\mathcal{KV}$ of that modified/new block is set to be equal to *ctr*.

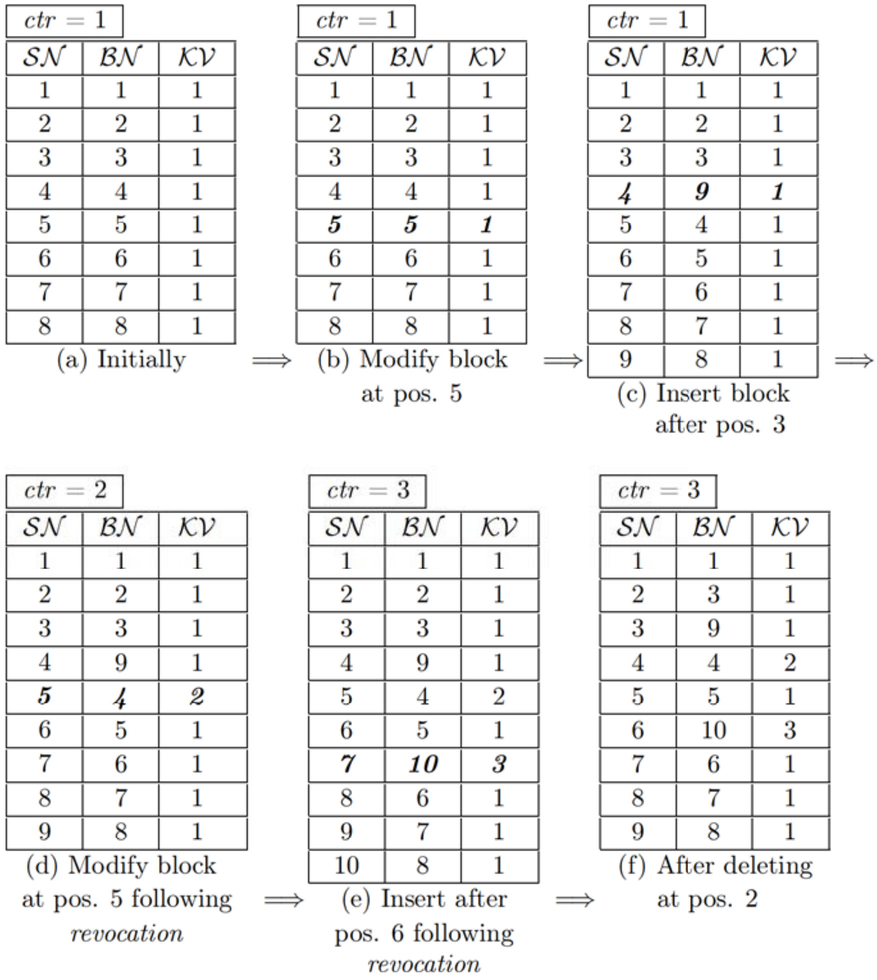
Figure 5.2 shows some examples demonstrating the changes in the BST due to dynamic operations on a data file $F = \{b_j\}_{1\leq j\leq 8}$. When the file blocks are initially created (Figure 5.2a), *ctr* is initialized to 1, $\mathcal{SN}_j = \mathcal{BN}_j = j$, and $\mathcal{KV}_j = 1$: $1 \leq j \leq 8$. Figure 5.2b shows no change for updating the block at position 5 since no revocation is performed. To insert a new block after position 3 in the file F, Figure 5.2c shows that a new entry $\langle 4, 9, 1\rangle$ is inserted in the BST after $\mathcal{SN}_3$, where 4 is the physical position of the newly inserted block, 9 is the new logical block number computed by incrementing the maximum of all previous logical block numbers, and 1 is the version of the key used for encryption.

A first revocation in the system increments *ctr* by 1 ($ctr = 2$). Modifying the block at position 5 following a revocation (Figure 5.2d) results in setting $\mathcal{KV}_5 = ctr$. Thus, the table entry at position 5 becomes $\langle 5, 4, 2\rangle$. Figure 5.2e shows that a new block is to be inserted after position 6 following a second revocation, which increments *ctr* to be 3. In Figure 5.2e, a new table entry $\langle 7, 10, 3\rangle$ is inserted after $\mathcal{SN}_6$, where $\mathcal{KV}_7$ is set to be equal to *ctr* (the most recent key version). Deleting a block at position 2 from the data file requires deleting the table entry at $\mathcal{SN}_2$ and shifting all subsequent entries one position up (Figure 5.2f). Note that during all dynamic operations, $\mathcal{SN}$ indicates the actual physical positions of the data blocks in F.

5.5.5 Procedural Steps of the Proposed Scheme

- ***Setup and File Preparation***. The setup is done only once during the life time of the data storage system, which may be for tens of years. The system setup has two parts: one is done on the owner side, and the other is done on the TTP side.

    - **Owner Role**. The data owner initializes *ctr* to 1, and generates an initial secret key K_{ctr}/K_1. K_{ctr} can be rotated forward following user revocations, and



$ctr = 1$

$\mathcal{SN}$	$\mathcal{BN}$	$\mathcal{KV}$
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1
6	6	1
7	7	1
8	8	1

(a) Initially ⟹

$ctr = 1$

$\mathcal{SN}$	$\mathcal{BN}$	$\mathcal{KV}$
1	1	1
2	2	1
3	3	1
4	4	1
5	***5***	***1***
6	6	1
7	7	1
8	8	1

(b) Modify block at pos. 5 ⟹

$ctr = 1$

$\mathcal{SN}$	$\mathcal{BN}$	$\mathcal{KV}$
1	1	1
2	2	1
3	3	1
4	***9***	***1***
5	4	1
6	5	1
7	6	1
8	7	1
9	8	1

(c) Insert block after pos. 3 ⟹

$ctr = 2$

$\mathcal{SN}$	$\mathcal{BN}$	$\mathcal{KV}$
1	1	1
2	2	1
3	3	1
4	9	1
5	***4***	***2***
6	5	1
7	6	1
8	7	1
9	8	1

(d) Modify block at pos. 5 following *revocation* ⟹

$ctr = 3$

$\mathcal{SN}$	$\mathcal{BN}$	$\mathcal{KV}$
1	1	1
2	2	1
3	3	1
4	9	1
5	4	2
6	5	1
7	***10***	***3***
8	6	1
9	7	1
10	8	1

(e) Insert after pos. 6 following *revocation* ⟹

$ctr = 3$

$\mathcal{SN}$	$\mathcal{BN}$	$\mathcal{KV}$
1	1	1
2	3	1
3	9	1
4	4	2
5	5	1
6	10	3
7	6	1
8	7	1
9	8	1

(f) After deleting at pos. 2

Figure 5.2: Changes in the BST due to different dynamic operations on a file $F = \{b_j\}_{1 \leq j \leq 8}$. $\mathcal{SN}$ is the serial number, $\mathcal{BN}$ is the block number, and $\mathcal{KV}$ is the key version.

rotated backward to enable authorized users to access blocks that are encrypted under older versions of K_{ctr}.

For a file $F = \{b_j\}_{1 \leq j \leq m}$, the owner generates a BST with $\mathcal{SN}_j = \mathcal{BN}_j = j$ and $\mathcal{KV}_j = ctr$. To achieve privacy-preserving, the owner creates an encrypted file version $\widetilde{F} = \{\tilde{b}_j\}_{1 \leq j \leq m}$, where $\tilde{b}_j = E_{DEK}(\mathcal{BN}_j || b_j)$ and $DEK = h(K_{ctr})$.[2] Moreover, the owner creates a rotator $Rot = \langle ctr, \mathsf{bENC}(K_{ctr}) \rangle$, where bENC enables only authorized users to decrypt K_{ctr} and access the outsourced file. The owner sends $\{\widetilde{F}, \text{BST}, Rot\}$ to the TTP, and deletes the data file from its local storage.

Embedding $\mathcal{BN}_j$ with the block b_j during the encryption process helps in reconstructing the file blocks in the correct order. If the encrypted blocks $\{\tilde{b}_j\}_{1 \leq j \leq m}$ are not corrupted over cloud servers, but randomly delivered to an authorized user, the latter can utilize the embedded $\mathcal{BN}_j$ and the BST to orderly reconstruct the data file F. More details will be explained later.

- ⧫ **TTP Role**. As previously explained, a small part of the owner's work is delegated to the TTP to reduce the storage overhead and lower the overall system computation. For the TTP to resolve disputes that may arise regarding data integrity/newness, it computes and locally stores combined hash values for the encrypted file $\widetilde{F}$ and the BST. The TTP computes $FH_{TTP} = \oplus_{j=1}^{m} h(\tilde{b}_j)$ and $TH_{TTP} = \oplus_{j=1}^{m} h(\mathcal{BN}_j || \mathcal{KV}_j)$, then sends $\{\widetilde{F}, \text{BST}\}$ to the CSP (it is possible for the owner to use the technique of one-sender-multiple-receiver (OSMR) transmission to send $\{\widetilde{F}, \text{BST}\}$ to both the TTP and the CSP). The TTP keeps only FH_{TTP} and TH_{TTP} on its local storage.

Remark 2. The BST is used by the authorized users to reconstruct and access the outsourced data file. The proposed scheme in this work assumes that the data owner is *intermittently* online and the authorized users are enabled to access the data file even when the owner is offline. To this end, the CSP stores a copy of the BST along

[2] Hash in needed to compress the size of K_{ctr}

with the outsourced data file. When an authorized user requests to access the data, the CSP responds by sending both the BST and the encrypted file $\widetilde{F}$.

Moreover, the BST is used during each dynamic operation on the outsourced data file, where one table entry is modified/inserted/deleted with each dynamic change on the block level. If the BST is stored only on the CSP side, it needs to be retrieved and validated each time the data owner wants to issue a dynamic request on the outsourced file. To avoid such communication and computation overheads, the owner keeps a local copy of the BST, and thus there are two copies of the BST: one is stored on the owner side referred to as BST_O, and the other is stored on the CSP side referred to as BST_C. Recall that the BST is a small dynamic data structure with a table entry size = 8 bytes. For 1GB file with 4KB block size, the BST size is only 2MB (0.2% of the file size). Table 5.1 summarizes the data stored by each component in the proposed scheme.

Table 5.1: Data stored by each component in the proposed scheme.

Owner	TTP	CSP
ctr, K_{ctr}, BST_O	Rot, FH_{TTP}, TH_{TTP}	$\widetilde{F}$, BST_C

- ***Dynamic Operations on the Outsourced Data***. The dynamic operations in the proposed scheme are performed at the block level via a request in the general form $\langle$BlockOp, $\text{TEntry}_{\textsf{BlockOp}}$, j, $\mathcal{KV}_j$, $h(\tilde{b}_j)$, `RevFlag`, $b^*\rangle$, where BlockOp corresponds to block modification (denoted by BM), block insertion (denoted by BI), or block deletion (denoted by BD). $\text{TEntry}_{\textsf{BlockOp}}$ indicates an entry in BST_O corresponding to the issued dynamic request. The parameter j indicates the block index on which the dynamic operation is to be performed, $\mathcal{KV}_j$ is the value of the key version at index j of BST_O before running a modification operation, and $h(\tilde{b}_j)$ is the hash value of the block at index j before modification/deletion. `RevFlag` is a 1-bit flag (true/false and is initialized to false) to indicate whether a revocation has been performed, and b^* is the new block value.

- **Modification**. Data modification is one of the most frequently used dynamic operations in the outsourced data. For a file $F = \{b_1, b_2, \ldots, b_m\}$, suppose the owner wants to modify a block b_j with b'_j. Figure 5.3 describes the steps performed by each system component (owner, CSP, and TTP) during block modification. The owner uses the technique of OSMR transmission to send the modify request to both the CSP and the TTP.

  The TTP updates the combined hash value FH_{TTP} for $\widetilde{F}$ through the step $FH_{TTP} = FH_{TTP} \oplus h(\tilde{b}_j) \oplus h(\tilde{b}'_j)$, which simultaneously replaces the hash of the old block $h(\tilde{b}_j)$ with the new one $h(\tilde{b}'_j)$. This is possible due to the basic properties of the $\oplus$ operator. The same idea is used when `RevFlag` $=$ true to update the combined hash value TH_{TTP} on the TTP side by replacing the hash of the old table entry at index j with the hash of the new value.

- **Insertion**. In a block insertion operation, the owner wants to insert a new block $\bar{b}$ *after* index j in a file $F = \{b_1, b_2, \ldots, b_m\}$, *i.e.*, the newly constructed file $F' = \{b_1, b_2, \ldots, b_j, \bar{b}, \ldots, b_{m+1}\}$, where $b_{j+1} = \bar{b}$. The block insertion operation changes the logical structure of the file, while block modification does not. Figure 5.4 describes the steps performed by each system component (owner, CSP, and TTP) during block insertion.

- **Append**. Block append operation means adding a new block at the end of the outsourced data. It can simply be implemented via insert operation after the last block of the data file.

- **Deletion**. Block deletion operation is the opposite of the insertion operation. When one block is deleted all subsequent blocks are moved one step forward. Figure 5.5 describes the steps performed by each system component (owner, CSP, and TTP) during block deletion. The step $FH_{TTP} = FH_{TTP} \oplus h(\tilde{b}_j)$ is used to delete the hash value of the block $\tilde{b}_j$ from the combined hash FH_{TTP} (properties of $\oplus$ operator). The same idea is used with the TH_{TTP} value.

```
/* Modification of a block $b_j$ with $b'_j$ for the outsourced file */
/* RevFlag is initialized to false */
Data Owner
  1. If the access of one or more users has been revoked then
     (a) Rolls $K_{ctr}$ forward (using key rotation)
     (b) Increments $ctr = ctr + 1$, and sets RevFlag = true
     (c) Copies $\mathcal{KV}_j$ from BST$_O$ to $\overline{\mathcal{KV}}_j$ (i.e., $\overline{\mathcal{KV}}_j = \mathcal{KV}_j$)
     (d) Sets $\mathcal{KV}_j = ctr$ in BST$_O$, and generates $Rot = \langle ctr, \mathsf{bENC}(K_{ctr})\rangle$
     (e) Sends $Rot$ to the TTP
  2. Creates an encrypted block $\tilde{b}'_j = E_{DEK}(\mathcal{BN}_j||b'_j)$, where $DEK = h(K_{ctr})$
  3. Forms a block-modify table entry TEntry$_{\mathsf{BM}}$ = $\{\mathcal{BN}_j, \mathcal{KV}_j\}$
  4. Sends a modify request $\langle$BM, TEntry$_{\mathsf{BM}}$, $j$, $\overline{\mathcal{KV}}_j$, $h(\tilde{b}_j)$, RevFlag, $\tilde{b}'_j\rangle$ to both the
     CSP and the TTP (OSMR transmission), where $h(\tilde{b}_j)$ is the hash of the outsourced
     block to be modified. The $\overline{\mathcal{KV}}_j$ is not sent in the modify request if RevFlag = false
  5. The CSP accepts the modify request only if $\{\mathcal{BN}_j, \overline{\mathcal{KV}}_j\}$ sent from the owner matches
     $\{\mathcal{BN}_j, \mathcal{KV}_j\}$ in BST$_C$, and $h(\tilde{b}_j)$ is equal to the hash of the block $\tilde{b}_j$ on the cloud
     server (to guarantee that correct values are sent to the TTP)
CSP /* upon accepting the modify request from the owner */
  1. Replaces the block $b_j$ with $b'_j$ in the outsourced file $\widetilde{F}$
  2. If RevFlag = true then
     Updates the table entry at index $j$ of BST$_C$ using TEntry$_{\mathsf{BM}}$ components
TTP
  1. Updates $FH_{TTP} = FH_{TTP} \oplus h(\tilde{b}_j) \oplus h(\tilde{b}'_j)$
  2. If RevFlag = true then
     (a) Updates the previously stored $Rot$ with the newly received value
     (b) Updates $TH_{TTP} = TH_{TTP} \oplus h(\mathcal{BN}_j||\overline{\mathcal{KV}}_j) \oplus h(\mathcal{BN}_j||\mathcal{KV}_j)$
```

Figure 5.3: Block modification procedure in the proposed scheme.

```
/* Insertion of a block $\bar{b}$ after index $j$ in the outsourced file */
/* RevFlag is initialized to false */
Data Owner
    1. If the access of one or more users has been revoked then
        (a) Rolls $K_{ctr}$ forward (using key rotation)
        (b) Increments $ctr = ctr + 1$, and sets $\texttt{RevFlag} = \text{true}$
        (c) Generates $Rot = \langle ctr, \mathsf{bENC}(K_{ctr}) \rangle$
        (d) Sends $Rot$ to the TTP
    2. Constructs a new block-insert table entry $\text{TEntry}_{\mathsf{BI}} = \{\mathcal{BN}_{j+1}, \mathcal{KV}_{j+1}\} = \{1 + Max\{\mathcal{BN}_j\}_{1 \leq j \leq m}, ctr\}$, and inserts this entry in $\text{BST}_O$ after index $j$
    3. Creates an encrypted block $\tilde{\bar{b}} = E_{DEK}(\mathcal{BN}_j || \bar{b})$, where $DEK = h(K_{ctr})$
    4. Sends an insert request $\langle \mathsf{BI}, \text{TEntry}_{\mathsf{BI}}, j, null, null, \texttt{RevFlag}, \tilde{\bar{b}} \rangle$ to both the CSP and the TTP (OSMR transmission)
CSP /* upon receiving the insert request from the owner */
    1. Inserts the block $\tilde{\bar{b}}$ after index $j$ in the outsourced file $\widetilde{F}$
    2. Inserts the table entry $\text{TEntry}_{\mathsf{BI}}$ after index $j$ in the $\text{BST}_C$
TTP
    1. Updates $FH_{TTP} = FH_{TTP} \oplus h(\tilde{\bar{b}})$
    2. Updates $TH_{TTP} = TH_{TTP} \oplus h(\mathcal{BN}_{j+1} || \mathcal{KV}_{j+1})$
    3. If $\texttt{RevFlag} = \text{true}$ then
       Replaces the previously stored $Rot$ with the newly received value
```

Figure 5.4: Block insertion procedure in the proposed scheme.

/ Deletion of a block b_j from the outsourced file */*

Data Owner

1. Copies the entry at index j from BST_O to a block-delete table entry $\mathrm{TEntry}_{\mathsf{BD}} = \{\mathcal{BN}_j, \mathcal{KV}_j\}$
2. Deletes the entry at index j from BST_O
3. Sends a delete request $\langle \mathsf{BD}$, $\mathrm{TEntry}_{\mathsf{BD}}$, j, *null*, $h(\tilde{b}_j)$, false, *null*$\rangle$ to both the CSP and the TTP (OSMR transmission), where $h(\tilde{b}_j)$ is the hash of the outsourced block to be deleted
4. The CSP accepts the delete request only if $\mathrm{TEntry}_{\mathsf{BD}}$ sent from the owner matches $\{\mathcal{BN}_j, \mathcal{KV}_j\}$ in BST_C and $h(\tilde{b}_j)$ is equal to the hash of the block $\tilde{b}_j$ on the cloud server (to guarantee that correct values are sent to the TTP).

CSP */* upon receiving the delete request from the owner */*

1. Deletes the block at index j (block $\tilde{b}_j$) from the outsourced file $\widetilde{F}$
2. Deletes the entry at index j from the BST_C

TTP

1. Updates $FH_{TTP} = FH_{TTP} \oplus h(\tilde{b}_j)$
2. Updates $TH_{TTP} = TH_{TTP} \oplus h(\mathcal{BN}_j || \mathcal{KV}_j)$

Figure 5.5: Block deletion procedure in the proposed scheme.

- ***Data Access and Cheating Detection***. Figure 5.6 shows the verifications performed for the data received from the CSP, and presents how authorized users get access to the outsourced file.

  An authorized user sends a data-access request to both the CSP and the TTP to access the outsourced file. For achieving non-repudiation, the CSP generates two signatures σ_F and σ_T for $\widetilde{F}$ and BST_C, respectively. The authorized user receives $\{\widetilde{F}$, BST_C σ_F, $\sigma_T\}$ from the CSP, and $\{FH_{TTP}$, TH_{TTP}, $Rot\}$ from the TTP. The authorized user verifies the signatures, and proceeds with the data access procedure only if both signatures are valid.

  The authorized user verifies the contents of BST_C entries by computing a combined hash value $TH_U = \oplus_{j=1}^{m} h(\mathcal{BN}_j||\mathcal{KV}_j)$, and comparing it with the authentic value TH_{TTP} received from the TTP. If the user claims that $TH_U \neq TH_{TTP}$, a report is issued to the owner and the TTP is invoked to determine the dishonest party.

  In case of $TH_U = TH_{TTP}$, the authorized user continues to verify the contents of the file $\widetilde{F}$. A combined hash value $FH_U = \oplus_{j=1}^{m} h(\tilde{b}_j)$ is computed and compared with FH_{TTP}. If there is a dispute that $FH_U \neq FH_{TTP}$, the owner is informed and we resort to the TTP to resolve such a conflict.

  For the authorized user to access the encrypted file $\widetilde{F} = \{\tilde{b}_j\}_{1\leq j\leq m}$, BST_C and Rot are used to generate the key DEK that decrypts the block $\tilde{b}_j$. The component $\mathsf{bENC}(K_{ctr})$ of Rot is decrypted to get the most recent key K_{ctr}. Using the key rotation technique, the authorized user rotates K_{ctr} backward with each block until it reaches the version that is used to decrypt the block $\tilde{b}_j$. Both ctr and the key version $\mathcal{KV}_j$ can determine how many rotation steps for K_{ctr} with each block $\tilde{b}_j$. Decrypting the block $\tilde{b}_j$ returns $(\mathcal{BN}_j||b_j)$. Both $\mathcal{BN}_j$ and BST_C are utilized to get the physical block position $\mathcal{SN}_j$ into which the block b_j is inserted, and thus the file F is reconstructed in plain form.

1. An authorized user sends a data-access request to both the CSP and the TTP

2. The CSP responds by sending the outsourced file $\widetilde{F} = \{\tilde{b}_j\}_{1 \leq j \leq m}$ associated with a signature σ_F (CSP's signature on the entire file), and sending BST$_C$ associated with a signature σ_T (CSP's signature on the entire table) to the authorized user

3. The authorized user verifies σ_F and σ_T, and accepts the data only if σ_F and σ_T are valid signatures

4. The TTP sends FH_{TTP} , TH_{TTP}, and $Rot = \langle ctr, \mathsf{bENC}(K_{ctr}) \rangle$ to the authorized user

5. **Verification of the BST$_C$ entries**

   (a) The authorized user computes $TH_U = \oplus_{j=1}^{m} h(\mathcal{BN}_j || \mathcal{KV}_j)$

   (b) **If** the authorized user claims that $TH_U \neq TH_{TTP}$ **then** report "integrity violation" to the owner and invoke cheating detection procedure (Figure 5.7)

6. **Verification of the data file $\widetilde{F}$**

   (a) The authorized user computes $FH_U = \oplus_{j=1}^{m} h(\tilde{b}_j)$

   (b) **If** the authorized user claims that $FH_U \neq FH_{TTP}$ **then** report "integrity violation" to the owner and invoke cheating detection procedure (Figure 5.7)

7. **Data access**

   (a) The authorized user gets K_{ctr} by decrypting $\mathsf{bENC}(K_{ctr})$ part in Rot

   (b)
```
for $j = 1$ to $m$ do
/* rotate backward the current $K_{ctr}$ to the version that is used to decrypt $\tilde{b}_j$ */
  – Set $K_j = K_{ctr}$
  – for $i = 1$ to $ctr$ - $\mathcal{KV}_j$ do
        $K_j = (K_j)^e \bmod N$ /* $N$ is RSA modulus and $(N, e)$ is the public key */
    end for
  – $(\mathcal{BN}_j || b_j) = E_{DEK}^{-1}(\tilde{b}_j)$, where $DEK = h(K_j)$
  – Get the physical position $\mathcal{SN}_j$ of $b_j$ using $\mathcal{BN}_j$ and BST$_C$
  – The authorized user places $b_j$ in the correct order of the decrypted file $F$
end for
```

Figure 5.6: Data access procedure in the proposed scheme.

Optimization. In Figure 5.6, the backward key rotation done in the *inner* for loop of step 7.b can be highly optimized by computing a set of keys $Q = \{K_i\}$ from K_{ctr}. Each key K_i in Q is the result of rotating K_{ctr} backward $ctr - i$ times. For example, if $ctr = 20$, a set $Q = \{K_1, K_5, K_{10}, K_{15}\}$ can be computed from K_{ctr}. To decrypt a block $\tilde{b}_j$, the authorized user chooses one key K_i from Q, which has the minimum *positive* distance $i - \mathcal{KV}_j$. The key K_i is then rotated backward to get the actual key that is used to decrypt the block $\tilde{b}_j$. A relatively large portion of the outsourced data is kept unchanged on the CSP, and thus K_1 from Q can be used to decrypt many blocks without any further key rotation. The size of the set Q is negligible compared with the size of the received data file.

Figure 5.7 shows how the TTP determines the dishonest party in the system. The TTP verifies the signatures σ_T and σ_F, which are previously verified and accepted by the authorized user. If any signature is invalid, this indicates that the owner/user is dishonest for corrupting either the data or the signatures. In case of valid signatures, the TTP computes temporary combined hash values $TH_{temp} = \oplus_{j=1}^{m} h(\mathcal{BN}_j || \mathcal{KV}_j)$ and $FH_{temp} = \oplus_{j=1}^{m} h(\tilde{b}_j)$. If $TH_{temp} \neq TH_{TTP}$ or $FH_{temp} \neq FH_{TTP}$, this indicates that the CSP is dishonest for sending corrupted data to the authorized user, otherwise the owner/user is dishonest for falsely claiming integrity violation of received data.

5.6 Security Analysis

In this section, we investigate the security of the proposed scheme by analyzing its fulfillment of the security requirements described in Section 5.3, namely, confidentiality, integrity, newness, access control, and CSP's defence.

Data confidentiality. For this requirement, we need to prove that the CSP, the TTP, and unauthorized users cannot access the outsourced data.

Theorem 1. *The proposed scheme preserves the confidentiality of the outsourced data against the CSP, the TTP, and unauthorized users.*

```
Cheating Detection Procedure: determination of the dishonest party.
The TTP is invoked to determine which component is misbehaving as follows.

1. The TTP verifies $\sigma_T$ and $\sigma_F$

2. If any signature verification fails then
       TTP reports "dishonest owner/user" and exits

3. The TTP computes $TH_{temp} = \oplus_{j=1}^{m} h(\mathcal{BN}_j || \mathcal{KV}_j)$ and $FH_{temp} = \oplus_{j=1}^{m} h(\tilde{b}_j)$

4. If $TH_{temp} \neq TH_{TTP}$ or $FH_{temp} \neq FH_{TTP}$ then
       TTP reports "dishonest CSP" and exits /* data is corrupted */
   else
       TTP reports "dishonest owner/user" and exits /* data is NOT corrupted */
```

Figure 5.7: Cheating detection procedure in the proposed scheme.

Proof (Sketch). Before outsourcing a data file $F = \{b_j\}_{1 \leq j \leq m}$, the owner generates an encrypted version $\widetilde{F} = \{\tilde{b}_j\}_{1 \leq j \leq m}$, where $\tilde{b}_j = E_{DEK}(\mathcal{BN}_j || b_j)$, and $DEK = h(K_{ctr})$. The encrypted data file $\widetilde{F}$ is sent to the TTP for computing FH_{TTP} and to the CSP for storage.

Based on the security of the underlying symmetric encryption algorithm E_{DEK}, the confidentiality of the outsourced file $\widetilde{F}$ is preserved on the CSP side (*e.g.*, our proposed scheme utilizes AES – a standardized encryption algorithm by NIST [1] – with 128-bit security level to achieve a robust security requirement).

The data confidentiality on the TTP side is based on the security of the underlying broadcast encryption algorithm bENC. To decrypt $\widetilde{F}$, the key K_{ctr} is needed to generate DEK. The TTP stores $Rot = \langle ctr, \mathsf{bENC}(K_{ctr}) \rangle$. Thus, for the TTP to get K_{ctr}, bENC must be broken (the proposed scheme utilizes bENC [26], which is proved to be semantically secure).

bENC also prevents unauthorized users from getting K_{ctr} to access the data file $\widetilde{F}$. Moreover, based on the hardness of the RSA problem [28], revoked users who possess K_l ($l < ctr$) are not able to generate K_{ctr}. Hence, such revoked users can access only stale data blocks, while updated or new blocks encrypted using K_{ctr} are kept secret. □

Detection of data integrity violation. We want the assurance that any corruption to the outsourced data file $\widetilde{F}$ or the table BST_C on cloud servers can be detected. We prove this feature for $\widetilde{F}$ and the same ideas are applied to BST_C. The proof depends on the preimage and second-preimage resistance properties of the cryptographic hash function h.

Definitions.

1. **Preimage resistance**: given a hash value y, it is computationally infeasible to find any input x such that $h(x) = y$ [77]. Input x is called a preimage of y.

2. **Second-preimage**: given an input x, it is computationally infeasible to find a second input $x' \neq x$ such that $h(x) = h(x')$ [77].

Theorem 2. *Given a cryptographic hash function h with preimage and second-preimage resistance properties along with the non-collusion incentive of the TTP, any attempt to violate the integrity of outsourced data file on cloud servers will be detected.*

Proof. We prove the theorem by contradiction. The goal of a dishonest CSP is to send a corrupted or stale data file to authorized users without being detected. Let $\widetilde{D} = \{\tilde{d}_j\}_{1 \leq j \leq m}$ be the data file received by an authorized user from the CSP during the data access phase of the proposed scheme, where $\{\tilde{d}_j\}_{1 \leq j \leq m}$ denotes the file blocks. Let $\widetilde{F} = \{\tilde{b}_j\}_{1 \leq j \leq m}$ be the actual outsourced data file. The authorized user receives the authentic FH_{TTP} from the TTP, computes $FH_U = \oplus_{j=1}^{m} h(\tilde{d}_j)$, and checks $FH_U \overset{?}{=} FH_{TTP}$. If $FH_U \neq FH_{TTP}$, then $\widetilde{D} \neq \widetilde{F}$ (data has been corrupted on cloud servers).

For violating data integrity without being detected there are two possible scenarios. First, the CSP has to generate values $\{h_j^*\}_{1 \leq j \leq m}$ such that $FH_{TTP} = \oplus_{j=1}^{m} h_j^*$, and at least one $h_j^* \neq h(\tilde{b}_j)$. If the CSP could create $\widetilde{D} = \{\tilde{d}_j\}_{1 \leq j \leq m}$ such that $h_j^* = h(\tilde{d}_j)\ \forall j$, the cheating is possible. Due to the preimage-resistance property of h (one-way function), the CSP cannot generate such data file $\widetilde{D}$, *i.e.*, $\tilde{d}_j$ must be equal to $\tilde{b}_j\ \forall j$.

Second, the received data $\widetilde{D}$ has at least one block $\tilde{d}_j \neq \tilde{b}_j$, but $h(\tilde{d}_j) = h(\tilde{b}_j)\ \forall j$ to guarantee that $FH_U = FH_{TTP}$. Due to the second-preimage resistance property of h, there is no such data file $\widetilde{D}$, *i.e.*, $\tilde{d}_j$ must be equal to $\tilde{b}_j\ \forall j$. □

Remark 3. The encrypted block $\tilde{b}_j = E_{DEK}(\mathcal{BN}_j||b_j)$. Block number $\mathcal{BN}_j$ is embedded with the block b_j to be used along with BST_C to orderly reconstruct the plain file F if the blocks $\{b_j\}_{1\leq j\leq m}$ are randomly received. Using a proof similar to that of Theorem 2, we can show that BST_C cannot be corrupted without being detected. Swapping the entries of BST_C without changing their contents can cause the file F to be reconstructed in an incorrect order. Although the CSP has no incentive and no financial benefit of doing such swapping, one can defend this weird behavior by storing the $\mathcal{BN}$ column of BST_O on the TTP side. The authorized user can retrieve and use this column during the data access phase. This countermeasure adds little extra storage on the TTP ($4m$ bytes).

Assurance of newness property. Assurance of newness property is identical to detection of data integrity violation. With each dynamic operation (modification, insertion, deletion, append), the TTP updates the values FH_{TTP} and TH_{TTP} to reflect the most recent state of the outsourced data. Thus, the CSP cannot respond to an access request by sending stale data without being detected.

Enforcement of access control. The proposed scheme combines the techniques of broadcast encryption, key rotation, and lazy revocation to enforce access control of outsourced data.

Theorem 3. *The data owner can grant or revoke access to users for outsourced data.*

Proof (Sketch). The owner creates $Rot = \langle ctr, \mathsf{bENC}(K_{ctr})\rangle$ and encrypts the outsourced data using $DEK = h(K_{ctr})$. Broadcast encryption bENC allows the data owner to encrypt the key K_{ctr} for an arbitrary subset of a group of users. According to the security strength of bENC [26], the users in the subset are only allowed to decrypt K_{ctr} and access the outsourced data.

It is acceptable for revoked users to access unmodified data blocks. However, updated/new blocks must be inaccessible by such revoked users. In case of data modification/insertion following a revocation, the data owner rolls K_{ctr} forward: $K_{ctr+1} = K_{ctr}^d \bmod N$, and then increments ctr by 1 (preparing for next rotation). Since factoring the RSA modulus N is assumed to be intractable [28], revoked users who possess K_l ($l < ctr$) are not able to generate K_{ctr}. Thus, combining broadcast encryption, key rotation, and lazy revocation achieves access control in the proposed scheme.

Detection of dishonest owner/user. The CSP signs both the file $\widetilde{F}$ and the table BST_C. The generated signatures σ_F and σ_T are sent along with $\widetilde{F}$ and BST_C to an authorized user during the data access phase. If the signature scheme is existentially unforgeable, the owner/user cannot falsely accuse the CSP regarding data integrity; the TTP performs signature verifications if there is a claim of data corruption. Recall that the signatures σ_F and σ_T are accepted by the authorized user as valid signatures in the beginning of the data access phase (step 3 in Figure 5.6).

5.7 Performance Analysis

5.7.1 Settings and Overheads

The data file F used in our performance analysis is of size 1GB with 4KB block size. Without loss of generality, we assume that the desired security level is 128-bit. Thus, we utilize a cryptographic hash h of size 256 bits (*e.g.*, SHA-256), an elliptic curve defined over Galois field $GF(p)$ with $|p| = 256$ bits (used for bENC), and BLS (Boneh-Lynn-Shacham) signature [27] of size 256 bits (used to compute σ_F and σ_T).

Here we evaluate the performance of the proposed scheme by analyzing the storage, communication, and computation overheads. We investigate overheads that the proposed scheme brings to a cloud storage system for *static* data with only *confidentiality* requirement. This investigation demonstrates whether the features of our scheme come at a reasonable cost. The computation overhead is estimated in terms of the used cryptographic functions, which are notated in Table 5.2.

Table 5.2: Notation of cryptographic functions

Notation	Description
h	Cryptographic hashing
$\mathcal{FR}$	Forward key rotation
$\mathcal{BR}$	Backward key rotation
$\mathcal{S}_\sigma$	Signature genration
$\mathcal{V}_\sigma$	Signature verification
E_{DEK}	Symmetric encryption using the key DEK
bENC^{-1}	Decryption of bENC

Let m and n denote the number of file blocks and the total number of system users, respectively. Table 5.3 presents a theoretical analysis for the storage, communication, and computation overheads of the proposed scheme. Table 5.4 summarizes the storage and communication overheads for our data file F (1GB with 4KB block size) and 100,000 authorized users.

5.7.2 Comments

Storage overhead. It is the additional storage space used to store necessary information other than the outsourced file $\widetilde{F}$. The overhead on the owner side is due to storing BST_O. An entry of BST_O is of size 8 bytes (2 integers), and the total number of entries equals the number of file blocks m. During implementation $\mathcal{SN}$ is not needed to be stored in BST_O; $\mathcal{SN}$ is considered to be the entry/table index (BST_O is implemented as a linked list). The size of BST_O for the file F is only 2MB (0.2% of F). BST_O size can be further reduced if the file F is divided into larger blocks (*e.g.*, 16KB). Like the owner, the storage overhead on the CSP side comes from the storage of BST_C. To resolve disputes that may arise regarding data integrity or newness property, the TTP stores FH_{TTP} and TH_{TTP}, each of size 256 bits. Besides, the TTP stores $Rot = \langle ctr, \mathsf{bENC}(K_{ctr})\rangle$ that enables the data owner to enforce access control for the outsourced data. ctr is 4 bytes, and bENC has storage complexity $O(\sqrt{n})$, which is practical for an organization (data owner) with n = 100,000

Table 5.3: Overhead analysis of the proposed scheme. The overheads shown in square brackets are *not* always present and are incurred when revocation(s) actually occur.

Overheads	Operations	Owner	User	CSP	TTP
Storage (in bytes)		$8m$	—	$8m$	$68{+}32\sqrt{n}$
Communication (in bytes)	Dynamic Operations	$45 + [8 + 32\sqrt{n}]$	—	—	—
	Data Access	—	—	$64 + 8m$	$68 + 32\sqrt{n}$
Computation	Dynamic Operations	$h + E_{DEK} + [\mathcal{FR} + \mathsf{bENC}]$	—	—	$2\,h + [2\,h]$ ‡
	Data Access	—	$2\,\mathcal{V}_\sigma + 3m\,h + \mathsf{bENC}^{-1} + [\mathcal{BR}]$ ‡	$2\,\mathcal{S}_\sigma$	—
	Cheating Detection				$2\,\mathcal{V}_\sigma + [2m\,h]$ ‡

‡ The cost of $\oplus$ is usually negligible and is omitted in the overhead expressions.

Table 5.4: Storage and communication overheads for the data file F (1GB with 4KB block size) and 100,000 authorized users. The values shown in square brackets are not always present and are incurred when revocation(s) actually occur.

Overheads	Operations	Owner	User	CSP	TTP
Storage		2MB	—	2MB	≈ 10KB †
Communication	Dynamic Operations	45 bytes + [≈ 10KB]	—	—	—
	Data Access	—	—	≈ 2MB	≈ 10KB

† Storage overhead is independent of F.

users. A point on the elliptic curve used to implement bENC can be represented by 257 bits ($\approx$ 32 bytes) using compressed representation [14]. Therefore, the storage overhead on the TTP side is close to 10KB, which is independent of the outsourced file size. Overall, the storage overhead of the proposed scheme for the file F is less than 4.01MB ($\approx$ 0.4% of F).

Communication overhead. It is the additional information sent along with the outsourced data blocks. During dynamic operations, the communication overhead on the owner side comes from the transmission of a block operation $\mathsf{BlockOP}$ (can be represented by 1 byte), a table entry $\text{TEntry}_{\mathsf{BlockOP}}$ (8 bytes), and a block index j (4 bytes). If a block is to be modified following a revocation process, $\mathcal{KV}_j$ (4 bytes) is sent to the TTP. Moreover, in case of a block modification/deletion, the owner sends a hash (32 bytes) of the block to be modified/deleted to the TTP for updating FH_{TTP}. Recall that the owner also sends *Rot* ($4 + 32\sqrt{n}$ bytes) to the TTP if block modifications/insertions are to be performed following user revocations. Therefore, in the worst case scenario (*i.e.*, block modifications following revocations), the owner's overhead is less than 10KB. The *Rot* represents the major factor in the communication overhead, and thus the overhead is only 45 bytes if block modification/deletion operations are to be preformed without revocations (only 13 bytes for insertion operations). In practical applications, the frequency of dynamic requests to the outsourced data is higher than that of user revocations. Hence, the communication overhead due to dynamic changes on the data is about 1% of the block size (the block is 4KB in our analysis).

As a response to access the outsourced data, the CSP sends the file along with σ_F (32 bytes), σ_T (32 bytes), and BST_C ($8m$ bytes). Moreover, the TTP sends FH_{TTP} (32 bytes), TH_{TTP} (32 bytes), and *Rot*. Thus, the communication overhead due to data access is $64 + 8m$ bytes on the CSP side, and $68 + 32\sqrt{n}$ bytes on the TTP side. Overall, to access the file F, the proposed scheme has communication overhead close to 2.01MB ($\approx$ 0.2% of F).

Computation overhead. A cloud storage system for static data with only confidentiality requirement has computation cost for encrypting the data before outsourcing and decrypting the data after being received from the cloud servers. For the proposed scheme, the computation overhead on the owner side due to dynamic operations (modification/insertion) comes from computing $DEK = h(K_{ctr})$ and encrypting the updated/inserted block, *i.e.*, the overhead is one hash and one encryption operations. If a block modification/insertion operation is to be performed following a revocation of one or more users, the owner performs $\mathcal{FR}$ to roll K_{ctr} forward, and bENC to generate the *Rot*. Hence, the computation overhead on the owner side for the dynamic operations is $h + E_{DEK} + \mathcal{FR} + \mathsf{bEnc}$ (worst case scenario). Updating BST_O and BST_C is done without usage of cryptographic operations (add, remove, or modify a table entry).

To reflect the most recent version of the outsourced data, the TTP updates the values FH_{TTP} and TH_{TTP}. If no revocation has been performed before sending a modify request, only FH_{TTP} is updated on the TTP side. Therefore, the maximum computation overhead on the TTP side for updating both FH_{TTP} and TH_{TTP} is $4\,h$.

Before accessing the data received from the CSP, the authorized user verifies two signatures (generated by the CSP), BST_C entries, and the data file. These verifications cost $2\,\mathcal{V}_\sigma + 2m\,h$. Moreover, the authorized user decrypts $\mathsf{bENC}(K_{ctr})$ part in the *Rot* to get K_{ctr}. For each received block, K_{ctr} is rotated backward to obtain the actual key that is used to decrypt the data block. The optimized way of key rotation (using the set Q) highly affects the performance of data access; many blocks need a few or no rotations. Moreover, one hash operation is performed per block to compute DEK. Overall, the computation overhead due to data access is $2\,\mathcal{V}_\sigma + 3m\,h + \mathsf{bENC}^{-1} + [\mathcal{BR}]$ on the owner side, and $2\,\mathcal{S}_\sigma$ on the CSP side.

For determining a dishonest party, the TTP verifies σ_T and σ_F. In case of valid signatures, the TTP proceeds to compute TH_{temp} and FH_{temp}. The values TH_{temp} and FH_{temp} are compared with TH_{TTP} and FH_{TTP}, respectively. Hence, the *maximum* computation overhead on the TTP side due to cheating detection is $2\,\mathcal{V}_\sigma + 2m\,h$.

5.8 Implementation and Experimental Evaluation

5.8.1 Implementation

We have implemented the proposed scheme on top of Amazon Elastic Compute Cloud (Amazon EC2) [5] and Amazon Simple Storage Service (Amazon S3) [6] cloud platforms. Virtual servers (Linux/Unix/Windows) can be launched and managed on Amazon's data centers through Amazon EC2 web service. Amazon S3 is a web service that can be used to store and retrieve almost unlimited amount of data, where customers are enabled to specify geographic locations for their outsourced data.

Our implementation of the proposed scheme consists of four modules: `OModule` (owner module), `CModule` (CSP module), `UModule` (user module), and `TModule` (TTP module). `OModule`, which runs on the owner side, is a library to be used by the owner to perform the owner role in the setup and file preparation phase. Moreover, this library is used by the owner during the dynamic operations on the outsourced data. `CModule` is a library that runs on Amazon EC2 and is used by the CSP to store, update, and retrieve data from Amazon S3. `UModule` is a library to be run at the authorized users' side, and include functionalities that allow users to interact with the TTP and the CSP to retrieve and access the outsourced data. `TModule` is a library used by the TTP to perform the TTP role in the setup and file preparation phase. Moreover, the TTP uses this library during the dynamic operations and to determine the cheating party in the system.

Implementation settings. In our implementation we use a ”large” Amazon EC2 instance to run `CModule`. This instance type provides total memory of size 7.5GB and 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each). One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0 - 1.2GHz 2007 Opteron or 2007 Xeon processor [4]. A separate server in the lab is used to run `TModule`. This server has Intel(R) Xeon(TM) 3.6GHz processor, 2.75GB RAM, and Windows XP operating system. The `OModule` is executed on a desktop computer with Intel(R) Xeon(R) 2GHz processor and 3GB RAM running Windows XP. A laptop with Intel(R) Core(TM) 2.2GHz processor and 4GB RAM running Windows 7 is used to execute the `UModule`. We outsource a data file of size 1GB to Amazon S3. Algorithms (hashing, broadcast encryption, digital signatures, etc.) are implemented using MIRACL library version 5.5.4. For a 128-bit security level, bENC uses an elliptic curve with a 256-bit group order. In the experiments, we utilize SHA-256, 256-bit BLS signature, and Barreto-Naehrig (BN) [13] curve defined over prime field $GF(p)$ with $|p| = 256$ bits and embedding degree = 12 (the BN curve with these parameters is provided by the MIRACL library).

5.8.2 Experimental Evaluation

In this section we describe the experimental evaluation of the computation overhead the proposed scheme brings to a cloud storage system that has been dealing with static data with only confidentiality requirement.

Owner computation overhead. To experimentally evaluate the computation overhead on the owner side due to the dynamic operations, we have performed 100 different block operations with number of authorized users ranging from 20,000 to 100,000. We have run our experiment three times, each time with a different revocation percentage. In the first time, 5% of 100 dynamic operations are executed following revocations. We increased the revocation percentage to 10% for the second time and 20% for the third time. Figure 5.8 shows the owner's average computation overhead per operation. For a large organization (data owner) with 100,000 users, performing dynamic operations and enforcing access control with 5% revocations add about 63 milliseconds of overhead. With 10% and 20% revocation percentages, which are high percentages than an average value in practical applications, the owner overhead is 0.12 and 0.25 seconds, respectively.

Scalability (*i.e.*, how the system performs when more users are added) is an important feature of cloud storage systems. The access control of the proposed scheme depends on the square root of the total number of system users. Figure 5.8 shows that for a large organization with 100,000 users, performing dynamic operations and enforcing access control for outsourced data remains practical.

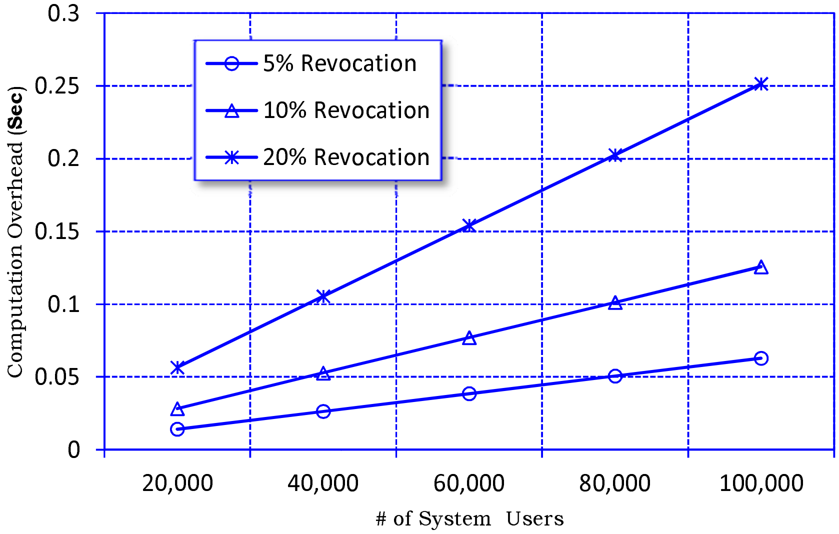


Figure 5.8: Owner's average computation overhead due to dynamic operations.

Table 5.5 shows the computation overheads of the proposed scheme on the TTP, the CSP, and the authorized users sides.

Table 5.5: Experimental results of the computation overheads

Component	TTP	Authorized Users	CSP
Computation Overhead	0.04 ms / 3.59 s	0.55 s	6.04 s

TTP computation overhead. In the worst case, the TTP executes only 4 hashes per dynamic request to reflect the change on the outsourced data. Thus, the maximum computation overhead on the TTP side is about 0.04 milliseconds, *i.e.*, the proposed scheme brings light overhead on the TTP during the normal system operations.

To identify the dishonest party in the system in case of disputes, the TTP verifies two signatures (σ_F and σ_T), computes combined hashes for the data (file and table), and compare the computes hashes with the authentic values (TH_{TTP} and FH_{TTP}). Thus, the computation overhead on the TTP side is about 3.59 seconds. Through our experiments, we use only one desktop computer to simulate the TTP and accomplish its work. The TTP may choose to split the work among a few devices or use a single device with a multi-core processor which is becoming prevalent these days, and thus the computation time on the TTP side is significantly reduced in many applications.

User computation overhead. The computation overhead on the user side due to data access comes from five aspects divided into two groups. The first group involves signatures verification and hash operations to verify the received data (file and table). The second group involves broadcast decryption, backward key rotations, and hash operations to compute the DEK. The first group costs about 5.87 seconds, which can be easily hidden in the receiving time of the data (1GB file and 2MB table).

To investigate the time of the second group, we access the file after running 100 different block operations (with 5% and 10% revocation percentages). Moreover, we implement the backward key rotations in the optimized way. The second group costs about 0.55 seconds, which can be considered as the user's computation overhead due to data access.

CSP computation overhead. As a response to the data access request, the CSP computes two signatures: σ_F and σ_T. Thus, the computation overhead on the CSP side due to data access is about 6.04 seconds and can be easily hidden in the transmission time of the data (1GB file and 2MB table).

5.9 Reducing the Communication Cost

In this section, we discuss a slight modification to the proposed cloud-based storage scheme to reduce the communication cost on the owner and the TTP sides. This slight modification will be applied during the system setup phase and the dynamic operations on the outsourced data. After preparing the BST and the encrypted version $\widetilde{F}$ of the data to be outsourced, the owner sends $\{\widetilde{F}, \text{BST}\}$ to the CSP, and $\{Rot, FH_O, TH_O\}$ to the TTP. The values $FH_O = \oplus_{j=1}^{m} h(\tilde{b}_j)$ and $TH_O = \oplus_{j=1}^{m} h(\mathcal{BN}_j||\mathcal{KV}_j)$ are combined hashes computed by the owner for $\widetilde{F}$ and BST, respectively.

Upon receiving the data from the owner, the CSP computes $FH_C = \oplus_{j=1}^{m} h(\tilde{b}_j)$ and

$TH_C = \oplus_{j=1}^{m} h(\mathcal{BN}_j||\mathcal{KV}_j)$, and sends $\{FH_C, TH_C\}$ to the TTP. The TTP checks $FH_O \overset{?}{=} FH_C$ and $TH_O \overset{?}{=} TH_C$. If they match, the TTP sets $FH_{TTP} = FH_O$ (or FH_C) and $TH_{TTP} = TH_O$ (or TH_C). In case of mismatch, the TTP asks the owner to follow the procedures of the original scheme.

Similar ideas can be applied during the dynamic operations. For example, to modify a data block b_j with b'_j, the owner sends $\langle \mathsf{BM},\ \text{TEntry}_{\mathsf{BM}},\ j,\ \overline{\mathcal{KV}}_j,\ h(\tilde{b}_j),\ \tilde{b}'_j\rangle$ to the CSP, and $\langle \mathsf{BM},\ \text{TEntry}_{\mathsf{BM}},\ \overline{\mathcal{KV}}_j,\ \texttt{RevFlag},\ h_O, h'_O\rangle$ to the TTP, where $h_O = h(\tilde{b}_j)$ and $h'_O = h(\tilde{b}'_j)$. If the CSP accepts the modify request (based on conditions of step 5 in Figure 5.3), it sends h'_C (a hash value computed by the CSP for $\tilde{b}'_j$, *i.e.*, $h'_C = h(\tilde{b}'_j)$) to the TTP. The latter checks $h'_O \overset{?}{=} h'_C$ and if they match, it updates $FH_{TTP} = FH_{TTP} \oplus h_O \oplus h'_O$. The values TH_{TTP} and *Rot* will also be updated as in Figure 5.3. If $h'_O \neq h'_C$, the owner sends the original modify request (step 4 in Figure 5.3) to both the TTP and the CSP.

The slight modification presented here allows the owner to send the outsourced data (or modified blocks) directly to the CSP and hash values along with some metadata to the TTP, which reduces the communication cost on both the owner and the TTP sides. To guarantee the consistency of data between the owner and the CSP, the TTP checks hash values sent from both parties. Only if there is a mismatch, both the TTP and the CSP will receive the data blocks from the owner (the original proposed scheme). The existence of the TTP motivates the owner and the CSP to behave honestly, and thus most of the time they should send equal hash values.

Nevertheless, the discussed modification will cause some extra computation overhead on both the owner and the CSP sides to compute the hash values. Moreover, a slight communication overhead will be imposed on the CSP during the system setup phase and with each dynamic operation to send the computed hash to the TTP.

5.10 Summary

Outsourcing data to remote servers has become a growing trend for many organizations to alleviate the burden of local data storage and maintenance. In this chapter, we have studied different aspects of outsourcing data storage: block-level dynamic operations, newness, mutual trust, and access control.

We have proposed a cloud-based storage scheme which supports outsourcing of dynamic data, where the owner is capable of not only archiving and accessing the data stored by the CSP, but also updating and scaling this data on the remote servers. The proposed scheme enables the authorized users to ensure that they are receiving the most recent version of the outsourced data. Moreover, in case of dispute regarding data integrity/newness, a TTP is able to determine the dishonest party. The data owner enforces access control for the outsourced data by combining three cryptographic techniques: broadcast encryption, lazy revocation, and key rotation. We have studied the security features of the proposed scheme.

In this work, we have investigated the overheads added by the proposed scheme when incorporated into a cloud storage model for *static* data with only *confidentiality* requirement. The storage overhead is $\approx 0.4\%$ of the outsourced data size, the communication overhead due to block-level dynamic changes on the data is $\approx 1\%$ of the block size, and the communication overhead due to retrieving the data is $\approx 0.2\%$ of the outsourced data size. For a large organization (data owner) with 100,000 users, performing dynamic operations and enforcing access control add about 63 milliseconds of overhead. Therefore, important features of outsourcing data storage can be supported without excessive overheads in storage, communication, and computation.

Chapter 6

Conclusions and Future Work

In this chapter, we conclude our dissertation and present future research directions.

6.1 Conclusions

The research area of cloud computing is not merely academic, but has also received considerable attention from industry due to a number of key advantages it offers: cost effectiveness, low management overhead, immediate access to a wide range of applications, flexibility to scale up and down information technology capacity, and mobility where customers can access information wherever they are, rather than having to remain at their desks.

Currently, we are living in a digital world, where a large amount of sensitive data (*e.g.*, personal information, electronic health records, and financial data) is generated by various organizations. Managing such huge amount of data locally at the organization's end is problematic and costly due to the requirements of high storage capacity and qualified personnel. Therefore, cloud service providers (CSPs) offer Storage-as-a-Service as a paid facility to reduce the maintenance cost and mitigates the burden of large local data storage. Through this facility, data owners are enabled to outsource their data to be stored over cloud servers.

For data owners, being not the direct controller over the data raises serious concerns regarding confidentiality, integrity, and access control of the data in cloud computing systems. Provable data possession (PDP) has been introduced as a technique that allows a verifier to efficiently, periodically, and securely validate that a remote server – which supposedly stores the owner's potentially very large amount of data – is actually storing the data in its original form. In this dissertation, we have taken some steps towards mitigating the concerns of outsourcing data storage. These steps can be summarized as follows.

- In Chapter 3, we have addressed the problem of guaranteeing the storage of multiple data copies over untrusted cloud servers. To tackle this problem, we have proposed a pairing-based provable multi-copy data possession (PB-PMDP) scheme that remotely verifies the integrity of multiple data copies stored by the CSP. The proposed scheme considers three main parties: a data owner, a CSP, and authorized users. To seamlessly access any data copy received from the CSP, authorized users decrypt the received copy using a single key shared with the data owner. In that sense, it is not necessarily to recognize which copy has been received. The PB-PMDP scheme has important features including public verifiability, unlimited number of auditing, and possession-free verification where the verifier is enabled to verify data integrity even though he neither possesses nor retrieves the file blocks from the server.

  We have performed security analysis and showed that the proposed PB-PMDP scheme is provably secure against colluding servers. In our analysis we have investigated all possible combinations of malicious CSP responses $\{(\sigma', \mu'), (\sigma, \mu'), (\sigma', \mu)\}$. For the case (σ', μ'), if it is accepted as a valid response, then the CDH (computational Diffie-Hellman) problem can be solved. The response (σ, μ') is rejected, otherwise there is an adversary that can break the DL (discrete logarithm) problem. According to the correctness of verification equation 3.1, (σ', μ) is not accepted unless $\sigma' = \sigma$.

  We have illustrated the performance of the PB-PMDP scheme through theoretical analysis, which is then validated by experimental results. Moreover, a comparative study has been held between the proposed PB-PMDP scheme and the MR-PDP (multiple-replica provable data possession) model. Experimental results have shown that the verification time of PB-PMDP is practically independent of the number of file copies, which makes the scheme computationally cost-effective and more efficient when verifying a large number of file copies.

To recognize which copy has been corrupted in case of failed verification, a slight modification can be applied to the PB-PMDP scheme. This modification utilizes a recursive divide-and-conquer (binary search) approach, where the verifier can identify the indices of corrupted copies. To show the effect of identifying the corrupted copies on the verification time, we have designed some experiments by generating data copies and randomly corrupting different percentages of these copies. Interestingly, when the percentage of corrupted copies is up to 15% of the total copies, the performance of using the binary search algorithm in the verification is more efficient than individual verification for each copy.

- In Chapter 4, we have studied creating multiple copies of dynamic data file and verifying those copies stored on untrusted cloud servers. We have proposed a new PDP scheme referred to as MB-PMDDP (map-based provable multi-copy dynamic data possession), which supports outsourcing of dynamic data, *i.e.*, it supports *block-level* operations such as block modification, insertion, deletion, and append. The proposed MB-PMDDP scheme provides a guarantee that the CSP is storing all data copies that are agreed upon in the service contract, and all these copies are consistent with the most recent modifications issued by the owner. To the best of our knowledge, MB-PMDDP is the first to address the integrity verification of multiple copies of dynamic data. The MB-PMDDP scheme enables unlimited number of auditing, considers the interaction between authorized users and the CSP (*i.e.*, users can seamlessly access a data copy received from the CSP using a single secret key shared with the data owner), supports public verifiability, and allows possession-free verification.

  To verify dynamic data, a verifier needs to be aware of block versions and indices. Therefore, the proposed MB-PMDDP scheme is based on using a small data structure, which we call a map-version table (MVT). The MVT stores the version of each block, and updates this version with each block modification operation. In addition, it keeps a mapping between logical block numbers and their physical positions in the data file. It is important to note that the verifier retains only *one* table for unlimited number of file copies, which mitigates the storage overhead on the verifier side.

Homomorphic linear authenticators are basic building blocks of the MB-PMDDP scheme, and their unforgeability is the base to prove the security of the proposed scheme. In our security analysis, we have shown that only correctly computed proof $\mathbb{P} = \{\sigma,\ \mu\}$ is accepted as a valid response to a challenge vector sent from a verifier, and thus the MB-PMDDP scheme is provably secure against colluding servers.

To illustrate the performance of the proposed MB-PMDDP scheme, we have presented an extension – labelled as TB-PMDDP (tree-based provable multi-copy dynamic data possession) – to provable possession models for single-copy dynamic data to work in the setting of multiple copies of dynamic data. We have performed a comparative study between the MB-PMDDP and the TB-PMDDP schemes. Theoretical analysis, implementation, and experimental results have demonstrated that the proposed MB-PMDDP scheme outperforms the TB-PMDDP approach from many perspectives: storage overhead on the CSP side, computation cost on both the CSP and the verifer sides, and communication cost for the CSP's response and dynamic block operations.

As we have done with the PB-PMDP scheme for static data, the proposed MB-PMDDP scheme can be slightly modified to support the feature of identifying the indices of corrupted copies. We have also designed experiments to show the effect of identifying the corrupted copies on the verification time. The experiments have indicated that up to 15% corruption percentage, applying divide-and-conquer approach during verification is more efficient than individual verification for each copy.

- In Chapter 5, we have complemented our research by proposing a new cloud-based storage scheme that allows the data owner to benefit from facilities offered by the CSP and enables indirect mutual trust between them. The proposed scheme have considered important aspects of outsourcing data storage: block-level dynamic operations, newness, mutual trust, and access control. It enables data owners to release their concerns regarding confidentiality, integrity, and access control of the outsourced data. Moreover, the CSP is protected from any false accusation that may be claimed by a dishonest owner to get some sort of compensation.

The proposed scheme allows authorized users to make sure that they are receiving the most recent version of the outsourced data. To resolve disputes that may occur regarding data integrity/newness, a trusted third party is invoked to determine the dishonest side (owner/users or CSP). In addition, our scheme combines three cryptographic techniques: broadcast encryption, lazy revocation, and key rotation to enforce access control for outsourced data.

We have studied the security features of the proposed scheme, and showed that the scheme satisfies: (i) data confidentiality based on the security of underlying encryption algorithm, (ii) detection of data integrity violation based on the preimage and second-preimage resistance properties of the utilized cryptographic hash function, (iii) assurance of newness property, which is identical to detection of data integrity violation, (iv) enforcement of access control based on combining broadcast encryption, lazy revocation, and key rotation; and (v) detection of dishonest owner/user based on unforgeable signatures.

The performance of the proposed scheme has been justified through theoretical analysis and a prototype implementation on Amazon cloud platform to evaluate storage, communication, and computation overheads. We have showed that important features of outsourcing data storage can be supported without excessive overheads.

6.2 Future Research Directions

The area of cloud computing has attracted many researchers from diverse fields; however, much effort remains to achieve the wide acceptance and usage of cloud computing technology. A number of future research directions stem from our current research. Below, we summarize some problems to address during our future research.

Ensuring data replication across diverse geographic location. In this dissertation work, we have proposed schemes to verify that the CSP is actually storing all data copies that are agreed upon in the service contact. Replicating data in different geographic locations is crucial to prevent simultaneous failure caused by natural disasters or power outages. Moreover, it is effective in reducing access time and communication cost for users in different parts in the world.

It will be interesting to study the problem of verifying that the data is actually replicated in diverse geolocations. This will require collaboration between researchers from both industry (to build data center components, services, and software) and academia (to provide mathematical models and theoretical frameworks for the verification process).

Self-organized data replication over cloud servers. Current data centers are subject to failure of any type, and high access to the data stored can be one reason for such failure. As the number of access requests to outsourced data increases, its availability becomes more complex. For example, the University of Waterloo has an online course system (LEARN) based on cloud computing technology. During exam days, almost all students access course materials on LEARN, which might affect data availability.

One possible future direction is to address the problem of designing self-managed storage systems that can dynamically adapt to varying query load by allocating/deallocating storage space for data copies on cloud servers. An optimization model is needed to specify the optimal number of copies and their storage locations across the servers. Through this model one can minimize the response time for data access requests, and optimize the use of CSP's storage capacity.

User authentication for cloud computing systems. The development of cloud computing encourages the use of resource-constrained devices (PDA/cell phones) on the client side. Thus, rather than local data storage and software installation, users will be authenticated to access data and use applications from the cloud. Such computing model makes software piracy more difficult and enables centralized monitoring. Although cloud computing architecture stimulates mobility of users, it increases the need of secure authentication.

Relying on passwords for user authentication in not an efficient approach for sensitive data/applications on the cloud. Passwords is a major point of vulnerability in computer security; they are often easy to guess by automated programs running dictionary attacks, users cannot remember very long passwords, and the common use of meaningful passwords makes them subject to dictionary attacks.

Implicit authentication is another interesting area of research to address user authentication problem. One can use learning algorithms to construct a model for the user based on previous behavior patterns, and then compare the recent behavior with the user model to authorize legitimate users. This may require collaboration with researchers from computer science to develop efficient learning algorithms using artificial intelligence, machine learning, and neural networks tools.

Outsourcing computation to untrusted cloud servers. Outsourcing computation is a growing desire for resource-constrained clients to benefit from powerful cloud servers. Such clients prefer to outsource computationally-intensive operations (*e.g.*, image processing) to the cloud and yet obtaining a strong assurance that the computations are correctly performed. To save the computational resources, a dishonest CSP may totally ignore the computations, or execute just a portion of them. Sometimes the computations outsourced to the cloud are so critical that it is essential to preclude accidental errors during the processing.

The ability to verify computations and validate the returned results is a key requirement of cloud customers. Another imperative point is that the amount of work performed by the clients to verify the outsourced computations must be substantially cheaper than performing the actual computations on the client side. One direction of future research is to investigate the area of verifiable computations and outsourcing computational tasks to untrusted cloud servers. It is also interesting to address mutual trust feature, so a client who receives incorrect results from cloud servers can detect and prove this misbehavior. Moreover, a dishonest client must not be able to falsely accuse a CSP and claim that the outsourced computations are malformed.

Appendix A

Examples of the TB-PMDDP scheme

This appendix contains some examples of the TB-PMDDP scheme that demonstrate the effect of dynamic operations on the MHTs over the CSP side. Moreover, these examples show how the owner uses the information received from the CSP to generate the new directory root and update the metadata. We assume that the data owner has a file of 4 blocks and the CSP stores n copies of this file. Also we assume that during the system setup, the owner and the CSP have agreed to use left-to-right sequence to generate the Merkle trees.

- ♦ **Modification**. Figure A.1 shows that the second block is to be modified in all copies outsourced to the CSP. *On the CSP side*, the dashed nodes indicate the tree nodes that are updated due to the modification of the second block. The dashed *leaf* nodes $\{h_{12}, h_{22}, \ldots, h_{n2}\}$ are updated as $h_{i2} = h(\mathcal{H}(\tilde{b}'_{i2}))$, where $\tilde{b}'_{i2}$ – created and sent from the owner – is the modified second block dedicated for copy i: $1 \leq i \leq n$. The dashed *non-leaf* nodes are updated by h(left child || right child). The grey nodes indicate the authentication paths of the modified blocks, *e.g.*, $\{h_{11}, h_{1B}\}$ is the authentication path of the modified block $\tilde{b}'_{12}$. *On the owner side*, the owner uses the authentication paths $\langle\{h_{11}, h_{1B}\}, \{h_{21}, h_{2B}\}, \ldots, \{h_{n1}, h_{nB}\}\rangle$ sent from the CSP and the modified blocks $\{\tilde{b}'_{12}, \tilde{b}'_{22}, \ldots, \tilde{b}'_{n2}\}$ to generate the new directory root h'_{DR} and update the metadata $\mathcal{M}' = h(ID_F||h'_{DR})$. The dashed nodes indicate the generated cryptographic hashes using the authentication paths (grey circles) sent from the CSP and the modified blocks $\{\tilde{b}'_{i2}\}_{1\leq i\leq n}$. Thus, $h_{12} = h(\mathcal{H}(\tilde{b}'_{12}))$, $h_{1A} = h(h_{11}||h_{12})$, and $h_{1R} = h(h_{1A}||h_{1B})$. The computation of $\{h_{iR}\}_{2\leq i\leq n}$ is done the same way. The owner uses the computed $\{h_{iR}\}_{1\leq i\leq n}$ to generate the updated directory root h'_{DR}, and finally computes the updated metadata $\mathcal{M}' = h(ID_F||h'_{DR})$.

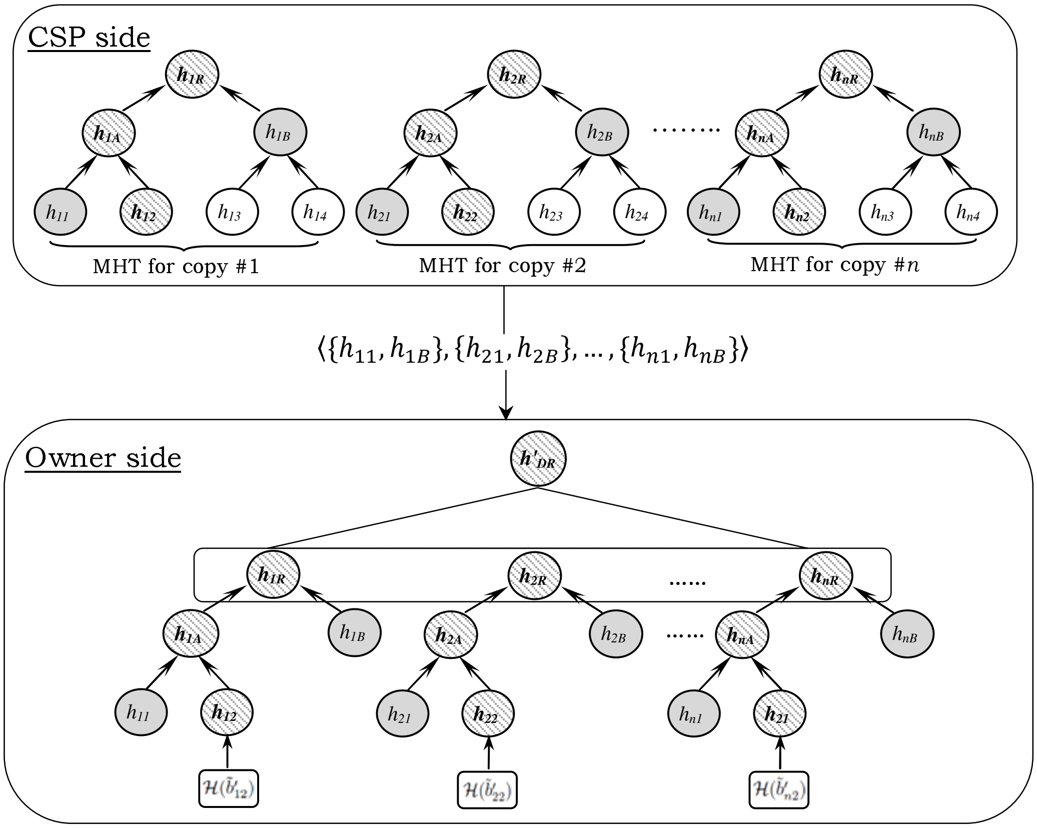


Figure A.1: Effect of block modification operation on the MHTs and the directory root.

- ⧫ **Insertion**. Figure A.2 shows that a new block is to be inserted after position 2 in all copies outsourced to the CSP. *On the CSP side*, the cross-dashed nodes indicate the newly added leaf nodes, *i.e.*, $\hat{h}_{i2} = h(\mathcal{H}(\hat{b}_i))$, where $\hat{b}_i$ – created and sent from the owner – is the new block to be inserted in copy i: $1 \leq i \leq n$. The dashed nodes indicate the tree nodes that are updated due to the insertion of the new block. The updated hash values of these nodes are computed as h(left child||right child).

The nodes $\{\hat{h}_{iC}\}_{1\leq i\leq n}$ are generated to re-arrange the structure of the MHTs according to the newly added leaf nodes. The grey nodes indicate the authentication paths of the newly inserted blocks, *e.g.*, $\{h_{12}, h_{11}, h_{1B}\}$ is the authentication path of the new block $\hat{b}_1$. *On the owner side*, the owner uses the authentication paths $\langle\{h_{12}, h_{11}, h_{1B}\}, \{h_{22}, h_{21}, h_{2B}\}, \ldots, \{h_{n2}, h_{n1}, h_{nB}\}\rangle$ sent from the CSP and the new blocks $\{\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_n\}$ to generate the new directory root h'_{DR} and update the metadata $\mathcal{M}' = h(ID_F||h'_{DR})$. The dashed nodes indicate the generated cryptographic hashes using the authentication paths (grey circles) sent from the CSP and the new blocks $\{\hat{b}_i\}_{1\leq i\leq n}$. Thus, $\hat{h}_1 = h(\mathcal{H}(\hat{b}_1)), \hat{h}_{1C} = h(h_{12}||\hat{h}_1), h_{1A} = h(h_{11}||\hat{h}_{1C})$, and $h_{1R} = h(h_{1A}||h_{1B})$. The computation of $\{h_{iR}\}_{2\leq i\leq n}$ is done the same way. The owner uses the computed $\{h_{iR}\}_{1\leq i\leq n}$ to generate the updated directory root h'_{DR}, and finally computes the updated metadata $\mathcal{M}' = h(ID_F||h'_{DR})$.

- **Deletion**. Figure A.3 shows that the second block is to be deleted from all copies outsourced to the CSP. *On the CSP side*, the leaf nodes with crosses indicate the nodes to be deleted. The fragmented curved arrows indicate that after deleting the specified leaf nodes, the nodes $\{h_{i1}\}_{1\leq i\leq n}$ replace the nodes $\{h_{iA}\}_{1\leq i\leq n}$, and thus the MHTs are re-arranged. The dashed nodes indicate the tree nodes that are updated due to the deletion of the second block. The updated hash values of these nodes are computed as $h(\text{left child}||\text{right child})$. The grey nodes indicate the authentication paths of the deleted blocks, *e.g.*, $\{h_{11}, h_{1B}\}$ is the authentication path of the deleted block $\tilde{b}_{12}$. *On the owner side*, the owner uses the authentication paths $\langle\{h_{11}, h_{1B}\}, \{h_{21}, h_{2B}\}, \ldots, \{h_{n1}, h_{nB}\}\rangle$ sent from the CSP to generate the new directory root h'_{DR} and update the metadata $\mathcal{M}' = h(ID_F||h'_{DR})$. The dashed nodes indicate the generated cryptographic hashes using the authentication paths (grey circles) sent from the CSP. Thus, $h_{iR} = h(h_{i1}||h_{iB}) : 1 \leq i \leq n$. The owner uses the computed $\{h_{iR}\}_{1\leq i\leq n}$ to generate the updated directory root h'_{DR}, and finally computes the updated metadata $\mathcal{M}' = h(ID_F||h'_{DR})$.

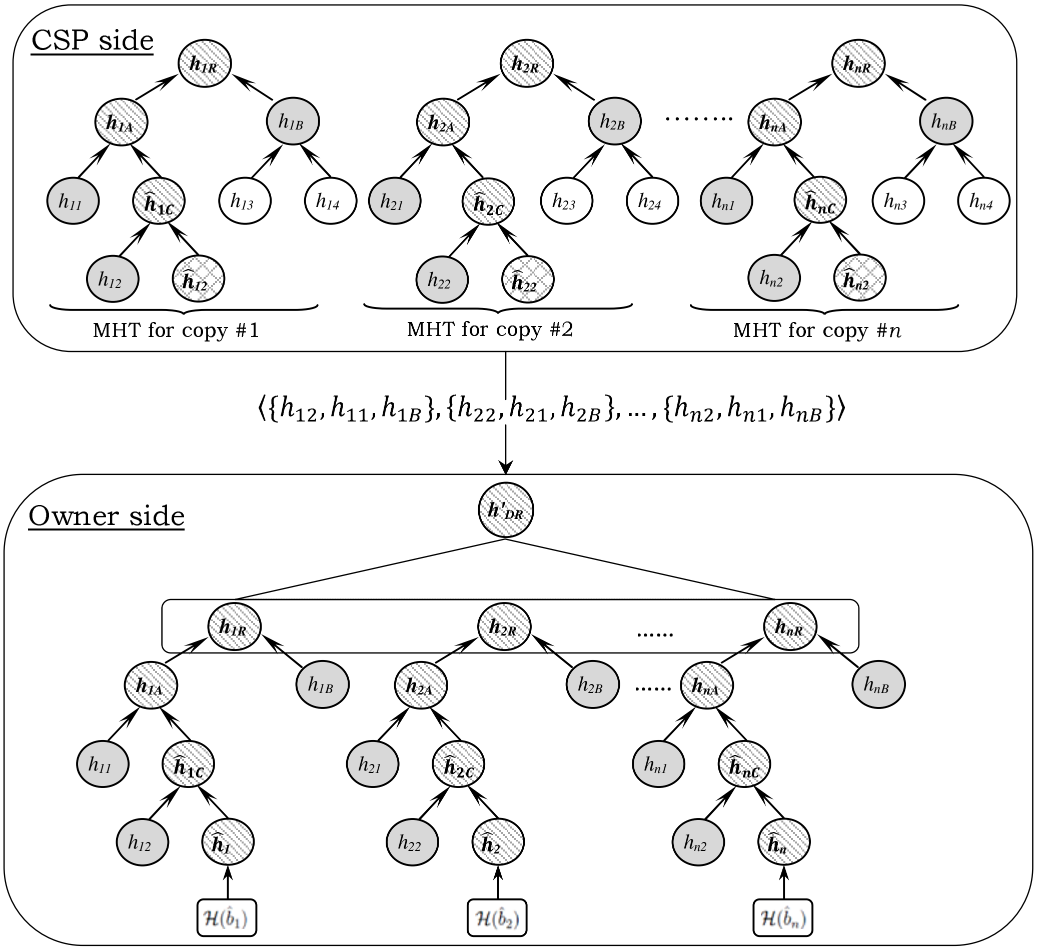


Figure A.2: Effect of block insertion operation on the MHTs and the directory root.

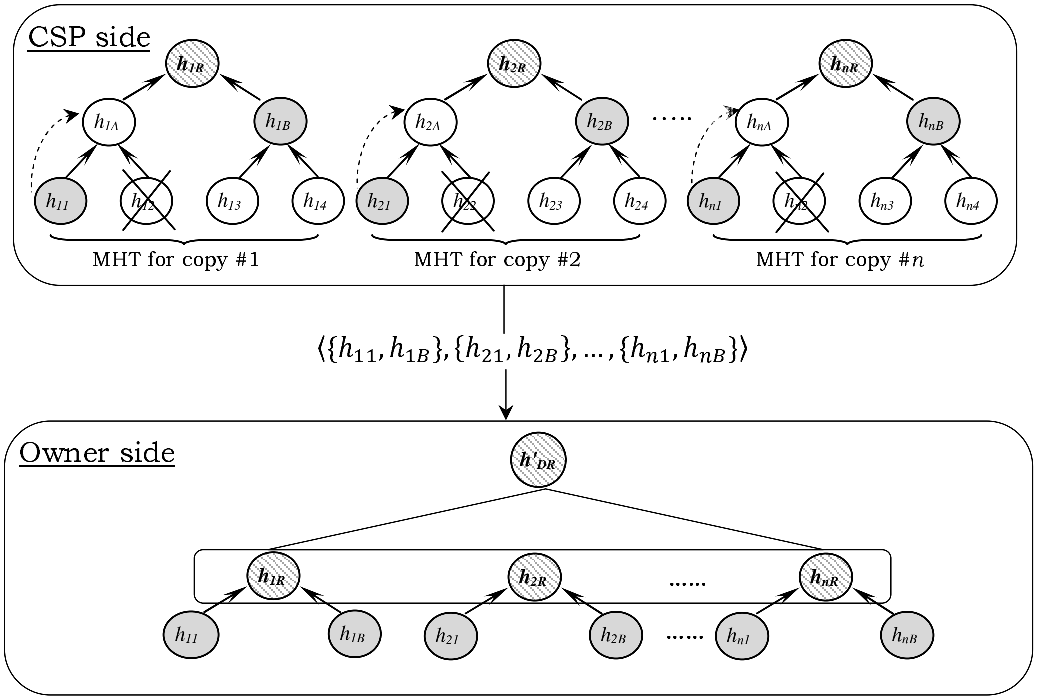


Figure A.3: Effect of block deletion operation on the MHTs and the directory root.

References

[1] FIPS PUB 197, Advanced Encryption Standard (AES), National Institute of Standards and Technology, U.S. Department of Commerce, November 2001.

[2] 104th United States Congress. Health Insurance Portability and Accountability Act of 1996 (HIPAA). Online at http://aspe.hhs.gov/admnsimp/pl104191.htm, 1996.

[3] Marcos K. Aguilera, Ramaprabhu Janakiraman, and Lihao Xu. Using erasure codes efficiently for storage in a distributed system. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks*, DSN '05, pages 336–345, Washington, DC, USA, 2005. IEEE Computer Society.

[4] Amazon EC2 Instance Types. http://aws.amazon.com/ec2/.

[5] Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/.

[6] Amazon Simple Storage Service (Amazon S3). http://aws.amazon.com/s3/.

[7] Mikhail J. Atallah, Keith B. Frikken, and Marina Blanton. Dynamic and efficient key management for access hierarchies. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, CCS '05, pages 190–202. ACM, 2005.

[8] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *CCS '07: Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 598–609, New York, NY, USA, 2007.

[9] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *Proceedings of the Network and Distributed System Security Symposium, NDSS*. The Internet Society, 2005.

[10] Giuseppe Ateniese, Seny Kamara, and Jonathan Katz. Proofs of storage from homomorphic identification protocols. In *ASIACRYPT '09: Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security*, pages 319–333, Berlin, Heidelberg, 2009.

[11] Giuseppe Ateniese, Roberto Di Pietro, Luigi V. Mancini, and Gene Tsudik. Scalable and efficient provable data possession. In *SecureComm '08: Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks*, pages 1–10, New York, NY, USA, 2008.

[12] Michael Backes, Christian Cachin, and Alina Oprea. Secure key-updating for lazy revocation. In *Computer Security - ESORICS 2006, 11th European Symposium on Research in Computer Security*, Lecture Notes in Computer Science, pages 327–346. Springer, 2006.

[13] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In *Proceedings of SAC 2005, volume 3897 of LNCS*, pages 319–331. Springer-Verlag, 2005.

[14] Paulo S. L. M. Barreto and Michael Naehrig. IEEE P1363.3 submission: Pairing-friendly elliptic curves of prime order with embedding degree 12. New Jersey: IEEE Standards Association, 2006.

[15] Ayad F. Barsoum and M. Anwar Hasan. Provable multi-copy data possession for cloud computing storage systems. *submitted to a journal*.

[16] Ayad F. Barsoum and M. Anwar Hasan. Provable multi-copy dynamic data possession in cloud computing systems. *submitted to a journal*.

[17] Ayad F. Barsoum and M. Anwar Hasan. Enabling dynamic data and indirect mutual trust for cloud computing storage systems. Centre For Applied Cryptographic Research (CACR), University of Waterloo, Report 2012/05, 2010. http://cacr.uwaterloo.ca/techreports/2012/cacr2012-05.pdf.

[18] Ayad F. Barsoum and M. Anwar Hasan. Provable possession and replication of data over cloud servers. Centre For Applied Cryptographic Research (CACR), University of Waterloo, Report 2010/32, 2010. http://www.cacr.math.uwaterloo.ca/techreports/2010/cacr2010-32.pdf.

[19] Ayad F. Barsoum and M. Anwar Hasan. On verifying dynamic multiple data copies over cloud servers. Cryptology ePrint Archive, Report 2011/447, 2011, 2011. http://eprint.iacr.org/.

[20] Ayad F. Barsoum and M. Anwar Hasan. Integrity verification of multiple data copies over untrusted cloud servers. In *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2012*, pages 829–834. IEEE Computer Society, 2012.

[21] Ayad F. Barsoum and M. Anwar Hasan. Enabling dynamic data and indirect mutual trust for cloud computing storage systems. *IEEE Transactions on Parallel and Distributed Systems*, 99(PrePrints):1, 2013.

[22] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 321–334. IEEE Computer Society, 2007.

[23] Matt Blaze. A cryptographic file system for unix. In *ACM Conference on Computer and Communications Security*, pages 9–16, 1993.

[24] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT*, pages 127–144, 1998.

[25] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 213–229, London, UK, UK, 2001. Springer-Verlag.

[26] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *Advances in Cryptology - CRYPTO*, pages 258–275, 2005.

[27] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, pages 514–532, London, UK, 2001.

[28] Dan Boneh and Ramarathnam Venkatesan. Breaking RSA may not be equivalent to factoring. In *EUROCRYPT*, pages 59–71, 1998.

[29] Kevin D. Bowers, Ari Juels, and Alina Oprea. HAIL: A High-Availability and Integrity Layer for Cloud Storage. In *CCS '09: Proceedings of the 16th ACM conference on*

Computer and communications security, pages 187–198, New York, NY, USA, 2009. ACM.

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

[31] 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.

[32] Saikat Chakrabarti and Mukesh Singhal. Password-based authentication: Preventing dictionary attacks. *Computer*, 40:68–74, 2007.

[33] Richard Chow, Philippe Golle, Markus Jakobsson, Elaine Shi, Jessica Staddon, Ryusuke Masuoka, and Jesus Molina. Controlling data in the cloud: Outsourcing computation without outsourcing control. In *CCSW '09: Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, pages 85–90, New York, NY, USA, 2009.

[34] 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 *NSDI'06: Proceedings of the 3rd Conference on Networked Systems Design & Implementation*, Berkeley, CA, USA, 2006.

[35] Claude Elwood Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656715.

[36] Reza Curtmola, Osama Khan, and Randal Burns. Robust remote data checking. In *StorageSS '08: Proceedings of the 4th ACM international workshop on Storage security and survivability*, pages 63–68, New York, NY, USA, 2008. ACM.

[37] Reza Curtmola, Osama Khan, Randal Burns, and Giuseppe Ateniese. MR-PDP: multiple-replica provable data possession. In *28th IEEE International Conference on Distributed Computing Systems, ICDCS*, pages 411–420, 2008.

[38] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES – The Advanced Encryption Standard*. Springer-Verlag, 2002, ISBN = 3-540-42580-2.

[39] Yves Deswarte, Jean-Jacques Quisquater, and Ayda Saïdane. Remote integrity checking. In Sushil Jajodia; Leon Strous, editor, *6th Working Conference on Integrity and Internal Control in Information Systems (IICIS)*, pages 1–11, 2003.

[40] Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Over-encryption: Management of access control evolution on outsourced data. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, pages 123–134. ACM, 2007.

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

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

[43] Extended Gmail Outage Hits Apps Admins. http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9117322.

[44] Jun Feng, Yu Chen, Wei-Shinn Ku, and Pu Liu. Analysis of integrity vulnerabilities and a non-repudiation protocol for cloud data storage platforms. In *Proceedings of the 2010 39th International Conference on Parallel Processing*, ICPP '10, pages 251–258. IEEE Computer Society, 2010.

[45] Jun Feng, Yu Chen, and Douglas H. Summerville. A fair multi-party non-repudiation scheme for storage clouds. In *2011 International Conference on Collaboration Technologies and Systems*, CTS 2011, pages 457–465, 2011.

[46] Anna Lisa Ferrara, Matthew Green, Susan Hohenberger, and Michael Pedersen. Practical short signature batch verification. In *The Cryptographer's Track at RSA Conference*, pages 309–324, 2009.

[47] Amos Fiat and Moni Naor. Broadcast encryption. In *Proceedings of the 13th annual international cryptology conference on Advances in cryptology*, pages 480–491. Springer-Verlag New York, Inc., 1994.

[48] Décio Luiz Gazzoni Filho and Paulo Sérgio Licciardi Messeder Barreto. Demonstrating data possession and uncheatable data transfer. Cryptology ePrint Archive, Report 2006/150, 2006.

[49] Kevin E. Fu. Group sharing and random access in cryptographic storage file systems. Technical report, Master's thesis, MIT, 1999.

[50] Eu-Jin Goh, Hovav Shacham, Nagendra Modadugu, and Dan Boneh. Sirius: Securing remote untrusted storage. In *Proceedings of the Network and Distributed System Security Symposium, NDSS*. The Internet Society, 2003.

[51] Nancy Gohring. Amazon's S3 down for several hours. Online at http://www.pcworld.com/businesscenter/article/142549/amazons_s3_down_for_severalhours.html, 2008.

[52] Philippe Golle, Stanislaw Jarecki, and Ilya Mironov. Cryptographic primitives enforcing communication and storage complexity. In *FC'02: Proceedings of the 6th International Conference on Financial Cryptography*, pages 120–135, Berlin, Heidelberg, 2003.

[53] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS '06, pages 89–98. ACM, 2006.

[54] Zhuo Hao, Sheng Zhong, and Nenghai Yu. A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability. *IEEE Transactions on Knowledge and Data Engineering*, 99(PrePrints), 2011.

[55] IDC Enterprise Panel, 2010.

[56] Ari Juels and Burton S. Kaliski. PORs: Proofs of Retrievability for large files. In *CCS'07: Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 584–597. ACM, 2007.

[57] Mahesh Kallahalla, Erik Riedel, Ram Swaminathan, Qian Wang, and Kevin Fu. Plutus: Scalable secure file sharing on untrusted storage. In *Proceedings of the FAST 03 Conference on File and Storage Technologies*. USENIX, 2003.

[58] Brian Krebs. Payment processor breach may be largest ever. Online at http://voices.washingtonpost.com/securityfix/2009/01/payment_processor_breach_may_b.html, Jan. 2009.

[59] Miroslav Lassak and Stefan Porubsky. Fermat-Euler theorem in algebraic number fields. *Journal of Number Theory*, 60(2):254–290, 1996.

[60] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. Dynamic authenticated index structures for outsourced databases. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pages 121–132, New York, NY, USA, 2006.

[61] Petros Maniatis, Mema Roussopoulos, T. J. Giuli, David S. H. Rosenthal, and Mary Baker. The LOCKSS Peer-to-Peer Digital Preservation System. *ACM Trans. Comput. Syst.*, 23(1):2–50, 2005.

[62] Chip Martel, Glen Nuckolls, Prem Devanbu, Michael Gertz, April Kwong, and Stuart G. Stubblebine. A general model for authenticated data structures. *Algorithmica*, 39, 2001.

[63] David Mazières, Michael Kaminsky, M. Frans Kaashoek, and Emmett Witchel. Separating key management from file system security. In *Proceedings of the seventeenth ACM symposium on Operating systems principles*, SOSP '99, pages 124–139. ACM, 1999.

[64] Peter Mell and Tim Grance. Draft NIST working definition of cloud computing. Online at http://csrc.nist.gov/groups/SNS/cloud-computing/index.html, 2009.

[65] Alfred Menezes. An introduction to pairing-based cryptography. Lecture Notes 2005, Online at http://www.math.uwaterloo.ca/~ajmeneze/publications/pairings.pdf.

[66] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., 1997.

[67] Ralph C. Merkle. Protocols for public key cryptosystems. *IEEE Symposium on Security and Privacy*, 0:122, 1980.

[68] Atsuko Miyaji, Masaki Nakabayashi, and Shunzou TAKANO. New explicit conditions of elliptic curve traces for FR-reduction. *IEICE Transactions on fundamental*, pages 1234–1243, 2001.

[69] Einar Mykletun, Maithili Narasimha, and Gene Tsudik. Authentication and integrity in outsourced databases. *Trans. Storage*, 2(2), 2006.

[70] Dalit Naor, Moni Naor, and Jeffrey B. Lotspiech. Revocation and tracing schemes for stateless receivers. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 41–62. Springer-Verlag, 2001.

[71] Shivaramakrishnan Narayan, Martin Gagné, and Reihaneh Safavi-Naini. Privacy preserving EHR system using attribute-based infrastructure. In *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, CCSW '10, pages 47–52. ACM, 2010.

[72] Jose Antonio Onieva, Javier Lopez, and Jianying Zhou. *Secure Multi-Party Non-Repudiation Protocols and Applications*, volume 43 of *Advances in Information Security.* Springer, 2009.

[73] Raluca Ada Popa, Jacob R. Lorch, David Molnar, Helen J. Wang, and Li Zhuang. Enabling security in cloud storage SLAs with cloudproof. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, USENIXATC'11. USENIX Association, 2011.

[74] William Pugh. Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33:668–676, 1990.

[75] Erik Riedel, Mahesh Kallahalla, and Ram Swaminathan. A framework for evaluating storage system security. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, FAST '02. USENIX Association, 2002.

[76] Ronald Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 26(1), 1983.

[77] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *FSE: Fast Software Encryption*, pages 371–388, 2004.

[78] A. Ruiz-Martínez, C. I. Marín-López, L. Baño López, and A. F. Gómez Skarmeta. A new fair non-repudiation protocol for secure negotiation and contract signing. In *Proceedings of the 2006 International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services*, PST '06. ACM, 2006.

[79] Thomas S. J. Schwarz and Ethan L. Miller. Store, forget, and check: Using algebraic signatures to check remotely administered storage. In *ICDCS '06: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, Washington, DC, USA, 2006.

[80] Francesc Sebé, Josep Domingo-Ferrer, Antoni Martinez-Balleste, Yves Deswarte, and Jean-Jacques Quisquater. Efficient remote data possession checking in critical information infrastructures. *IEEE Transactions on Knowledge and Data Engineering*, 20(8), 2008.

[81] 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: Advances in Cryptology*, ASIACRYPT '08, pages 90–107. Springer-Verlag, 2008.

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

[83] Mehul A. Shah, Ram Swaminathan, and Mary Baker. Privacy-preserving audit and extraction of digital contents. Cryptology ePrint Archive, Report 2008/186, 2008.

[84] Victor Shoup. On the security of a practical identification scheme. In *EUROCRYPT'96: Proceedings of the 15th Annual International Conference on Theory and Application of Cryptographic Techniques*, pages 344–353, Berlin, Heidelberg, 1996.

[85] Aameek Singh and Ling Liu. Sharoes: A data sharing platform for outsourced enterprise storage environments. In *Proceedings of the 24th International Conference on Data Engineering, ICDE*, pages 993–1002. IEEE, 2008.

[86] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for data storage security in cloud computing. In *INFOCOM*, pages 525–533, 2010.

[87] Qian Wang, Cong Wang, Jin Li, Kui Ren, and Wenjing Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *ESORICS'09: Proceedings of the 14th European Conference on Research in Computer Security*, pages 355–370, Berlin, Heidelberg, 2009.

[88] Weichao Wang, Zhiwei Li, Rodney Owens, and Bharat Bhargava. Secure and efficient access to outsourced data. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, CCSW '09, pages 55–66. ACM, 2009.

[89] Min Xie, Haixun Wang, Jian Yin, and Xiaofeng Meng. Integrity auditing of outsourced data. In *VLDB '07: Proceedings of the 33rd International Conference on Very Large Databases*, pages 782–793, 2007.

[90] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *Proceedings of the 29th conference on Information communications*, INFOCOM'10, pages 534–542. IEEE Press, 2010.

[91] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Attribute based data sharing with attribute revocation. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 261–270. ACM, 2010.

[92] Ke Zeng. Publicly verifiable remote data integrity. In *Proceedings of the 10th International Conference on Information and Communications Security*, ICICS '08, pages 419–434, Berlin, Heidelberg, 2008. Springer-Verlag.

[93] Yan Zhu, Huaixi Wang, Zexing Hu, Gail-Joon Ahn, Hongxin Hu, and Stephen S. Yau. Efficient provable data possession for hybrid clouds. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 756–758, New York, NY, USA, 2010. ACM.